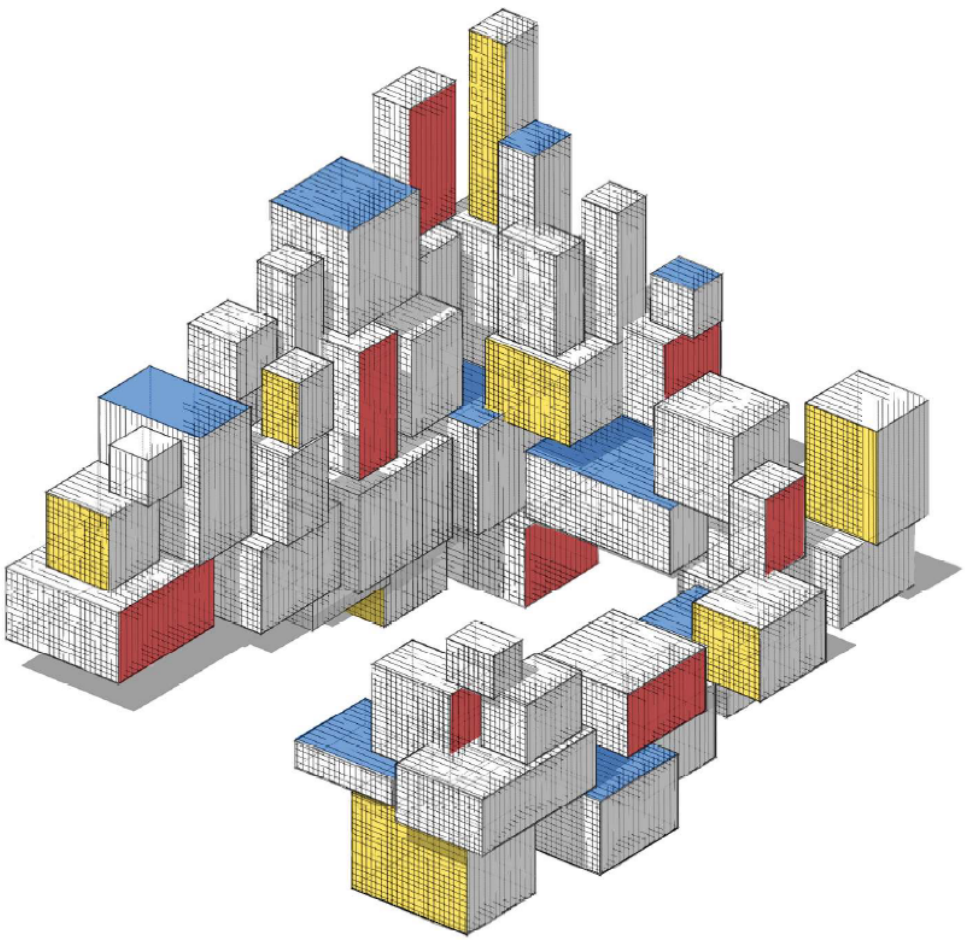
Importing, Summarizing and Visualizing Data



Purpose

In this lecture we discuss:

- Where to find useful data sets
- How to load them into Python.
- How to (re)structure the data
- How to summarize data via tables and figures

Readers unfamiliar with Python are advised to read the appendix of DSML first.

> Python code and data sets for each chapter can be downloaded from the GitHub site: https://github.com/DSML-book

Data

Data can be thought of as being the result of some random experiment — an experiment whose outcome cannot be determined in advance.

Data are often stored in a table or spreadsheet. Convention: variables or features correspond to *columns* and the individual items (or units) correspond to *rows*.

Three types of columns:

1. One column is often an identifier or index column.
2. Certain columns can correspond to the design of the experiment.
3. Other columns represent the observed measurements of the experiment. Usually, these measurements exhibit *variability*.

Data Repositories

A well-known repository of data sets is the Machine Learning Repository maintained by the University of California at Irvine (UCI): https://archive.ics.uci.edu/.

These data sets are typically stored in a CSV (comma separated values) format.

For example, to access the **abalone** data set from this website with Python, download the file to your working directory, import the **pandas** package via

```
import pandas as pd
```

and read in the data as follows:

```
abalone = pd.read_csv('abalone.data', header = None)
```

Data Repositories

Another useful repository of over 1000 data sets from various packages in the R programming language, collected by Vincent Arel-Bundock, can be found at:

https://vincentarelbundock.github.io/Rdatasets/datasets.html.

For example, to read Fisher's famous **iris** data set from R's `datasets` package into Python, type:

```
urlprefix = 'https://vincentarelbundock.github.io/Rdatasets/csv/'
dataname = 'datasets/iris.csv'
iris = pd.read_csv(urlprefix + dataname)
```

Iris Data

The **iris** data set contains four physical measurements (sepal/petal length/width) on 50 specimens (each) of 3 species of iris: setosa, versicolor, and virginica.

The output of **read_csv** is a `DataFrame` object, which is **pandas**'s implementation of a spreadsheet.

```
iris.head()
```

```
   Unnamed: 0  Sepal.Length   ...    Petal.Width  Species
0           1           5.1   ...            0.2   setosa
1           2           4.9   ...            0.2   setosa
2           3           4.7   ...            0.2   setosa
3           4           4.6   ...            0.2   setosa
4           5           5.0   ...            0.2   setosa

[5 rows x 6 columns]
```

The names of the features can be obtained via the `columns` attribute of the `DataFrame` object, as in `iris.columns`.

Abalone Data

The first three rows of the **abalone** data set from the UCI repository can be found as follows:

```
abalone.head(3)
```

	0	1	2	3	4	5	6	7	8
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9

We can manually add the names of the features to the `DataFrame` by reassigning the columns attribute, as in:

```
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
'Whole weight','Shucked weight', 'Viscera weight', 'Shell weight',
'Rings']
```

Structuring Features According to Type

Quantitative features possess "numerical quantity", such as height, age, number of births, etc., and can either be continuous or discrete.

- Continuous quantitative features take values in a continuous range of possible values, such as height, voltage, or crop yield,
- Discrete quantitative features have a countable number of possibilities, such as a count.

Qualitative features do not have a numerical meaning, but their possible values can be divided into a fixed number of categories, such as {M,F} for gender or {blue, black, brown, green} for eye color. For this reason such features are also called categorical or factors.

Nutrition Data

When manipulating, summarizing, and displaying data, it is important to correctly specify the type of the variables (features). We illustrate this using the `nutrition_elderly` data set.

```
xls = 'http://www.biostatisticien.eu/springeR/nutrition_elderly.xls'
nutri = pd.read_excel(xls)
```

This creates a `DataFrame` object **`nutri`**.

```
pd.set_option('display.max_columns', 8) # to fit display
nutri.head(3)
```

```
   gender  situation  tea ...   cooked_fruit_veg  chocol  fat
0       2          1    0 ...                  4       5    6
1       2          1    1 ...                  5       1    4
2       2          1    0 ...                  2       5    4

[3 rows x 13 columns]
```

Check the type of the variables via the **info** method:

```
nutri.info()
```

```
 <class 'pandas.core.frame.DataFrame'>
RangeIndex: 226 entries, 0 to 225
Data columns (total 13 columns):
gender              226 non-null int64
situation           226 non-null int64
tea                 226 non-null int64
coffee              226 non-null int64
height              226 non-null int64
weight              226 non-null int64
age                 226 non-null int64
meat                226 non-null int64
fish                226 non-null int64
raw_fruit           226 non-null int64
cooked_fruit_veg    226 non-null int64
chocol              226 non-null int64
fat                 226 non-null int64
dtypes: int64(13)
memory usage: 23.0 KB
```

All 13 features in **nutri** are (at the moment) interpreted by Python as *quantitative* variables, indeed as integers, simply because they have been entered as whole numbers. The *meaning* of these numbers becomes clear when we consider the description of the features. The following table shows how the variable types should be classified.

Table: The feature types for the data frame **nutri**.

Qualitative	`gender`, `situation`, `fat` `meat`, `fish`, `raw_fruit`, `cooked_fruit_veg`, `chocol`
Discrete quantitative	`tea`, `coffee`
Continuous quantitative	`height`, `weight`, `age`

Table: Description of the variables in the nutritional study.

Feature	**Description**	**Unit** or **Coding**
`gender`	Gender	1=Male; 2=Female
`situation`	Family status	1=Single 2=Living with spouse 3=Living with family 4=Living with someone else
`tea`	Daily consumption of tea	Number of cups
`coffee`	Daily consumption of coffee	Number of cups
`height`	Height	cm
`weight`	Weight (actually: mass)	kg
`age`	Age at date of interview	Years
`meat`	Consumption of meat	0=Never 1=Less than once a week 2=Once a week 3=2–3 times a week 4=4–6 times a week 5=Every day
`fish`	Consumption of fish	As in `meat`
`raw_fruit`	Consumption of raw fruits	As in `meat`
`cooked_fruit_veg`	Consumption of cooked fruits and vegetables	As in `meat`
`chocol`	Consumption of chocolate	As in `meat`
`fat`	Type of fat used for cooking	1=Butter 2=Margarine 3=Peanut oil 4=Sunflower oil 5=Olive oil 6=Mix of vegetable oils (e.g., Isio4) 7=Colza oil 8=Duck or goose fat

We can modify the Python value and type for each categorical feature, using the **`replace`** and **`astype`** methods. For categorical features, such as `gender`, we can replace the value 1 with `'Male'` and 2 with `'Female'`, and change the type to `'category'` as follows.

```
DICT = {1:'Male', 2:'Female'} # dictionary specifies replacement
nutri['gender'] = nutri['gender'].replace(DICT).astype('category')
```

The structure of the other categorical-type features can be changed in a similar way. Continuous features such as `height` should have type `float`:

```
nutri['height'] = nutri['height'].astype(float)
```

Repeat this for the other variables.

Summary Tables

It is often useful to summarize a large spreadsheet of data in a more condensed form.

A table of counts or a table of frequencies makes it easier to gain insight into the underlying distribution of a variable, especially if the data are qualitative.

Such tables can be obtained with the methods `describe` and `value_counts`.

As an example, we load the `nutri` `DataFrame`, which we restructured and saved as `'nutri.csv'`, and then construct a summary for the feature (column) `'fat'`.

```
nutri = pd.read_csv('nutri.csv')
nutri['fat'].describe()
```

```
count           226
unique            8
top       sunflower
freq             68
Name: fat, dtype: object
```

We see that there are 8 different types of fat used and that sunflower has the highest count, with 68 out of 226 individuals using this type of cooking fat.

The method `value_counts` gives the counts for the different fat types.

```
nutri['fat'].value_counts()
```

```
sunflower    68
peanut       48
olive        40
margarine    27
Isio4        23
butter       15
duck          4
colza         1
Name: fat, dtype: int64
```

Column labels are also attributes of a `DataFrame`, and `nutri.fat`, for example, is exactly the same object as `nutri['fat']`.

It is also possible to use **crosstab** to cross tabulate between two or more variables, giving a *contingency table*:

```
pd.crosstab(nutri.gender, nutri.situation)
```

```
situation  Couple  Family  Single
gender
Female         56       7      78
Male           63       2      20
```

We see, for example, that the proportion of single men is substantially smaller than the proportion of single women in the data set of elderly people. To add row and column totals to a table, use `margins=True`.

```
pd.crosstab(nutri.gender, nutri.situation, margins=True)
```

```
situation  Couple  Family  Single  All
gender
Female         56       7      78  141
Male           63       2      20   85
All           119       9      98  226
```

Summary Statistics

In the following, $\boldsymbol{x} = [x_1, \ldots, x_n]^\top$ is a column vector of n numbers. For our **nutri** data, the vector $\boldsymbol{x}$ could, for example, correspond to the heights of the $n = 226$ individuals.

The sample mean of $\boldsymbol{x}$, denoted by $\overline{x}$, is simply the average of the data values:

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i .$$

Using the **mean** method in Python for the **nutri** data, we have, for instance:

```
nutri['height'].mean()
```

```
163.96017699115043
```

The p-sample quantile ($0 < p < 1$) of $\boldsymbol{x}$ is a value x such that at least a fraction p of the data is less than or equal to x and at least a fraction $1 - p$ of the data is greater than or equal to x.

The sample median is the sample 0.5-quantile. The p-sample quantile is also called the $100 \times p$ percentile. The 25, 50, and 75 sample percentiles are called the first, second, and third quartiles of the data.

For the **nutri** data they are obtained as follows.

```
nutri['height'].quantile(q=[0.25,0.5,0.75])
```

```
0.25    157.0
0.50    163.0
0.75    170.0
```

Measures for the dispersion (spread) in the data include sample range, $\max_i x_i - \min_i x_i$, the sample variance

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2, \tag{1}$$

and the sample standard deviation $s = \sqrt{s^2}$.

```
nutri['height'].max() - nutri['height'].min()
```

```
48.0
```

```
round(nutri['height'].var(), 2)  # round to two decimal places
```

```
81.06
```

```
round(nutri['height'].std(), 2)
```

```
9.0
```

When applied to a quantitative feature, the **describe** method returns the minimum, maximum, mean, and the three quartiles.

For example, the 'height' feature in the **nutri** data has the following summary statistics.

```
nutri['height'].describe()
```

```
count    226.000000
mean     163.960177
std        9.003368
min      140.000000
25\%      157.000000
50\%      163.000000
75\%      170.000000
max      188.000000
Name: height, dtype: float64
```

Visualizing Data

Here, we describe various methods for visualizing data.

It is assumed that **matplotlib.pyplot**, **pandas**, and **numpy**, have been imported in the Python code as follows.

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

Plotting Qualitative Variables: Barplot

How many elderly people are living by themselves, as a couple, with family, or other?

```
width = 0.35 # the width of the bars
x = [0, 0.8, 1.6] # the bar positions on x-axis
situation_counts=nutri['situation'].value_counts()
plt.bar(x, situation_counts, width, edgecolor = 'black')
plt.xticks(x, situation_counts.index)
plt.show()
```

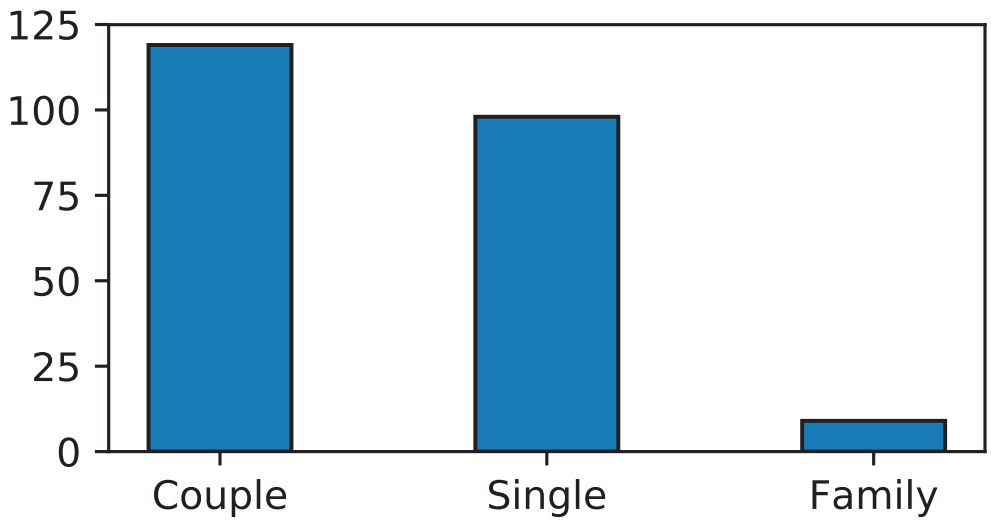


Figure: Barplot for the qualitative variable `'situation'`.

Plotting Quantitative Variables: Boxplot

How can we summarize the `'age'` feature of the **`nutri`** data?

```
plt.boxplot(nutri['age'],widths=width,vert=False)
plt.xlabel('age')
plt.show()
```

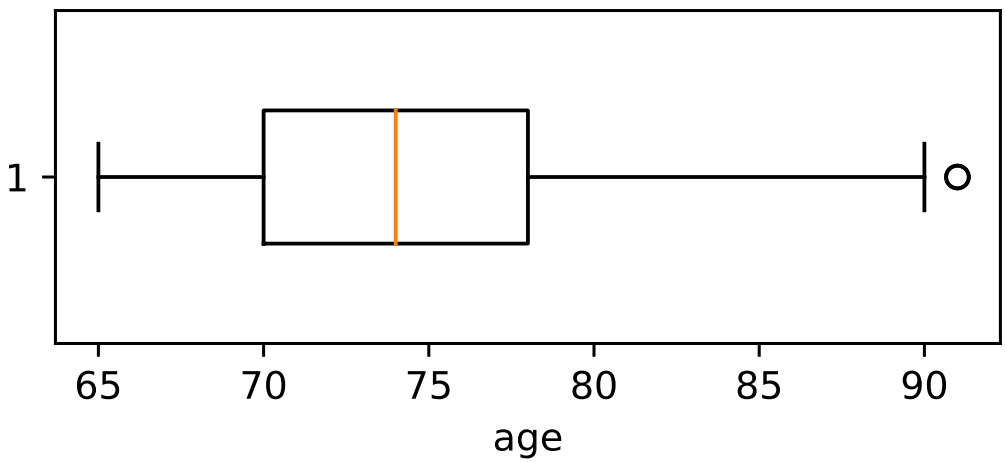
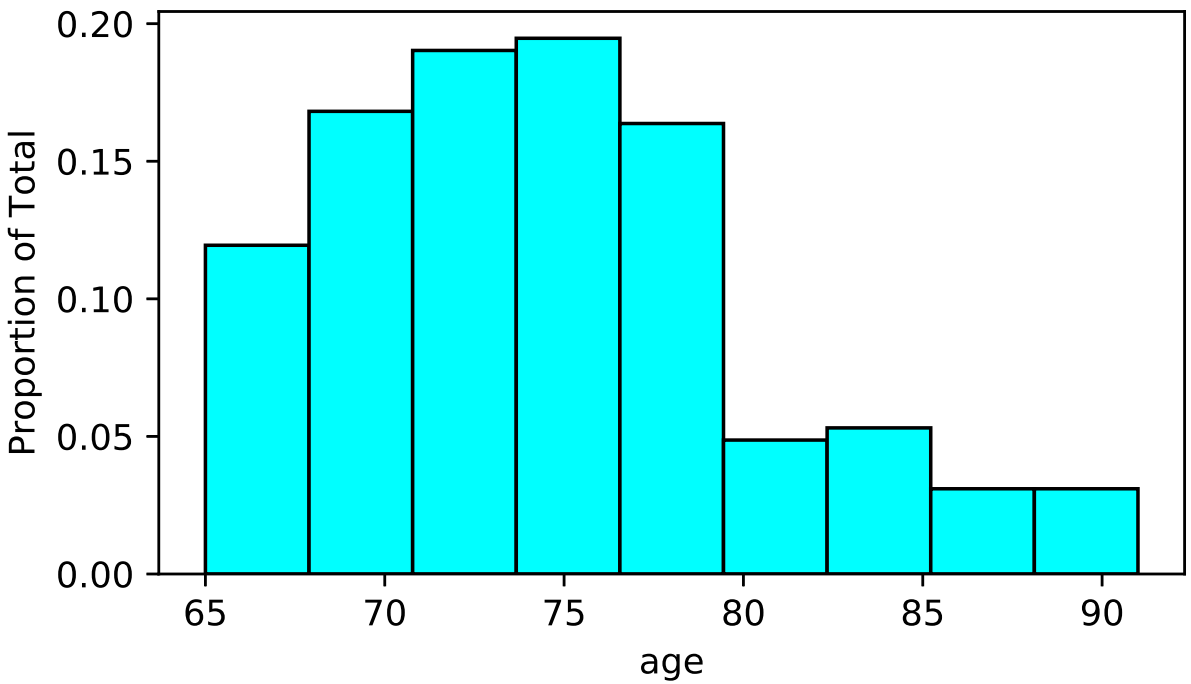


Figure: Boxplot for `'age'`.

Histogram

What is the age distribution of the elderly?

```
weights = np.ones_like(nutri.age)/nutri.age.count()
plt.hist(nutri.age,bins=9,weights=weights,facecolor='cyan',
         edgecolor='black', linewidth=1)
plt.xlabel('age')
plt.ylabel('Proportion of Total')
plt.show()
```



Empirical Cumulative Distribution Function

```
x = np.sort(nutri.age)
y = np.linspace(0,1,len(nutri.age))
plt.xlabel('age')
plt.ylabel('Fn(x)')
plt.step(x,y)
plt.xlim(x.min(),x.max())
```

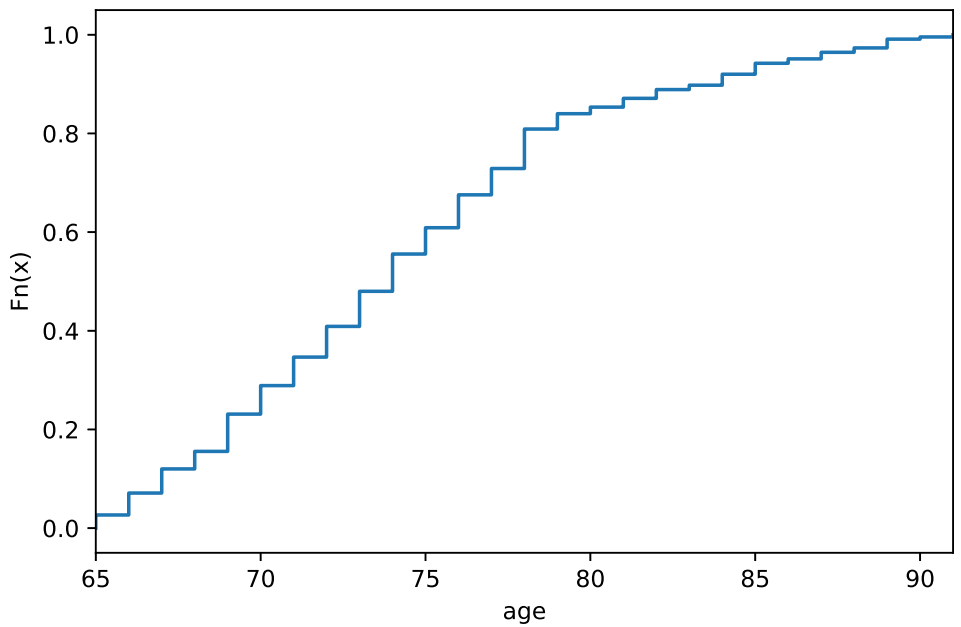


Figure: Plot of the empirical distribution function for the continuous quantitative feature `'age'`.

Data Visualization in a Bivariate Setting

Comparing barplots for two categorical variables involves introducing subplots.

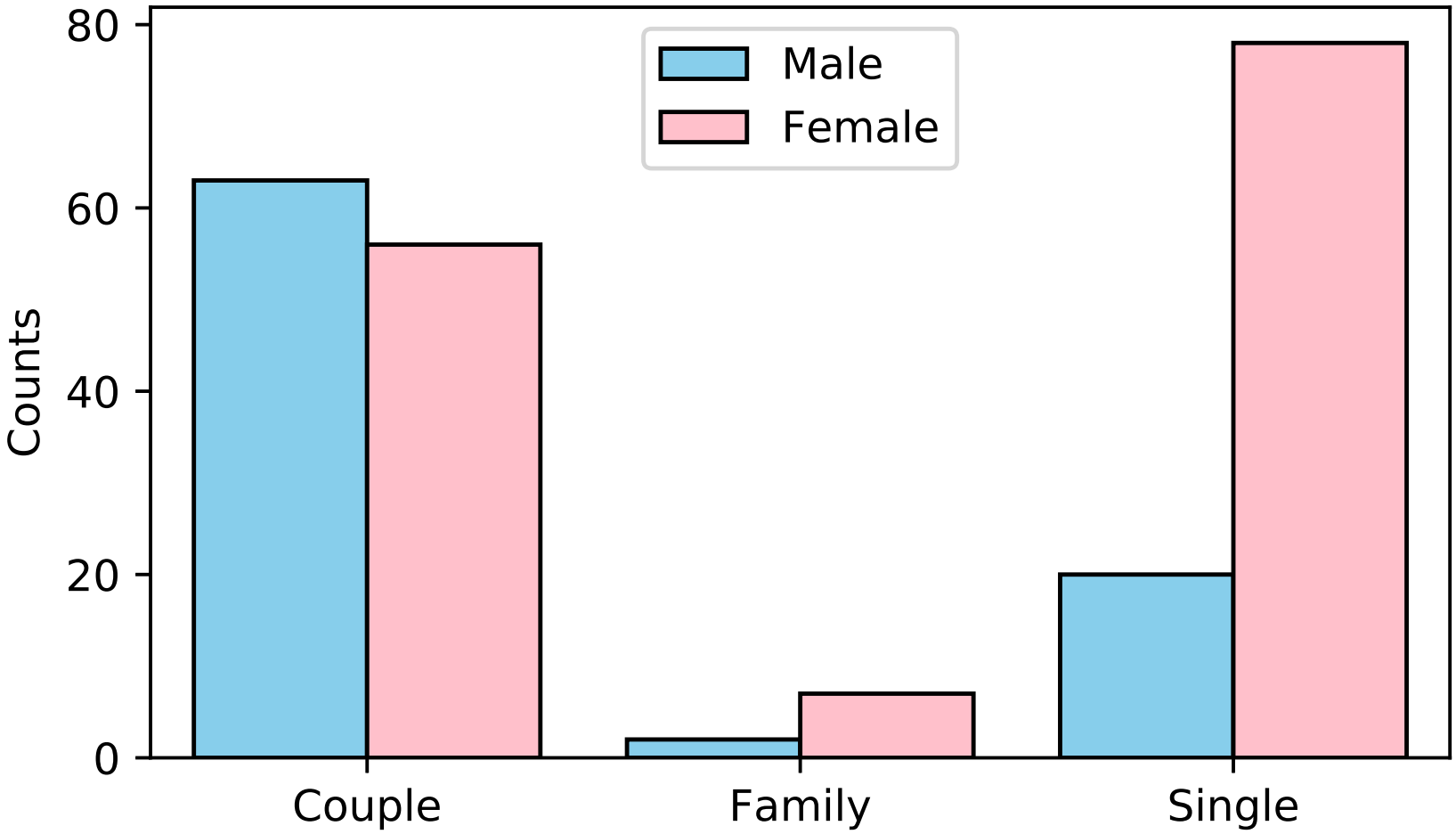


Figure: Barplot for two categorical variables

```
import seaborn as sns
sns.countplot(x='situation', hue = 'gender', data=nutri,
     hue_order = ['Male', 'Female'], palette = ['SkyBlue','Pink'],
      saturation = 1, edgecolor='black')
plt.legend(loc='upper center')
plt.xlabel('')
plt.ylabel('Counts')
```

Plots for Two Quantitative Variables: Scatter Plot

How are height and weight related?

```
plt.scatter(nutri.height, nutri.weight, s=12, marker='o')
plt.xlabel('height')
plt.ylabel('weight')
```

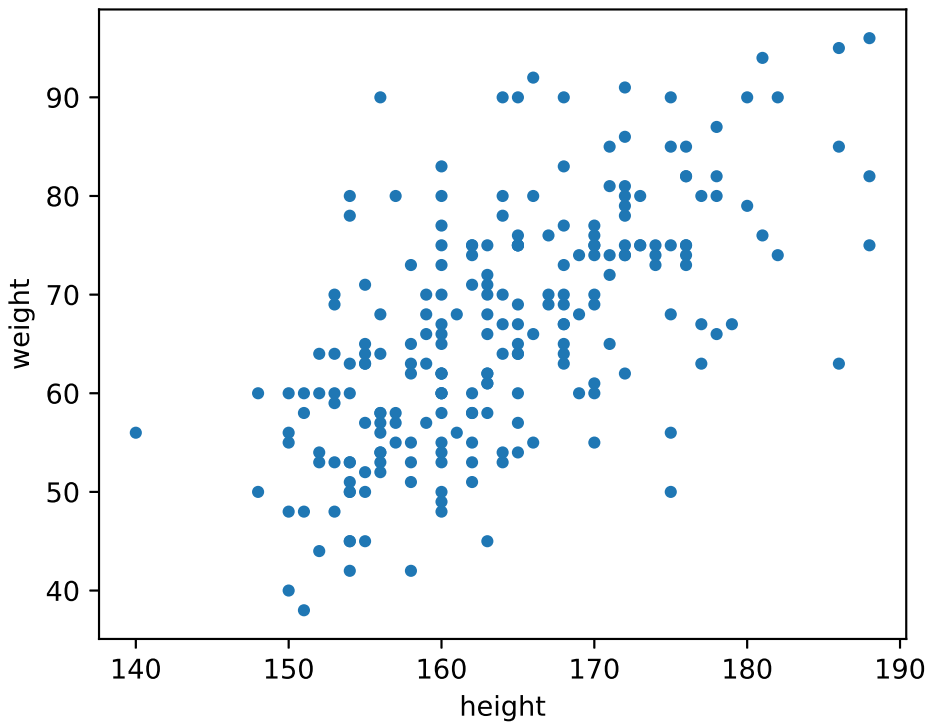


Figure: Scatterplot of `'weight'` against `'height'`.

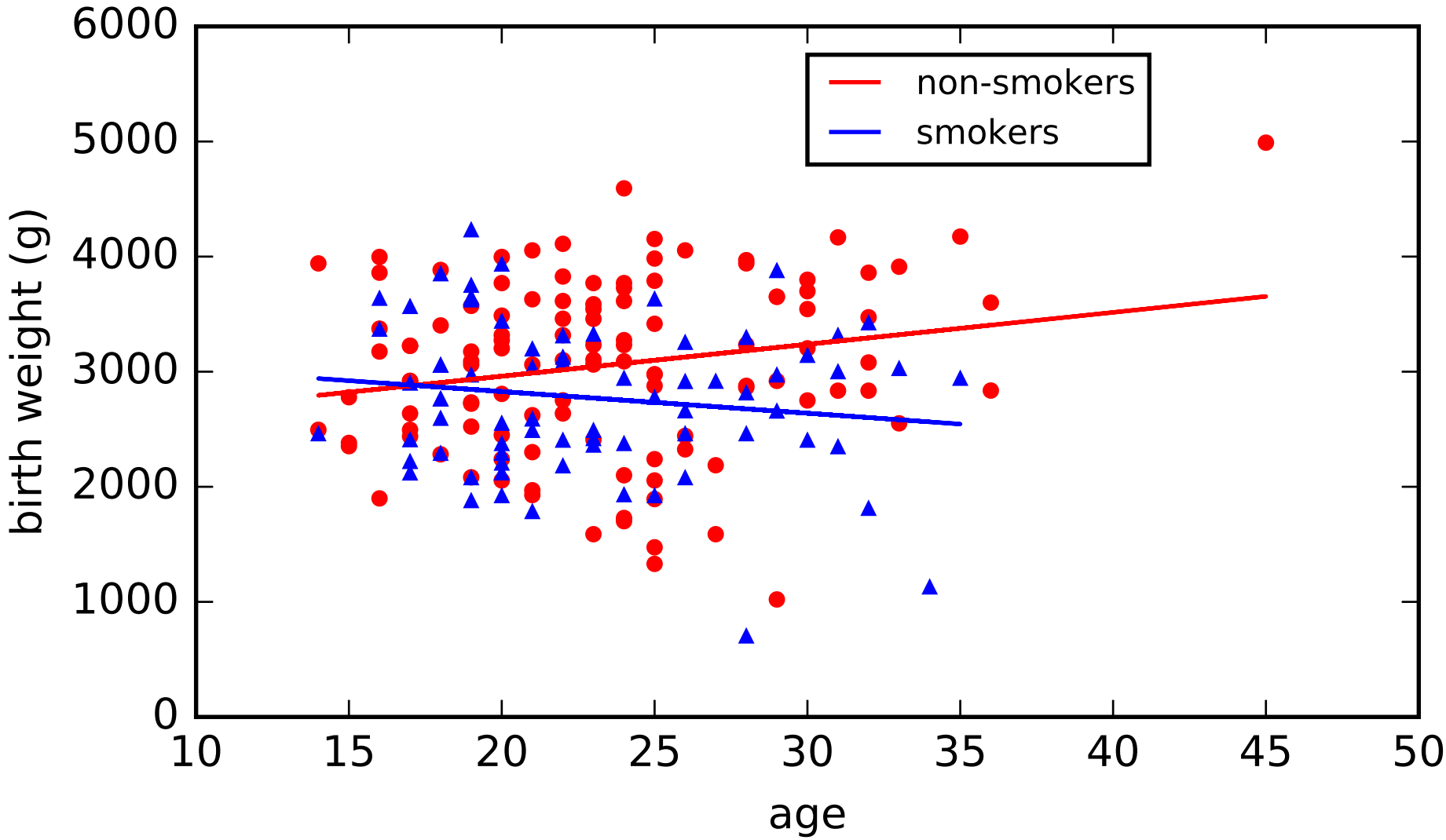


Figure: Birth weight against age for smoking and non-smoking mothers.

Plots for One Qualitative and One Quantitative Variable

```
males = nutri[nutri.gender == 'Male']
females = nutri[nutri.gender == 'Female']
plt.boxplot([males.coffee,females.coffee],notch=True,widths=(0.5,0.5))
plt.xlabel('gender')
plt.ylabel('coffee')
plt.xticks([1,2],['Male','Female'])
```

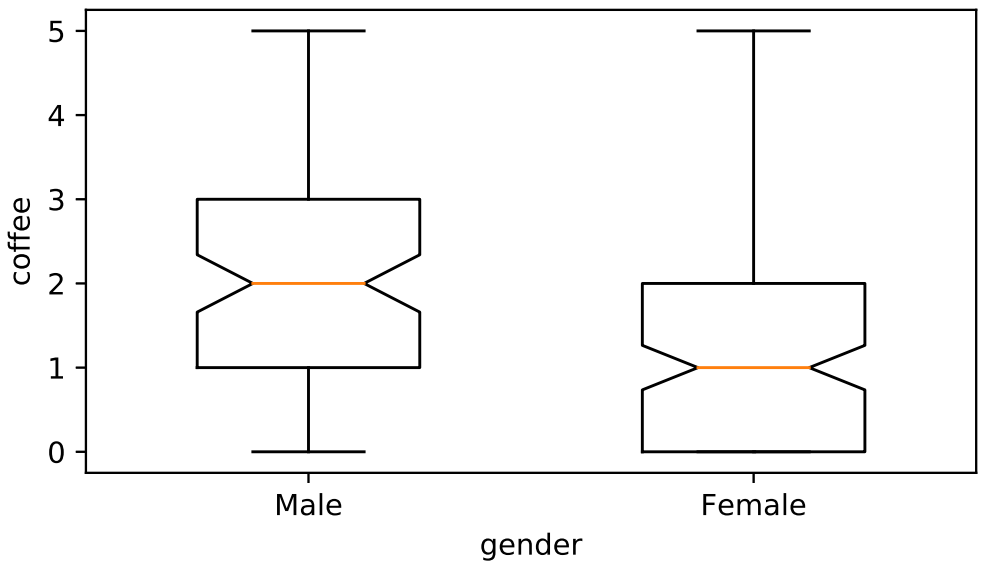


Figure: Boxplots of `'coffee'` as a function of `'gender'`.