Download Chapter pdf

Download Chapter notebook (ipynb)

Mandatory Lesson Feedback Survey

- What is bivariate or multivariate analysis?
- How are bivariate properties of data interpreted?
- How can a bivariate quantity be explained?
- When to use the correlation matrix?
- What are the ways to study relationships in data?

- Practise working with Pandas dataframes and Numpy arrays.
- Bivariate analysis of Pandas dataframe / Numpy array.
- The Pearson correlation coefficient (PCC).
- Correlation Matrix as an example of bivariate summary statistics.

Prerequisites

- Python Arrays
- Basic Statistics, in particular the correlation coefficient
- Pandas dataframes: import and handling

The following cell contains functions that need to be imported, please execute it before continuing with the Introduction.

```
# To import data from a csv file into a Pandas dataframe
from pandas import read_csv

# To import a dataset from scikit-learn
from sklearn import datasets

# To create figure environments and plots
from matplotlib.pyplot import subplots, show

# Specific numpy functions, description in the main body
from numpy import corrcoef, fill_diagonal, triu_indices, arange
```

Note

In many online tutorials you can find the following convention when importing functions:

```
import numpy as np
import pandas as pd
```

(or similar). In this case, the whole library is imported and any function in that library is then available using e.g. `pd.read_csv(my_file)`

We don't recommend this as the import of the whole library uses a lot of working memory (e.g. on the order of 100 MB for Numpy).

Introduction

In the previous lesson we have obtained some basic data quantifications using `describe`. Each of these quantities was calculated for individual columns (where each column contained a different measured variable). However, in data analysis in general, and especially in machine learning, the main point of analysis is to also try and exploit the presence of information that lies in relationships *between* variables, i.e. columns in our data.

Quantities that are based on data from two variables are referred to as **bivariate** measures. Analysis that makes use of bivariate (and potentially higher order) quantities is referred to as bivariate or (in general) **multivariate data analysis**.

When we combine uni- and multivariate analysis we can get an excellent overview of basic properties of a dataset.

Example: The diabetes data set

Using the diabetes dataset (introduced in the previous lesson), let us look at the data from three of its columns: The upper row of the below figure shows three histograms. A histogram is a summary plot of the recordings of a single variable. The histograms of columns with indices 3, 4, and 5 have similar means and variances but that is due to prior normalisation. The shapes differ but this does not tell us anything about a relationship between the measurements.

One thing that we want to know before we start to apply any machine learning is whether or not there is evidence of relationships between the individual variables in a dataframe. One of the potential relationships is that the variables are 'similar'. One way to check for the similarity between variables in a dataset is to create a scatter plot. The bottom row of the figure below contains the three scatter plots between variables used to create the histograms in the top row.

(Please execute the code to create the figures. We will describe the scatter plot and its features later on.)

```
# Figure Code

diabetes = datasets.load_diabetes()

```

```
diabetes_data = diabetes.data

fig, ax = subplots(figsize=(21, 10), ncols=3, nrows=2)

# Histograms
ax[0,0].hist(diabetes_data[:,3], bins=20)
ax[0,0].set_ylabel('Count', fontsize=20)

ax[0,1].hist(diabetes_data[:,4], bins=20)
ax[0,1].set_ylabel('Count', fontsize=20)

ax[0,2].hist(diabetes_data[:,5], bins=20)
ax[0,2].set_ylabel('Count', fontsize=20)

# Scatter plots
ax[1,0].scatter(diabetes_data[:,3], diabetes_data[:,4]);
ax[1,0].set_xlabel('Column 3', fontsize=20)
ax[1,0].set_ylabel('Column 4', fontsize=20)

ax[1,1].scatter(diabetes_data[:,4], diabetes_data[:,5]);
ax[1,1].set_xlabel('Column 4', fontsize=20)
ax[1,1].set_ylabel('Column 5', fontsize=20)

ax[1,2].scatter(diabetes_data[:,5], diabetes_data[:,3]);
ax[1,2].set_xlabel('Column 5', fontsize=20)
ax[1,2].set_ylabel('Column 3', fontsize=20);

show()
```

When plotting the data against each other in pairs (lower row), data column 3 versus column 4 (left) and column 5 versus 3 (right) both show a fairly uniform circular distribution of points. This is what would be expected if the data in the two columns were independent of each other.

In contrast, column 4 versus 5 (centre) shows an elliptic, pointed shape along the main diagonal. This shows that something particular goes on between these data sets. Specifically, it indicates that the two variables recorded in these columns (indices 4 and 5) are not independent of each other. They are more similar than would be expected for independent variables.

In this lesson we aim to get an overview of the similarities in a data set. We first introduce bivariate visualisation using Matplotlib. Then we use Numpy functions to calculate correlation coefficients and the correlation matrix as an introduction to multivariate analysis. Combined with the basic statistics obtained in the previous lesson we can get a good overview of a high-dimensional data set before applying machine learning algorithms.

Work Through: Properties of a Data Set

Univariate properties

For recordings of variables that are contained e.g. in the columns of a dataframe, we often assume the independence of samples: the measurement in one row does not depend on the recording in another row. Therefore all results of the features obtained e.g. under `describe` will not depend on the order of the rows. Also, while the numbers obtained from different rows can be similar (or even the same) by chance, there is no way to *predict* the values in one row from the values of another row.

When comparing different variables arranged in columns, in contrast, this is not necessarily so. (We assume here that they are consistent, e.g. all values in a single row obtained from the same subject.) The values in one column can be related to the numbers in another column and specifically they can show degrees of similarity. If, for instance, we have a number of subjects investigated some of who have an inflammatory disease and some of who are healthy controls, an inflammatory marker might be increased in the diseased subjects. If several markers are recorded from each subject (i.e. more than one column in the data frame), the values of several inflammatory markers may be elevated simultaneously in the diseased subjects. Thus, the profiles of these markers across the whole group will show a certain similarity.

The goal of multivariate data analysis is to find out whether similarities (or, in general, any relationships) between recorded variables exist.

Let us first import a demo data set and check its basic statistics.

For a work through example, let us work with the 'patients' data set. We import the data from the .csv file using `read_csv` from Pandas into a dataframe. We then check the number of columns and rows using the `len` function. We also check the data type of each column to find out which columns can be used for quantitative analysis.

```
# Please adjust path according to operating system & personal path to
    file
df = read_csv('data/patients.csv')

df.head()
print('Number of columns: ', len(df.columns))
print('Number of rows: ', len(df))
df.head()
```

```
    Age  Height  Weight  Systolic  Diastolic  Smoker  Gender
0    38      71   176.0     124.0       93.0       1    Male
1    43      69   163.0     109.0       77.0       0    Male
2    38      64   131.0     125.0       83.0       0  Female
3    40      67   133.0     117.0       75.0       0  Female
4    49      64   119.0     122.0       80.0       0  Female
```

```
Number of columns:  7
Number of rows:  100
    Age  Height  Weight  Systolic  Diastolic  Smoker  Gender
0    38      71   176.0     124.0       93.0       1    Male
1    43      69   163.0     109.0       77.0       0    Male
2    38      64   131.0     125.0       83.0       0  Female
3    40      67   133.0     117.0       75.0       0  Female
4    49      64   119.0     122.0       80.0       0  Female
```

```
print('The columns are of the following data types:')
df.dtypes
```

```
The columns are of the following data types:
Age            int64
Height         int64
Weight       float64
Systolic     float64
Diastolic    float64
Smoker         int64
Gender        object
dtype: object
```

Out of the seven columns, three containing integers, three containing floating point (decimal) numbers, and the last one containing gender specification as ‘female’ or ‘male’. We note that the sixth column contains a binary classification. Numerical analysis can thus be restricted to columns with indices 0 to 4.

DIY1: Univariate properties of the patients data

1. Get the basic statistical properties of first five columns from the ‘describe’ function.
2. Create a barchart of the means of each column. To access a row by its name you can use the convention ‘df_describe.loc[’name’]’.
3. **Optional:** In the bar chart of the means, try to add the standard deviation as an errorbar, using the keyword argument `yerr` in the form ‘yerr = df_describe.loc[’std’]’.

```
df = read_csv('data/patients.csv')
df_describe = df.iloc[:, :5].describe()
df_describe.round(2)
```

```
            Age  Height  Weight  Systolic  Diastolic
count  100.00  100.00  100.00    100.00     100.00
mean    38.28   67.07  154.00    122.78      82.96
std      7.22    2.84   26.57      6.71       6.93
min     25.00   60.00  111.00    109.00      68.00
25%     32.00   65.00  130.75    117.75      77.75
```

```
50%     39.00   67.00  142.50    122.00      81.50
75%     44.00   69.25  180.25    127.25      89.00
max     50.00   72.00  202.00    138.00      99.00
```

```
fig, ax = subplots()
bins = arange(5)
ax.bar(bins, df_describe.loc['min'])
show()
```

```

fig, ax = subplots()
bins = arange(5)
ax.bar(bins, df_describe.loc['min'], yerr=df_describe.loc['std'])
ax.set_xticks(bins)
ax.set_xticklabels(df.columns[:5], fontsize=12);
show()
```

Visual Search for Similarity: the Scatter Plot

In Matplotlib, the function `scatter` allows plotting of one variable against the other. This is a common way to visually check for relationships between individual columns in a dataframe.

```
# Scatter plot
fig, ax = subplots();

ax.scatter(df['Weight'], df['Height']);
ax.set_xlabel('Weight (pounds)', fontsize=16)
ax.set_ylabel('Height (inches)', fontsize=16)

show()
```

The data points appear to be grouped into two clouds. We will not deal with this qualitative aspect further at present. Grouping will be discussed as Unsupervised Machine Learning or Clustering later in the Course.

However, from the plot one might also suspect that there is a trend of heavier people being taller. For instance, we note that there are no points in the lower right corner of the plot (weight >160 pounds and height < 65 inches).

DIY2: Scatter plot from the patients data

Create a scatter plot of the systolic via the diastolic blood pressure. Do the two variables appear to be independent or related?

Scatter plots are useful for the inspection of select pairs of data. However, they are only qualitative and thus, in general, it is preferred to have a numerical quantity.

```
fig, ax = subplots();
ax.scatter(df['Systolic'], df['Diastolic']);
ax.set_xlabel('Systolic', fontsize=16)
ax.set_ylabel('Diastolic', fontsize=16)

show()
```

From the plot one might suspect that a larger systolic value is connected with a larger diastolic value. However, the plot in itself is not conclusive in that respect.

The Correlation Coefficient

Bivariate measures are quantities that are calculated from two variables of data. Bivariate features are the most widely used subset of multivariate features - all of which require more than one variable in order to be calculated.

The concept behind many bivariate measures is to quantify “similarity” between two data sets. If any similarity is discovered it is assumed that there is some connection or relationship between the sets. For similar variables, knowledge of one leads to some expectation about the other.

Here we are going to look at a specific bivariate quantity: the Pearson correlation coefficient PCC.

The formula for the Pearson PCC is set up such that two identical data sets yield a PCC of 1. (Technically this is done by normalising all variances to be equal to 1). This implies that all data points in a scatter plot of a variable against itself are aligned along the main diagonal (with positive slope).

Two perfectly antisymmetrical data sets (one variable can be obtained by multiplying the other by -1) yield a value -1. This implies that all data points in a scatter plot are aligned along the negative or anti diagonal (with negative slope). All other situations lie in between. A value of 0 refers to exactly balanced positive and negative contributions to the measure. (Note that strictly speaking, the latter does not necessarily mean that there is no relationship between the variables).

The PCC is an **undirected** measure in the sense that its value for the comparison between data set 1 and data set 2 is the same as the PCC between data set 2 and data set 1.

A direct way to calculate the PCC of two data-sets is to use the function `corr` applied to a dataframe. For instance, we can apply it to the Everleys data:

```
df_everley = read_csv('data/everleys_data.csv')
df_everley.corr()
```

```
          calcium    sodium
```

```
calcium  1.000000 -0.258001
sodium  -0.258001  1.000000
```

The result as a matrix of two-by-two numbers. Along the diagonal (top left and bottom right) are the values for the comparison of a column to itself. As any dataset is identical with itself, the values are one by definition.

The non-diagonal elements show that the $CC \approx -0.26$ for the two data sets. Both $CC(12)$ and $CC(21)$ are given in the matrix but because of the symmetry we would only need to report one of the two.

Note

Note that in this lesson we introduce how to calculate the PCC but do not discuss its significance. E.g. the interpretation of the value above needs to be checked against the fact that we only have 18 data points. Specifically, we refrain from concluding that because the PCC is negative, a high value of the calcium concentration is associated with a small value of the sodium concentration (relative to their respective means).

One quantitative way to assess whether or not a given value of the PCC is meaningful or not is to use surrogate data. In our case, we could e.g. create random numbers in an array with shape (18, 2) such that the two means and standard deviations are the same as in the Everley data but the two columns are independent of each other. Creating many realisations, we can check what distribution of PCC values is expected from the randomly generated data and compare this with the values from the Everleys data.

A lot of what we are going to do in the machine learning sessions will involve Numpy arrays. Let us therefore convert the Everleys data from a Pandas dataframe into a Numpy array.

```
everley_numpy = df_everley.to_numpy()
everley_numpy
```

```
array([[  3.4555817 , 112.69098   ],
       [  3.6690263 , 125.66333   ],
       [  2.7899104 , 105.82181   ],
       [  2.9399    ,  98.172772  ],
       [  5.42606   ,  97.931489  ],
       [  0.71581063, 120.85833   ],
       [  5.6523902 , 112.8715    ],
       [  3.5713201 , 112.64736   ],
       [  4.3000669 , 132.03172   ],
       [  1.3694191 , 118.49901   ],
       [  2.550962  , 117.37373   ],
       [  2.8941294 , 134.05239   ],
```

```
        [  3.6649873 , 105.34641   ],
        [  1.3627792 , 123.35949   ],
        [  3.7187978 , 125.02106   ],
        [  1.8658681 , 112.07542   ],
        [  3.2728091 , 117.58804   ],
        [  3.9175915 , 101.00987   ]])
```

We can see that the numbers remain the same but the format changed. E.g. we have lost the names of the columns. Similar to the Pandas dataframe, we can use 'shape' to see the dimensions of the data array.

```
everley_numpy.shape
```

```
(18, 2)
```

We can now use the Numpy function `corrcoef` to calculate the Pearson correlation:

```
from numpy import corrcoef

corr_matrix = corrcoef(everley_numpy, rowvar=False)

print(corr_matrix)
```

```
[[ 1.         -0.25800058]
 [-0.25800058  1.        ]]
```

The function `corrcoef` takes a two-dimensional array as input. The keyword argument `rowvar` is True by default which means the correlation will be calculated along the rows. As we have the data features in the columns, it needs to be set to False. (You can check what happens if you set it to 'True'. Instead of a 2x2 matrix for two columns you will get a 18x18 matrix for eighteen pair comparisons.)

We mentioned that the values of the PCC are calculated such that they must lie between -1 and 1. This is achieved by normalisation with the variance. If for some reason we don't want the similarity calculated using this normalisation, what we get is the so-called **covariance**. In contrast to the PCC its values will depend on the absolute size of the numbers in the data array. From Numpy, we can use the function `cov` to calculate the covariance:

```
from numpy import cov

cov_matrix = cov(everley_numpy, rowvar=False)

print(cov_matrix)
```

```
[[  1.70733842  -3.62631625]
 [ -3.62631625 115.70986192]]
```

The result shows how the covariance is strongly dependent on the actual numerical values in a data

column. The two values along the diagonal are identical with the variances obtained by squaring the standard deviation (calculated for example using the `describe` function).

DIY3: Correlations from the patients data

Calculate the Pearson PCC between the systolic and the diastolic blood pressure from the patients data using

i) the Pandas dataframe and

ii) the data as Numpy array.

```
df = read_csv('data/patients.csv')

df[['Systolic', 'Diastolic']].corr()
```

```
           Systolic  Diastolic
Systolic   1.000000   0.511843
Diastolic  0.511843   1.000000
```

```
df_SysDia_numpy = df[['Systolic', 'Diastolic']].to_numpy()

df_SysDia_corr = corrcoef(df_SysDia_numpy, rowvar=False)

print('Correlation coefficient between Systole and Diastole:', round(
    df_SysDia_corr[0, 1], 2))
```

```
Correlation coefficient between Systole and Diastole: 0.51
```

It is worth noting that it is equally possible to calculate the correlation between rows of a two-dimension array (i.e. rowvar=True) but the interpretation will differ. Imagine a dataset where for two subjects a large number, call it N, of metabolites were determined quantitatively (a Metabolomics dataset). If that dataset is of shape (2, N) then one can calculate the correlation between the two rows. This would be done to determine the correlation of the metabolite profiles between the two subjects.

The Correlation Matrix

If we have more than two columns of data, we can obtain a Pearson correlation coefficient for each pair. In general, for N columns, we get N^2 pairwise values. We omit the correlations of each column with itself, of which there are N, which means we are left with $N * (N-1)$ pairs. Because each value appears twice due to symmetry of the calculation, we can ignore half of them and we are left with $N * (N-1)/2$ coefficients for N columns.

Here is an example for the 'patients' data:

```
df = read_csv('data/patients.csv')

df.corr()
```

```
Error: ValueError: could not convert string to float: 'Male'
```

If we do the calculation with the Pandas dataframe, the 'Gender' is automatically ignored and by default we get $6 * 5/2 = 15$ coefficients for the six remaining columns. Note that the values that involves the 'Smoker' column are meaningless.

Let us now convert the dataframe to a Numpy array and check its shape:

```
patients_numpy = df.to_numpy()
patients_numpy.shape
```

```
(100, 7)
```

Now we can try to calculate the correlation matrix for the first five columns of this data array. If we do it directly to the array, we get an AttributeError: 'float' object has no attribute 'shape'.

This is mended by converting the array to floating point before using the `corrcoef` function. For this we use `astype(float)`:

```
cols = 5

patients_numpy_float = patients_numpy[:, :cols].astype(float)

patients_corr = corrcoef(patients_numpy_float, rowvar=False)

patients_corr
```

```
array([[1.        , 0.11600246, 0.09135615, 0.13412699, 0.08059714],
       [0.11600246, 1.        , 0.6959697 , 0.21407555, 0.15681869],
       [0.09135615, 0.6959697 , 1.        , 0.15578811, 0.22268743],
       [0.13412699, 0.21407555, 0.15578811, 1.        , 0.51184337],
       [0.08059714, 0.15681869, 0.22268743, 0.51184337, 1.        ]])
```

The result is called the **correlation matrix**. It contains all the bivariate comparisons possible for the five columns chosen.

In the calculation above we used the PCC to calculate the matrix. In general, any bivariate measure can be used to obtain a matrix of same shape.

Heat map in Matplotlib

To get an illustration of the correlation pattern in a dataset we can plot the correlation matrix as a heatmap.

Here is some code using Matplotlib to plot a heatmap of the correlation matrix from the patients dataset. We use the function `imshow`:

```
fig, ax = subplots(figsize=(5,5))

im = ax.imshow(patients_corr, cmap='coolwarm');

show()
```

Note that we have specified the colour map 'coolwarm'. For a list of Matplotlib colour maps, please refer to the gallery in the documentation. The names to use in the code are on the left hand side of the colour bar.

Let us add two more features to improve the figure.

First, to have true correlations stand out (rather than the trivial self correlations along the diagonal which are always one) we we can deliberately set the diagonal equal to zero. To achieve this, we use the Numpy function `fill_diagonal`.

Second, `imshow` scales the colours by default to the minimum and maximum value of the array. As such we don't know what red or blue means. To see the colour bar, it can be added to the figure environment 'fig' using `colorbar`.

```
from numpy import fill_diagonal

fill_diagonal(patients_corr, 0)

fig, ax = subplots(figsize=(7,7))

im = ax.imshow(patients_corr, cmap='coolwarm');

fig.colorbar(im, orientation='horizontal', shrink=0.7);

show()
```

The result is that the correlation between columns 'Height' and 'Weight' is the strongest and presumably higher than could be expected if these two measures were independent. We can confirm this by plotting a scatter plot for these two columns and compare to the scatter plot for (original) columns 2 (Height) and 5 (Diastolic blood pressure):

DIY4: Spearman Correlations from the patients data

Calculate and plot the correlation matrix of the first five columns as above based on the Spearman rank correlation coefficient. It is based on the ranking of values instead of their numerical values as for the Pearson coefficient. Spearman therefore tests for tests for monotonic relationships whereas Pearson tests for linear relationships.

To import the function use:

```
from scipy.stats import spearmanr
```

You can then apply it in the form:

```
data_spearman_corr = spearmanr(data).correlation
```

```
from scipy.stats import spearmanr
patients_numpy = df.to_numpy()
cols = 5

patients_numpy_float = patients_numpy[:, :cols].astype(float)
patients_spearman = spearmanr(patients_numpy_float).correlation

patients_spearman
```

```
array([[1.        , 0.11636668, 0.09327152, 0.12105741, 0.08703685],
       [0.11636668, 1.        , 0.65614849, 0.20036338, 0.14976559],
       [0.09327152, 0.65614849, 1.        , 0.12185782, 0.19738765],
       [0.12105741, 0.20036338, 0.12185782, 1.        , 0.48666928],
       [0.08703685, 0.14976559, 0.19738765, 0.48666928, 1.        ]])
```

```
from numpy import fill_diagonal
fill_diagonal(patients_spearman, 0)

fig, ax = subplots(figsize=(7,7))

im = ax.imshow(patients_spearman, cmap='coolwarm');
fig.colorbar(im, orientation='horizontal', shrink=0.7);

show()
```

Analysis of the Correlation matrix

The Correlation Coefficients

To analyse the correlations in a data set, we are only interested in the $N * (N-1)/2$ unduplicated correlation coefficients. Here is a way to extract them and assign them to a variable.

We import the function `triu_indices`. It provides the indices of a matrix with specified size. The size we need is obtained from our correlation matrix, using `len`. It is identical to the number of columns for which we calculated the CCs.

We also need to specify that we do not want the diagonal to be included. For this, there is an offset parameter 'k' which will collect the indices excluding the diagonal if it is set to 1. (To include the indices of the diagonal it would have to be 0).

```
from numpy import triu_indices

# Get the number of rows of the correlation matrix
no_cols = len(patients_corr)

# Get the indices of the 10 correlation coefficients for 5 data columns
corr_coeff_indices = triu_indices(no_cols, k=1)

# Get the 10 correlation coefficients
corr_coeffs = patients_corr[corr_coeff_indices]

print(corr_coeffs)
```

```
[0.11600246 0.09135615 0.13412699 0.08059714 0.6959697  0.21407555
 0.15681869 0.15578811 0.22268743 0.51184337]
```

Now we plot these correlation coefficients as a bar chart to see them one next to each other.

```
fig, ax = subplots()

bins = arange(len(corr_coeffs))

ax.bar(bins, corr_coeffs);

show()
```

If there is a large number of coefficients, we can also display their histogram or a boxplot as a summary statistics.

The Average Correlation per Column

On a higher level, we can calculate the overall or average correlation per data column. We achieve this by averaging over either the rows or the columns of the correlation matrix. Because our similarity measure is undirected, both ways of summing yield the same result.

However, we need to consider the sign. The correlation coefficients can be positive or negative. As such, adding for instance +1 ans -1 would yield an average of zero even though both indicate perfect

correlation and anti-correlation, respectively. We address this by using the absolute value `abs`, ignoring the sign.

To average, we use function `mean`. This function by default averages over all values of the matrix. To obtain the five values by averaging over the columns, we specify the 'axis' keyword argument as 0.

```
from numpy import abs, mean

# Absolute values of correlation matrix
corr_matrix_abs = abs(patients_corr)

# Average of the correlation strengths
corr_column_average = mean(corr_matrix_abs, axis=0)

fig, ax = subplots()

bins = arange(corr_column_average.shape[0])

ax.bar(bins, corr_column_average );

print(corr_column_average)

show()
```

```
[0.08441655 0.23657328 0.23316028 0.20316681 0.19438933]
```

The result is that the average column correlation is on the order of 0.2 for the columns with indices 1 to 4 and less than 0.1 for the column with index 0, which is the age.

The Average Data Set Correlation

The sum over rows or columns has given us a reduced set of values to look at. We can now take the final step and average over all correlation coefficients. This will yield the average correlation of the data set. It condenses the full bivariate analysis into a single number and can be a starting point when comparing e.g. different data sets of the same type.

```

# Average of the correlation strengths
corr_matrix_average = mean(corr_matrix_abs)

print('Average correlation strength: ', round(corr_matrix_average, 3))
```

```
Average correlation strength:  0.19
```

Application: The Diabetes Data Set

We now return to the data set that started our enquiry into dataframes in the previous lesson. Let us apply the above and do a summary analysis of its bivariate features.

First we import the data. it is one of the example datasets of scikit-learn, the Python library for Machine Learning. As such it is already included in the Anaconda package and you can import it directly.

```
from sklearn import datasets

diabetes = datasets.load_diabetes()

data_diabetes = diabetes.data
```

For the bivariate features, let us get the correlation matrix and plot it as a heatmap. We use code introduced above.

```
from numpy import fill_diagonal

data_corr_matrix = corrcoef(data_diabetes, rowvar=False)

fill_diagonal(data_corr_matrix, 0)

fig, ax = subplots(figsize=(8, 8))

im = ax.imshow(data_corr_matrix, cmap='coolwarm');

fig.colorbar(im, orientation='horizontal', shrink=0.5);

show()
```

There is one strongly correlated pair (column indices 4 and 5) and one strongly anti-correlated pair (column indices 6 and 7).

Now we calculate the $10 * 9/2 = 45$ correlation coefficients and plot them as a histogram:

```
from numpy import triu_indices

data_corr_coeffs = data_corr_matrix[triu_indices(data_corr_matrix.shape
    [0], k=1)]

fig, ax = subplots()

ax.hist(data_corr_coeffs, bins=10);

show()
```

This histogram shows that the data have a distribution that is shifted towards positive correlations. However, only four values are (absolutely) larger than 0.5 (three positive, one negative).

Next we can get the average (absolute) correlation per column.

```
data_column_average = mean(abs(data_corr_matrix), axis=0)

fig, ax = subplots()

bins = arange(len(data_column_average))

ax.bar(bins, data_column_average);
ax.set_title('Average Correlation Strength per Column')
ax.set_xticks(arange(len(diabetes.feature_names)))
ax.set_xticklabels(diabetes.feature_names);

show()
```

In the plot, note how the column names were extracted from the 'diabetes' data using `diabetes.feature_names`.

Finally, we can obtain the average correlation of the whole data set.

```
# Average of the correlation strengths
data_corr_matrix_average = mean(abs(data_corr_matrix))

print('Average Correlation Strength: ', round(data_corr_matrix_average,
    3))
```

```
Average Correlation Strength:  0.29
```

Exercises

End of chapter Exercises

Assignment: The Breast Cancer Data

Import the breast cancer data set using `read_csv`. Based on the code of this lesson, try to do the following:

1. Get the summary (univariate) statistics of columns 2-10 (accessing indices 1:10) using `describe`
2. Plot the means of each column as a bar chart with standard deviations as error bars. Why are some bars invisible?
3. Extract the values as Numpy array using `to_numpy`. The shape of the array should be (569, 31).
4. Calculate the correlation matrix using `corrcoef` from Numpy and plot it as a heatmap. The shape of the matrix should be (31, 31). Use `fill_diagonal` to set the diagonal elements to 0.
5. Calculate the average column correlation and plot it as a bar chart.

6. Calculate the average correlation strength of the data set.

In case of doubt, try to get help from the respective documentations for Pandas dataframes, Numpy and Matplotlib.

Q1

```
# To import data from a csv file into a Pandas dataframe
from pandas import read_csv

# To import a dataset from scikit-learn
from sklearn import datasets

# To create figure environments and plots
from matplotlib.pyplot import subplots, show

# Specific numpy functions, description in the main body
from numpy import corrcoef, fill_diagonal, triu_indices, arange
from numpy import mean

df_bc = read_csv("data/breast_cancer.csv")

df_bc_describe = df_bc.iloc[:, 1:10].describe()

df_bc_describe.round(2)
```

```
       radius_mean  texture_mean  ...  concave points_mean
          symmetry_mean
count       569.00        569.00  ...               569.00
    569.00
mean         14.13         19.29  ...                 0.05
    0.18
std           3.52          4.30  ...                 0.04
    0.03
min           6.98          9.71  ...                 0.00
    0.11
25%          11.70         16.17  ...                 0.02
    0.16
50%          13.37         18.84  ...                 0.03
    0.18
75%          15.78         21.80  ...                 0.07
    0.20
max          28.11         39.28  ...                 0.20
    0.30

[8 rows x 9 columns]
```

Q2

```
fig, ax = subplots()

bins = arange(df_bc_describe.shape[1])

ax.bar(bins, df_bc_describe.loc['mean'], yerr=df_bc_describe.loc['std'
    ])

show()
```

```
<BarContainer object of 9 artists>
```

Q3

```
bc_numpy = df_bc.to_numpy()

bc_numpy.shape
```

```
(569, 31)
```

Q4

```
bc_corr = corrcoef(bc_numpy, rowvar=False)

bc_corr.shape
```

```
(31, 31)
```

Q5

```
from numpy import fill_diagonal

fill_diagonal(bc_corr, 0)

fig, ax = subplots(figsize=(7,7))

im = ax.imshow(bc_corr, cmap='coolwarm');

fig.colorbar(im, orientation='horizontal', shrink=0.7);

show()
```

Q6

```
bc_column_average = mean(abs(bc_corr), axis=0)

fig, ax = subplots()

bins = arange(len(bc_column_average))

ax.bar(bins, bc_column_average);
ax.set_title('Average Correlation Strength per Column');

show()
```

```
# Average of the correlation strengths
bc_corr_matrix_average = mean(abs(bc_corr))

print('Average Correlation Strength: ', round(bc_corr_matrix_average,
    3))
```

```
Average Correlation Strength:  0.387
```

Keypoints

- Quantities based on data from two variables are referred as bivariate measures.
- Bivariate properties can be studied using `matplotlib` and `numpy`.
- Multivariate data analysis helps to find out relationships between recorded variables.
- Functions `corr` and `corrcoef` are used to calculate the PCC.
- A correlation matrix is visualised as a heatmap.