Statistics in Python

Why Python for statistics?
R is a language dedicated to statistics. Python is a general-purpose language with statistics modules. R has more statistical analysis features than Python, and specialized syntaxes. However, when it comes to building complex analysis pipelines that mix statistics with e.g. image analysis, text mining, or control of a physical experiment, the richness of Python is an invaluable asset.

Data as a table

The setting that we consider for statistical analysis is that of multiple *observations* or *samples* described by a set of different *attributes* or *features*. The data can than be seen as a 2D table, or matrix, with columns giving the different attributes of the data, and rows the observations. For instance, the data contained in `'brain_size.csv'`:

The pandas data-frame

We will store and manipulate this data in a **pandas.DataFrame**, from the pandas module. It is the Python equivalent of the spreadsheet table. It is different from a 2D numpy array as it has named columns, can contain a mixture of different data types by column, and has elaborate selection and pivotal mechanisms.

A. *Creating dataframes: reading data files or converting arrays*

Separator

It is a CSV file, but the separator is “;”

Reading from a CSV file: Using the above CSV file that gives observations of brain size and weight and IQ (Willerman et al. 1991), the data are a mixture of numerical and categorical values:

```
>>>
```

```
>>> import pandas
>>> data = pandas.read_csv('brain_size.csv', sep=';', na_values=".")
>>> data
   Unnamed: 0  Gender  FSIQ  VIQ  PIQ  Weight  Height  MRI_Count
0           1  Female   133  132  124   118.0    64.5     816932
1           2    Male   140  150  124     NaN    72.5    1001121
2           3    Male   139  123  150   143.0    73.3    1038437
3           4    Male   133  129  128   172.0    68.8     965353
4           5  Female   137  132  134   147.0    65.0     951545
...
```

Missing values

The weight of the second individual is missing in the CSV file. If we don’t specify the missing value (NA = not available) marker, we will not be able to do statistical analysis.

Creating from arrays: A **pandas.DataFrame** can also be seen as a dictionary of 1D ‘series’, eg arrays or lists. If we have 3 numpy arrays:

```
>>>
```

```
>>> import numpy as np
>>> t = np.linspace(-6, 6, 20)
>>> sin_t = np.sin(t)
>>> cos_t = np.cos(t)
```

We can expose them as a **pandas.DataFrame**:

```
>>> pandas.DataFrame({'t': t, 'sin': sin_t, 'cos': cos_t})
           t       sin       cos
0  -6.000000  0.279415  0.960170
1  -5.368421  0.792419  0.609977
2  -4.736842  0.999701  0.024451
3  -4.105263  0.821291 -0.570509
4  -3.473684  0.326021 -0.945363
5  -2.842105 -0.295030 -0.955488
6  -2.210526 -0.802257 -0.596979
7  -1.578947 -0.999967 -0.008151
8  -0.947368 -0.811882  0.583822
...
```

Other inputs: pandas can input data from SQL, excel files, or other formats. See the pandas documentation.

B. *Manipulating data*

data is a **pandas.DataFrame**, that resembles R's dataframe:

```
>>> data.shape    # 40 rows and 8 columns
(40, 8)

>>> data.columns  # It has columns
Index([u'Unnamed: 0', u'Gender', u'FSIQ', u'VIQ', u'PIQ', u'Weight', u'Height', u'MRI_Count'],
 dtype='object')

>>> print(data['Gender'])  # Columns can be addressed by name
0     Female
1       Male
2       Male
3       Male
4     Female
...

>>> # Simpler selector
>>> data[data['Gender'] == 'Female']['VIQ'].mean()
109.45
```

Note
For a quick view on a large dataframe, use its *describe* method: **pandas.DataFrame.describe()**.

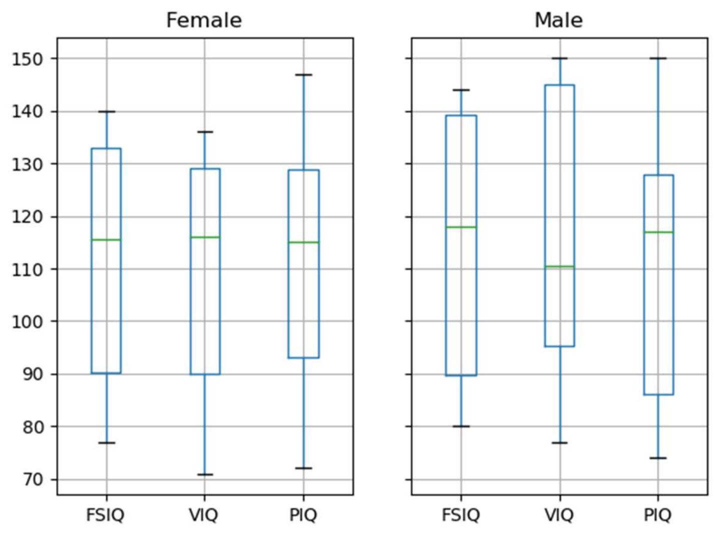
groupby: splitting a dataframe on values of categorical variables:

```
>>> groupby_gender = data.groupby('Gender')
>>> for gender, value in groupby_gender['VIQ']:
...     print((gender, value.mean()))
('Female', 109.45)
('Male', 115.25)
```

groupby_gender is a powerful object that exposes many operations on the resulting group of dataframes:

```
>>> groupby_gender.mean()
        Unnamed: 0   FSIQ     VIQ     PIQ      Weight     Height  MRI_Count
Gender
Female       19.65  111.9  109.45  110.45  137.200000  65.765000   862654.6
Male         21.35  115.0  115.25  111.60  166.444444  71.431579   954855.4
```

Use tab-completion on *groupby_gender* to find more. Other common grouping functions are median, count (useful for checking to see the amount of missing values in different subsets) or sum. Groupby evaluation is lazy, no work is done until an aggregation function is applied.



Exercise 3.1

- *What is the mean value for VIQ for the full population?*
- *What is the average value of MRI counts, for males and females?*

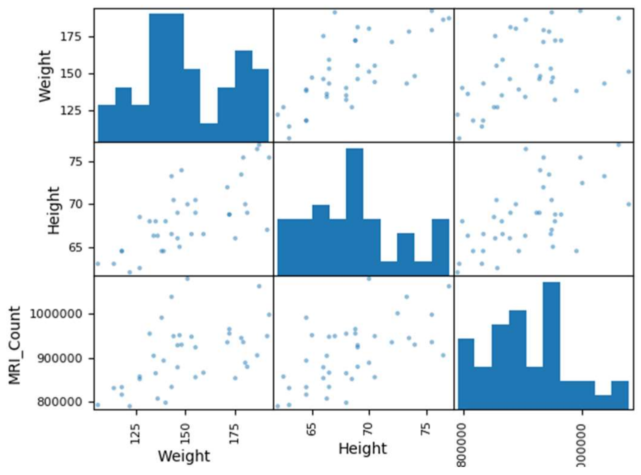
C. Plotting data

Pandas comes with some plotting tools (**`pandas.tools.plotting`**, using matplotlib behind the scene) to display statistics of the data in dataframes:

Scatter matrices:

```
>>>

>>> from pandas.plotting import scatter_matrix
```

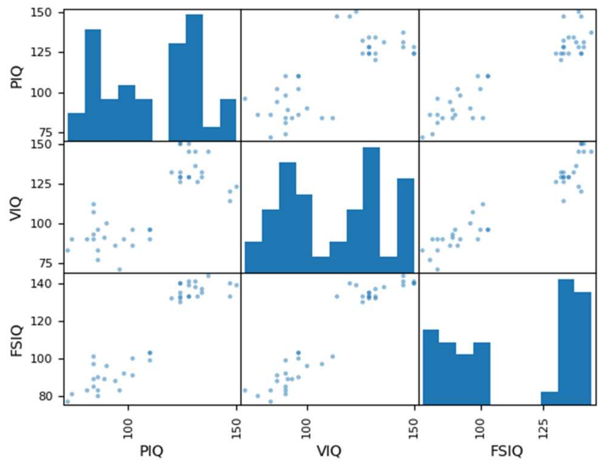
```
>>> scatter_matrix(data['Weight', 'Height', 'MRI_Count'], alpha=0.2, figsize=(6, 6), diagonal=
'hist')
```



```
>>> scatter_matrix(data['PIQ', 'VIQ', 'FSIQ'], alpha=0.2, figsize=(6, 6), diagonal='hist')
```

Two populations

The IQ metrics are bimodal, as if there are 2 sub-populations.

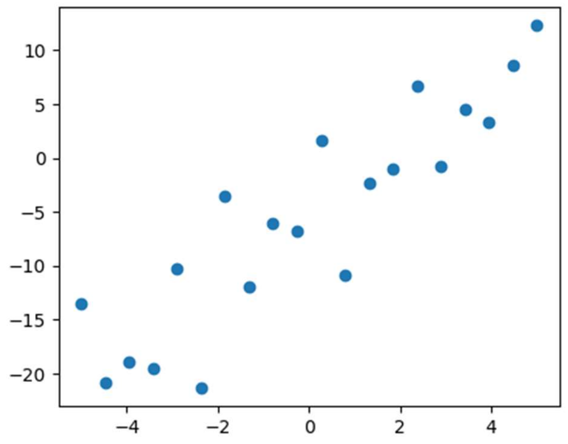


Exercise 3-2

Plot the scatter matrix for males only, and for females only.

D. Linear models, multiple factors, and analysis of variance

D-1: A simple linear regression



Given two set of observations, x and y, we want to test the hypothesis that y is a linear function of x. In other terms:

$$y = x * coef + intercept + e$$

where e is observation noise. We will use the statsmodels module to:

1. Fit a linear model. We will use the simplest strategy, ordinary least squares (OLS).
2. Test that *coef* is non zero.

First, we generate simulated data according to the model:

```
>>>
```

```
>>> import numpy as np
>>> x = np.linspace(-5, 5, 20)
>>> np.random.seed(1)
>>> # normal distributed noise
>>> y = -5 + 3*x + 4 * np.random.normal(size=x.shape)
>>> # Plot the data
>>> plt.figure(figsize=(5, 4))
>>> plt.plot(x, y, 'o')
>>> # Create a data frame containing all the relevant variables
>>> data = pandas.DataFrame({'x': x, 'y': y})
```

Then we specify an OLS model and fit it:

```
>>>
```

```
>>> from statsmodels.formula.api import ols
>>> model = ols("y ~ x", data).fit()
```

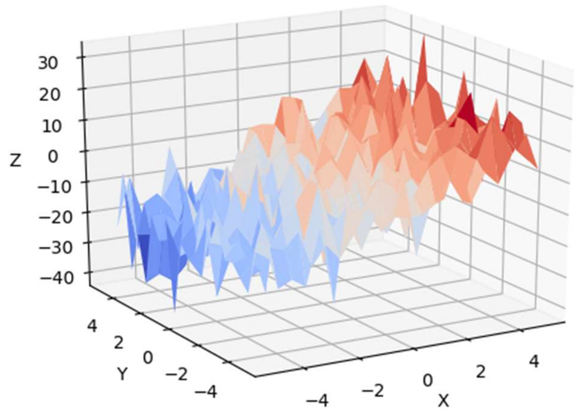
We can inspect the various statistics derived from the fit:

```
>>>
```

```
>>> print(model.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.804
Model:                            OLS   Adj. R-squared:                  0.794
Method:                 Least Squares   F-statistic:                     74.03
Date:                    ...            Prob (F-statistic):           8.56e-08
Time:                            ...    Log-Likelihood:                -57.988
No. Observations:                  20   AIC:                             120.0
Df Residuals:                      18   BIC:                             122.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -5.5335      1.036     -5.342      0.000      -7.710      -3.357
x              2.9369      0.341      8.604      0.000       2.220       3.654
==============================================================================
Omnibus:                        0.100   Durbin-Watson:                   2.956
Prob(Omnibus):                  0.951   Jarque-Bera (JB):                0.322
Skew:                          -0.058   Prob(JB):                        0.851
Kurtosis:                       2.390   Cond. No.                         3.03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
ecified.
```

Exercise 3-3
Similar to model above, use Analysis of Variance (ANOVA) on linear models, plot the fitted model and retrieve the parameter estimates

D-2: Multiple Regression: including multiple factors



Consider a linear model explaining a variable *z* (the dependent variable) with 2 variables *x* and *y*:

$$z = x\,c_1 + y\,c_2 + i + e$$

Such a model can be seen in 3D as fitting a plane to a cloud of (x, y, z) points.

Example 1: Simulate data

```
"""
Multiple Regression
====================
Calculate using 'statsmodels' just the best fit, or all the corresponding
statistical parameters.
Also shows how to make 3d plots. Original author: Thomas Haslwanter
"""

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

# For statistics. Requires statsmodels 5.0 or more
from statsmodels.formula.api import ols

##############################################################################
# Generate and show the data
x = np.linspace(-5, 5, 21)
# We generate a 2D grid
X, Y = np.meshgrid(x, x)

# To get reproducable values, provide a seed value
np.random.seed(1)

# Z is the elevation of this 2D grid
Z = -5 + 3*X - 0.5*Y + 8 * np.random.normal(size=X.shape)

# Plot the data
# For 3d plots. This import is necessary to have 3D plotting below
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap=plt.cm.coolwarm,
                       rstride=1, cstride=1)
ax.view_init(20, -120)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

##############################################################################
# Multilinear regression model, calculating fit, P-values, confidence
# intervals etc.
# Convert the data into a Pandas DataFrame to use the formulas framework
# in statsmodels

# First we need to flatten the data: it's 2D layout is not relevant.
X = X.flatten()
Y = Y.flatten()
Z = Z.flatten()
data = pandas.DataFrame({'x': X, 'y': Y, 'z': Z})

# Fit the model
model = ols("z ~ x + y", data).fit()
plt.show()

# Print the summary
print(model.summary())
print("\nRetrieving manually the parameter estimates:")
print(model._results.params)

# Analysis of Variance (ANOVA) on linear models
from statsmodels.stats.anova import anova_lm

# Peform analysis of variance on fitted linear model
anova_results = anova_lm(model)

print('\nANOVA results')
print(anova_results)
```

Example 2: the iris data (`iris.csv`)

Sepal and petal size tend to be related: bigger flowers are bigger! But is there in addition a systematic effect of species?

Analysis of Iris petal and sepal sizes

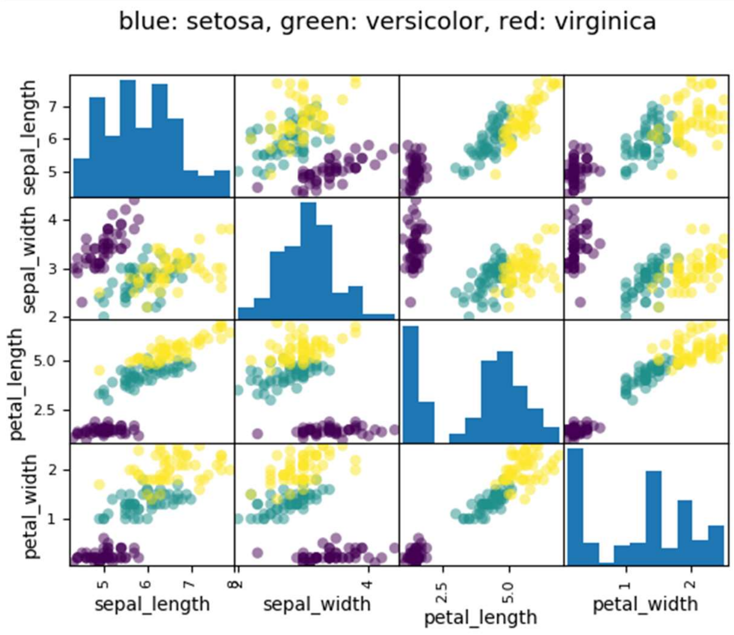
Ilustrate an analysis on a real dataset:

- Visualizing the data to formulate intuitions
- Fitting of a linear model
- Hypothesis test of the effect of a categorical variable in the presence of a continuous confound

```
import matplotlib.pyplot as plt

import pandas
from pandas.plotting import scatter_matrix
from statsmodels.formula.api import ols

# Load the data
data = pandas.read_csv('iris.csv')
```

- Plot a scatter matrix

```
# Express the names as categories
categories = pandas.Categorical(data['name'])

# The parameter 'c' is passed to plt.scatter and will control the color
scatter_matrix(data, c=categories.codes, marker='o')

fig = plt.gcf()
fig.suptitle("blue: setosa, green: versicolor, red: virginica", size=13)
```



Statistical analysis

```
# Let us try to explain the sepal length as a function of the petal
# width and the category of iris

model = ols('sepal_width ~ name + petal_length', data).fit()
print(model.summary())

# Now formulate a "contrast", to test if the offset for versicolor and virginica are identical
print('Testing the difference between effect of versicolor and virginica')
print(model.f_test([0, 1, -1, 0]))
plt.show()
```

***********************************The End***