CHAPTER

FOUR

STATISTICS

4.1 Univariate statistics

Basics univariate statistics are required to explore dataset:

- Discover associations between a variable of interest and potential predictors. It is strongly recommended to start with simple univariate methods before moving to complex multivariate predictors.
- Assess the prediction performances of machine learning predictors.
- Most of the univariate statistics are based on the linear model which is one of the main model in machine learning.

4.1.1 Estimators of the main statistical measures

Mean

Properties of the expected value operator $\mathrm{E}(\cdot)$ of a random variable X

$$E(X+c) = E(X)+c \tag{4.1}$$
$$E(X+Y) = E(X)+E(Y) \tag{4.2}$$
$$E(aX) = aE(X) \tag{4.3}$$

The estimator $\bar{x}$ on a sample of size n: $x = x_1, ..., x_n$ is given by

$$\bar{x} = \frac{1}{n}\sum_i x_i$$

$\bar{x}$ is itself a random variable with properties:

- $E(\bar{x}) = \bar{x}$,
- $Var(\bar{x}) = \frac{Var(X)}{n}$.

Variance

$$Var(X) = E((X-E(X))^2) = E(X^2) - (E(X))^2$$

The estimator is

$$\sigma_x^2 = \frac{1}{n-1}\sum_i (x_i - \bar{x})^2$$

Note here the subtracted 1 degree of freedom (df) in the divisor. In standard statistical practice, $df = 1$ provides an unbiased estimator of the variance of a hypothetical infinite population. With $df = 0$ it instead provides a maximum likelihood estimate of the variance for normally distributed variables.

Standard deviation

$$Std(X) = \sqrt{Var(X)}$$

The estimator is simply $\sigma_x = \sqrt{\sigma_x^2}$.

Covariance

$$Cov(X, Y) = E((X - E(X))(Y - E(Y))) = E(XY) - E(X)E(Y).$$

Properties:

$$\begin{aligned}\operatorname{Cov}(X, X) &= \operatorname{Var}(X)\\ \operatorname{Cov}(X, Y) &= \operatorname{Cov}(Y, X)\\ \operatorname{Cov}(cX, Y) &= c\operatorname{Cov}(X, Y)\\ \operatorname{Cov}(X + c, Y) &= \operatorname{Cov}(X, Y)\end{aligned}$$

The estimator with $df = 1$ is

$$\sigma_{xy} = \frac{1}{n-1}\sum_i (x_i - \bar{x})(y_i - \bar{y}).$$

Correlation

$$Cor(X, Y) = \frac{Cov(X, Y)}{Std(X)Std(Y)}$$

The estimator is

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}.$$

Standard Error (SE)

The standard error (SE) is the standard deviation (of the sampling distribution) of a statistic:

$$SE(X) = \frac{Std(X)}{\sqrt{n}}.$$

It is most commonly considered for the mean with the estimator $

$$\begin{aligned} SE(\bar{x}) &= \sigma_{\bar{x}} && (4.4)\\ &= \frac{\sigma_x}{\sqrt{n}}. && (4.5)\end{aligned}$$

Exercises

- Generate 2 random samples: $x \sim N(1.78, 0.1)$ and $y \sim N(1.66, 0.1)$, both of size 10.
- Compute $\bar{x}, \sigma_x, \sigma_{xy}$ (`xbar, xvar, xycov`) using only the `np.sum()` operation. Explore the `np.` module to find out which numpy functions performs the same computations and compare them (using `assert`) with your previous results.

4.1.2 Main distributions

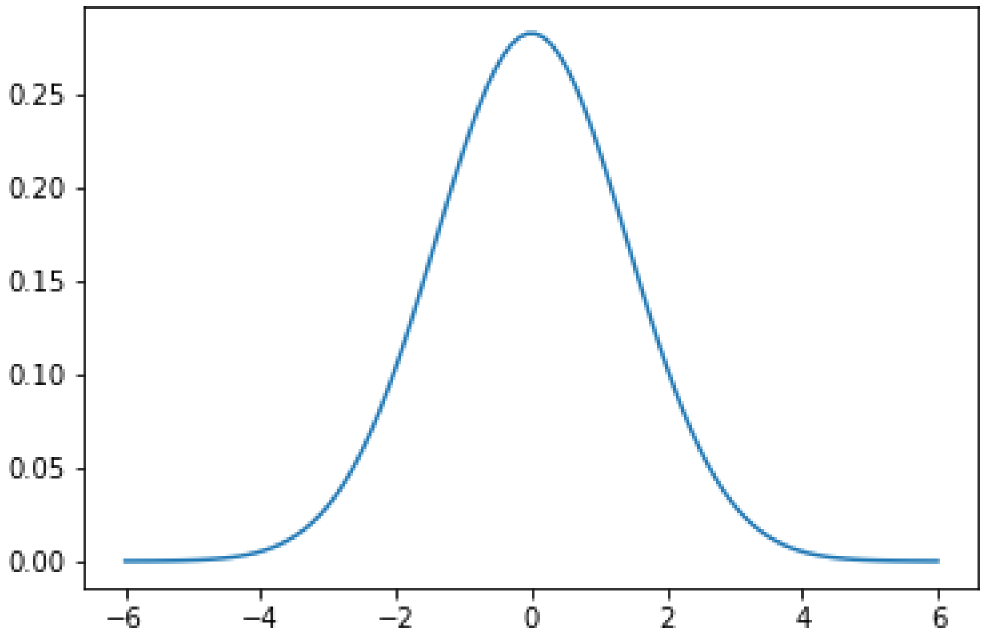
Normal distribution

The normal distribution, noted $\mathcal{N}(\mu, \sigma)$ with parameters: μ mean (location) and $\sigma > 0$ std-dev. Estimators: $\bar{x}$ and σ_x.

The normal distribution, noted
$mathcal N$, is useful because of the central limit theorem (CLT) which states that: given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined expected value and well-defined variance, will be approximately normally distributed, regardless of the underlying distribution.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
%matplotlib inline

mu = 0 # mean
variance = 2 #variance
sigma = np.sqrt(variance) #standard deviation\n",
x = np.linspace(mu-3*variance,mu+3*variance, 100)
plt.plot(x, norm.pdf(x, mu, sigma))
```

```
[<matplotlib.lines.Line2D at 0x7f6f2a4dae48>]
```



The Chi-Square distribution

The chi-square or χ^2_n distribution with n degrees of freedom (df) is the distribution of a sum of the squares of n independent standard normal random variables $\mathcal{N}(0,1)$. Let $X \sim \mathcal{N}(\mu, \sigma^2)$, then, $Z = (X - \mu)/\sigma \sim \mathcal{N}(0,1)$, then:

- The squared standard $Z^2 \sim \chi^2_1$ (one df).
- **The distribution of sum of squares** of n normal random variables: $\sum_i^n Z_i^2 \sim \chi^2_n$

The sum of two χ^2 RV with p and q df is a χ^2 RV with $p + q$ df. This is useful when summing/subtracting sum of squares.

The χ^2-distribution is used to model **errors** measured as **sum of squares** or the distribution of the sample **variance**.

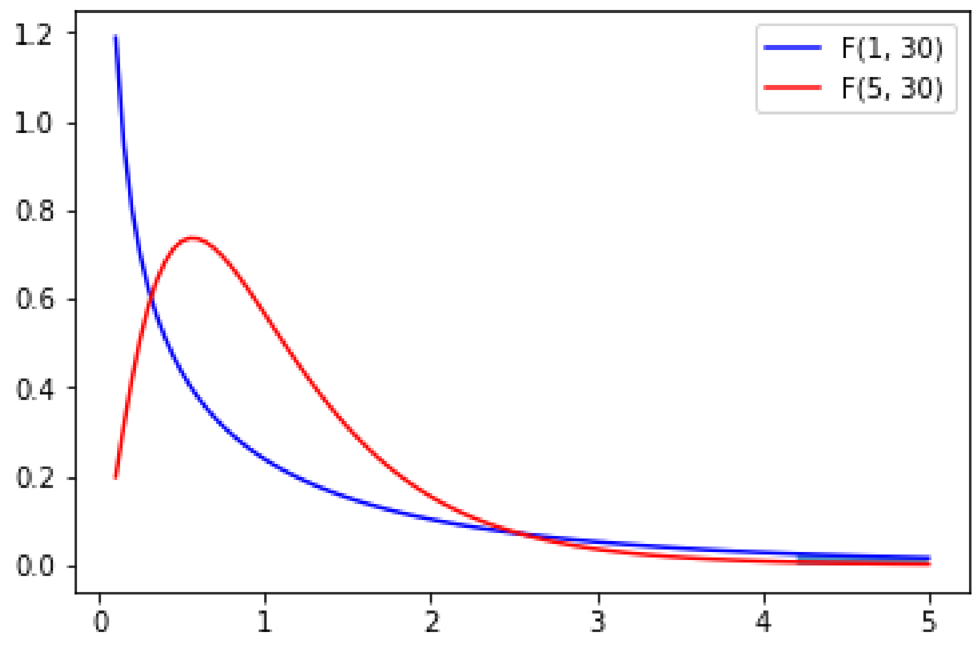
The Fisher's F-distribution

The F-distribution, $F_{n,p}$, with n and p degrees of freedom is the ratio of two independent χ^2 variables. Let $X \sim \chi^2_n$ and $Y \sim \chi^2_p$ then:

$$F_{n,p} = \frac{X/n}{Y/p}$$

The F-distribution plays a central role in hypothesis testing answering the question: **Are two variances equals?, is the ratio or two errors significantly large ?**.

```
import numpy as np
from scipy.stats import f
import matplotlib.pyplot as plt
%matplotlib inline

fvalues = np.linspace(.1, 5, 100)

# pdf(x, df1, df2): Probability density function at x of F.
plt.plot(fvalues, f.pdf(fvalues, 1, 30), 'b-', label="F(1, 30)")
plt.plot(fvalues, f.pdf(fvalues, 5, 30), 'r-', label="F(5, 30)")
plt.legend()

# cdf(x, df1, df2): Cumulative distribution function of F.
# ie.
proba_at_f_inf_3 = f.cdf(3, 1, 30) # P(F(1,30) < 3)

# ppf(q, df1, df2): Percent point function (inverse of cdf) at q of F.
f_at_proba_inf_95 = f.ppf(.95, 1, 30) # q such P(F(1,30) < .95)
assert f.cdf(f_at_proba_inf_95, 1, 30) == .95

# sf(x, df1, df2): Survival function (1 - cdf) at x of F.
proba_at_f_sup_3 = f.sf(3, 1, 30) # P(F(1,30) > 3)
assert  proba_at_f_inf_3 + proba_at_f_sup_3 == 1

# p-value: P(F(1, 30)) < 0.05
low_proba_fvalues = fvalues[fvalues > f_at_proba_inf_95]
plt.fill_between(low_proba_fvalues, 0, f.pdf(low_proba_fvalues, 1, 30),
                 alpha=.8, label="P < 0.05")
plt.show()
```



The Student's t-distribution

Let $M \sim \mathcal{N}(0,1)$ and $V \sim \chi^2_n$. The t-distribution, T_n, with n degrees of freedom is the ratio:

$$T_n = \frac{M}{\sqrt{V/n}}$$

The distribution of the difference between an estimated parameter and its true (or assumed) value divided by the standard deviation of the estimated parameter (standard error) follow a t-distribution. **Is this parameters different from a given value?**

4.1.3 Hypothesis Testing

Examples

- Test a proportion: Biased coin ? 200 heads have been found over 300 flips, is it coins biased ?
- Test the association between two variables.
  - Exemple height and sex: In a sample of 25 individuals (15 females, 10 males), is female height is different from male height ?
  - Exemple age and arterial hypertension: In a sample of 25 individuals is age height correlated with arterial hypertension ?

Steps

1. Model the data
2. Fit: estimate the model parameters (frequency, mean, correlation, regression coeficient)
3. Compute a test statistic from model the parameters.
4. Formulate the null hypothesis: What would be the (distribution of the) test statistic if the observations are the result of pure chance.

5. Compute the probability (p-value) to obtain a larger value for the test statistic by chance (under the null hypothesis).

Flip coin: Simplified example

Biased coin ? 2 heads have been found over 3 flips, is it coins biased ?

1. Model the data: number of heads follow a Binomial disctribution.
2. Compute model parameters: N=3, P = the frequency of number of heads over the number of flip: 2/3.
3. Compute a test statistic, same as frequency.
4. Under the null hypothesis the distribution of the number of tail is:

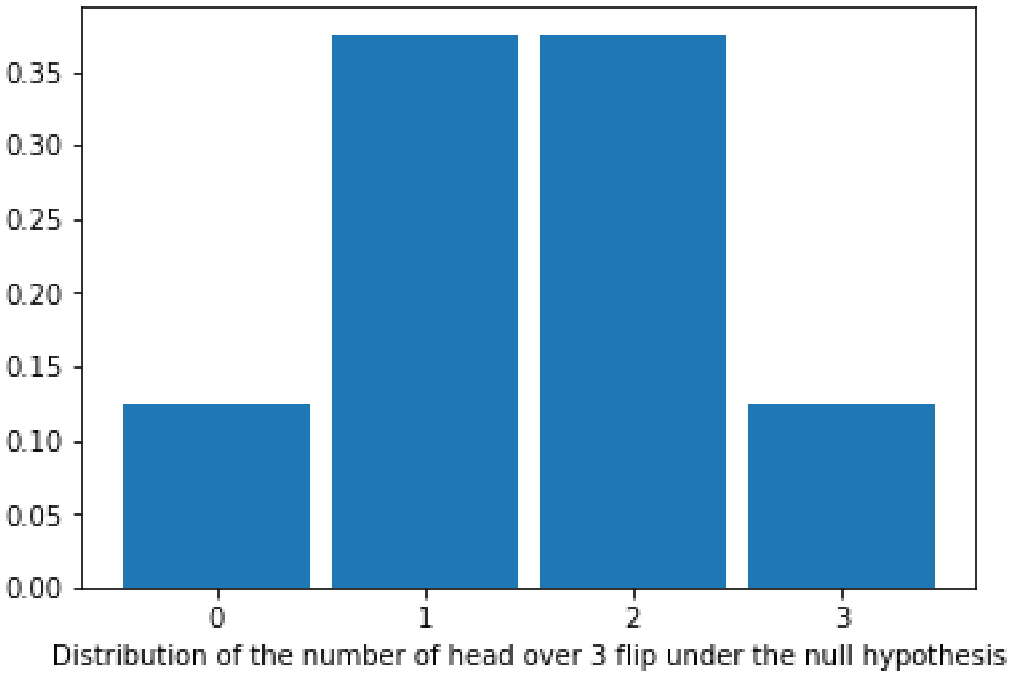
1	2	3	count #heads
			0
H			1
	H		1
		H	1
H	H		2
H		H	2
	H	H	2
H	H	H	3

8 possibles configurations, probabilities of differents values for p are: x measure the number of success.

- $P(x = 0) = 1/8$
- $P(x = 1) = 3/8$
- $P(x = 2) = 3/8$
- $P(x = 3) = 1/8$

```
plt.bar([0, 1, 2, 3], [1/8, 3/8, 3/8, 1/8], width=0.9)
_ = plt.xticks([0, 1, 2, 3], [0, 1, 2, 3])
plt.xlabel("Distribution of the number of head over 3 flip under the null hypothesis")
```

```
Text(0.5, 0, 'Distribution of the number of head over 3 flip under the null hypothesis')
```



3. Compute the probability (p-value) to observe a value larger or equal that 2 under the null hypothesis ? This probability is the p-value:

$$P(x \geq 2|H_0) = P(x = 2) + P(x = 3) = 3/8 + 1/8 = 4/8 = 1/2$$

Flip coin: Real Example

Biased coin ? 60 heads have been found over 100 flips, is it coins biased ?

1. Model the data: number of heads follow a Binomial disctribution.
2. Compute model parameters: N=100, P=60/100.
3. Compute a test statistic, same as frequency.
4. Compute a test statistic: 60/100.
5. Under the null hypothesis the distribution of the number of tail (k) follow the **binomial distribution** of parameters N=100, **P=0.5**:

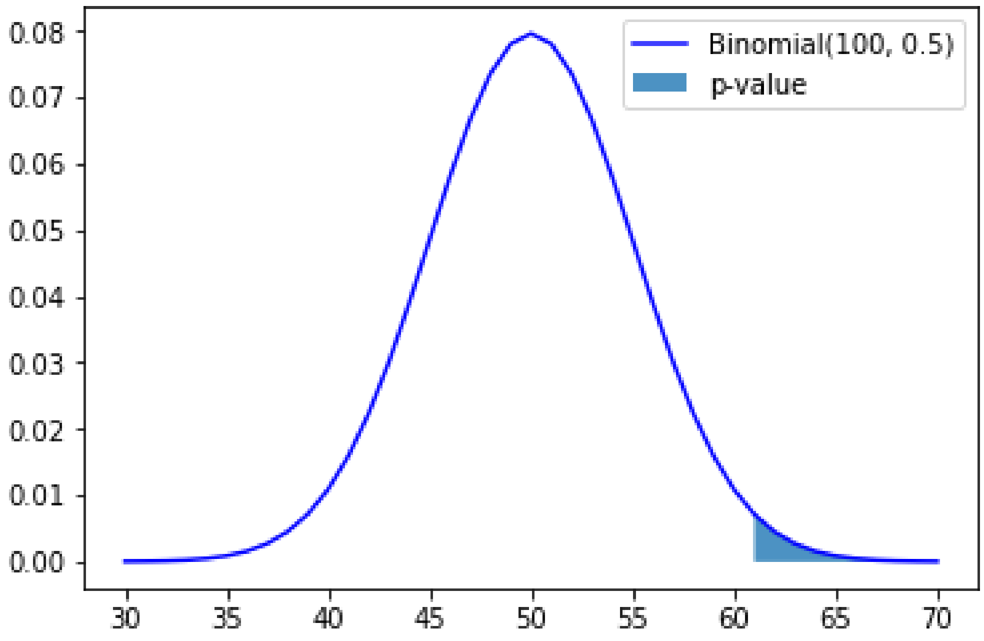
$$Pr(X = k|H_0) = Pr(X = k|n = 100, p = 0.5) = \binom{100}{k} 0.5^k (1 - 0.5)^{(100-k)}.$$

$$\begin{aligned} P(X = k \geq 60|H_0) &= \sum_{k=60}^{100} \binom{100}{k} 0.5^k (1 - 0.5)^{(100-k)} \\ &= 1 - \sum_{k=1}^{60} \binom{100}{k} 0.5^k (1 - 0.5)^{(100-k)}, \text{the cumulative distribution function.} \end{aligned}$$

Use tabulated binomial distribution

```
import scipy.stats
import matplotlib.pyplot as plt

#tobs = 2.39687663116 # assume the t-value
succes = np.linspace(30, 70, 41)
plt.plot(succes, scipy.stats.binom.pmf(succes, 100, 0.5), 'b-', label="Binomial(100, 0.5)
↪")
upper_succes_tvalues = succes[succes > 60]
plt.fill_between(upper_succes_tvalues, 0, scipy.stats.binom.pmf(upper_succes_tvalues, 100,
↪ 0.5), alpha=.8, label="p-value")
_ = plt.legend()


pval = 1 - scipy.stats.binom.cdf(60, 100, 0.5)
print(pval)
```

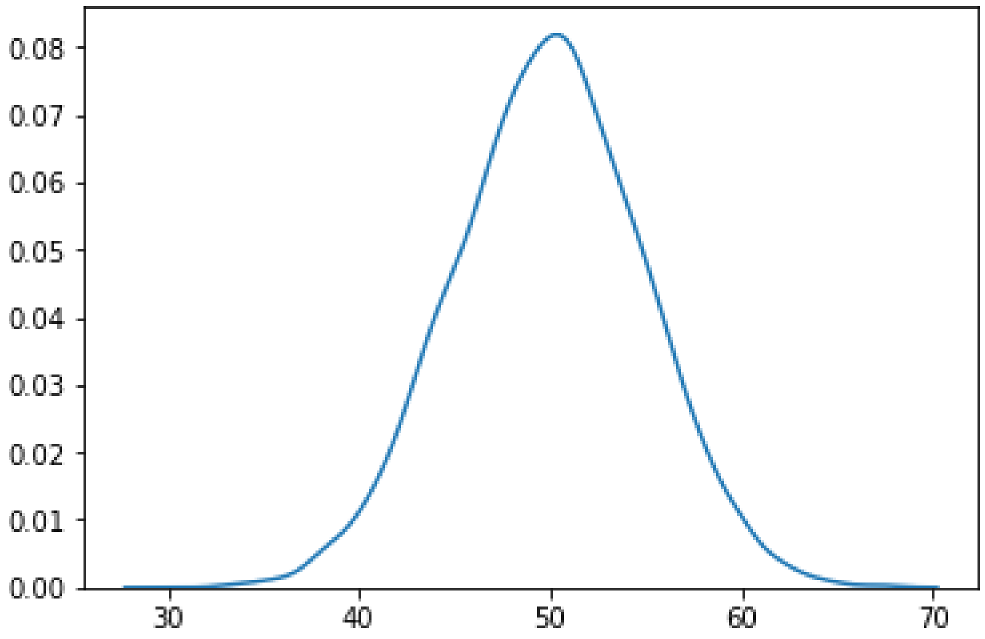
```
0.01760010010885238
```



Random sampling of the Binomial distribution under the null hypothesis

```
sccess_h0 = scipy.stats.binom.rvs(100, 0.5, size=10000, random_state=4)

#sccess_h0 = np.array([) for i in range(5000)])
import seaborn as sns
_ = sns.distplot(sccess_h0, hist=False)

pval_rnd = np.sum(sccess_h0 >= 60) / (len(sccess_h0) + 1)
print("P-value using monte-carlo sampling of the Binomial distribution under H0=", pval_
↪rnd)
```

```
P-value using monte-carlo sampling of the Binomial distribution under H0= 0.
↪025897410258974102
```



One sample t-test

The one-sample t-test is used to determine whether a sample comes from a population with a specific mean. For example you want to test if the average height of a population is $1.75\ m$.

1 Model the data

Assume that height is normally distributed: $X \sim \mathcal{N}(\mu, \sigma)$, ie:

$$\text{height}_i = \text{average height over the population} + \text{error}_i \tag{4.6}$$

$$x_i = \bar{x} + \varepsilon_i \tag{4.7}$$

The ε_i are called the residuals

2 Fit: estimate the model parameters

$\bar{x}, s_x$ are the estimators of μ, σ.

3 Compute a test statistic

In testing the null hypothesis that the population mean is equal to a specified value $\mu_0 = 1.75$, one uses the statistic:

$$t = \frac{\bar{x} - \mu_0}{s_x/\sqrt{n}}$$

Remarks: Although the parent population does not need to be normally distributed, the distribution of the population of sample means, $\overline{x}$, is assumed to be normal. By the central limit theorem, if the sampling of the parent population is independent then the sample means will be approximately normal.

4 Compute the probability of the test statistic under the null hypotheis. This require to have the distribution of the t statistic under H_0.

Example

Given the following samples, we will test whether its true mean is 1.75.

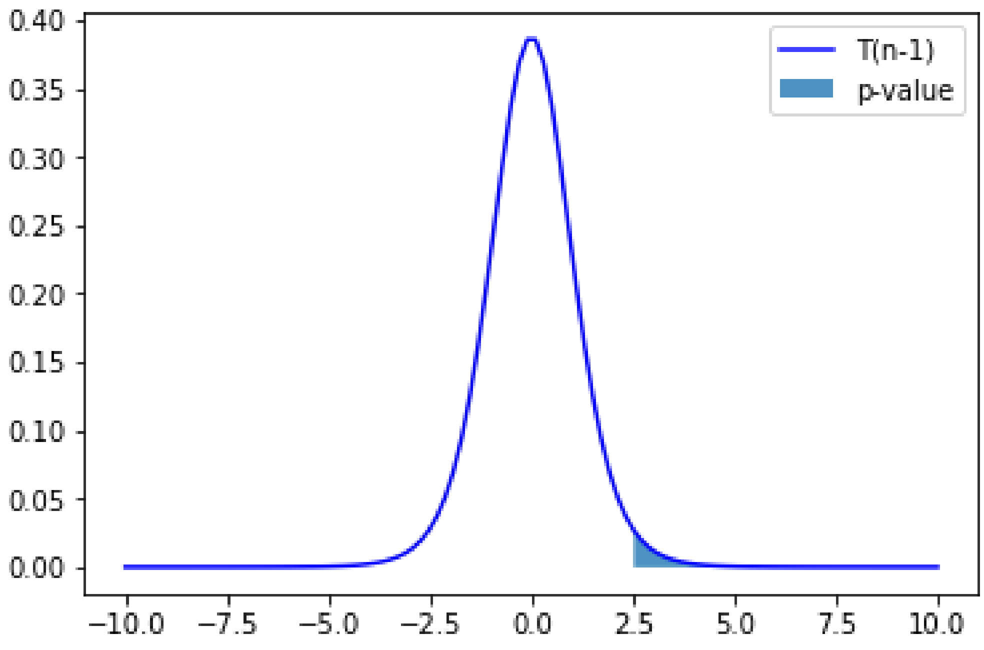
Warning, when computing the std or the variance, set `ddof=1`. The default value, `ddof=0`, leads to the biased estimator of the variance.

```python
import numpy as np

x= [ 1.83,  1.83,  1.73,  1.82,  1.83,  1.73,  1.99,  1.85,  1.68,  1.87]

xbar = np.mean(x) # sample mean
mu0 = 1.75 # hypothesized value
s = np.std(x, ddof=1) # sample standard deviation
n = len(x) # sample size

tobs = (xbar - mu0) / (s / np.sqrt(n))
print(tobs)
```

```
2.3968766311585883
```

The **:math:'p'-value** is the probability to observe a value t more extreme than the observed one t_{obs} under the null hypothesis H_0: $P(t > t_{obs}|H_0)$

```python
import scipy.stats as stats
import matplotlib.pyplot as plt

#tobs = 2.39687663116 # assume the t-value
tvalues = np.linspace(-10, 10, 100)
plt.plot(tvalues, stats.t.pdf(tvalues, n-1), 'b-', label="T(n-1)")
upper_tval_tvalues = tvalues[tvalues > tobs]
plt.fill_between(upper_tval_tvalues, 0, stats.t.pdf(upper_tval_tvalues, n-1), alpha=.8,␣
↪label="p-value")
_ = plt.legend()
```



4.1.4 Testing pairwise associations

Univariate statistical analysis: explore association betweens pairs of variables.

- In statistics, a **categorical variable** or **factor** is a variable that can take on one of a limited, and usually fixed, number of possible values, thus assigning each individual to a particular group or "category". The levels are the possibles values of the variable. Number of levels = 2: binomial; Number of levels > 2: multinomial. There is no intrinsic ordering to the categories. For example, gender is a categorical variable having two categories (male and female) and there is no intrinsic ordering to the categories. For example, Sex (Female, Male), Hair color (blonde, brown, etc.).
- An **ordinal variable** is a categorical variable with a clear ordering of the levels. For example: drinks per day (none, small, medium and high).
- A **continuous** or **quantitative variable** $x \in \mathbb{R}$ is one that can take any value in a range of possible values, possibly infinite. E.g.: salary, experience in years, weight.

What statistical test should I use?

See: http://www.ats.ucla.edu/stat/mult_pkg/whatstat/

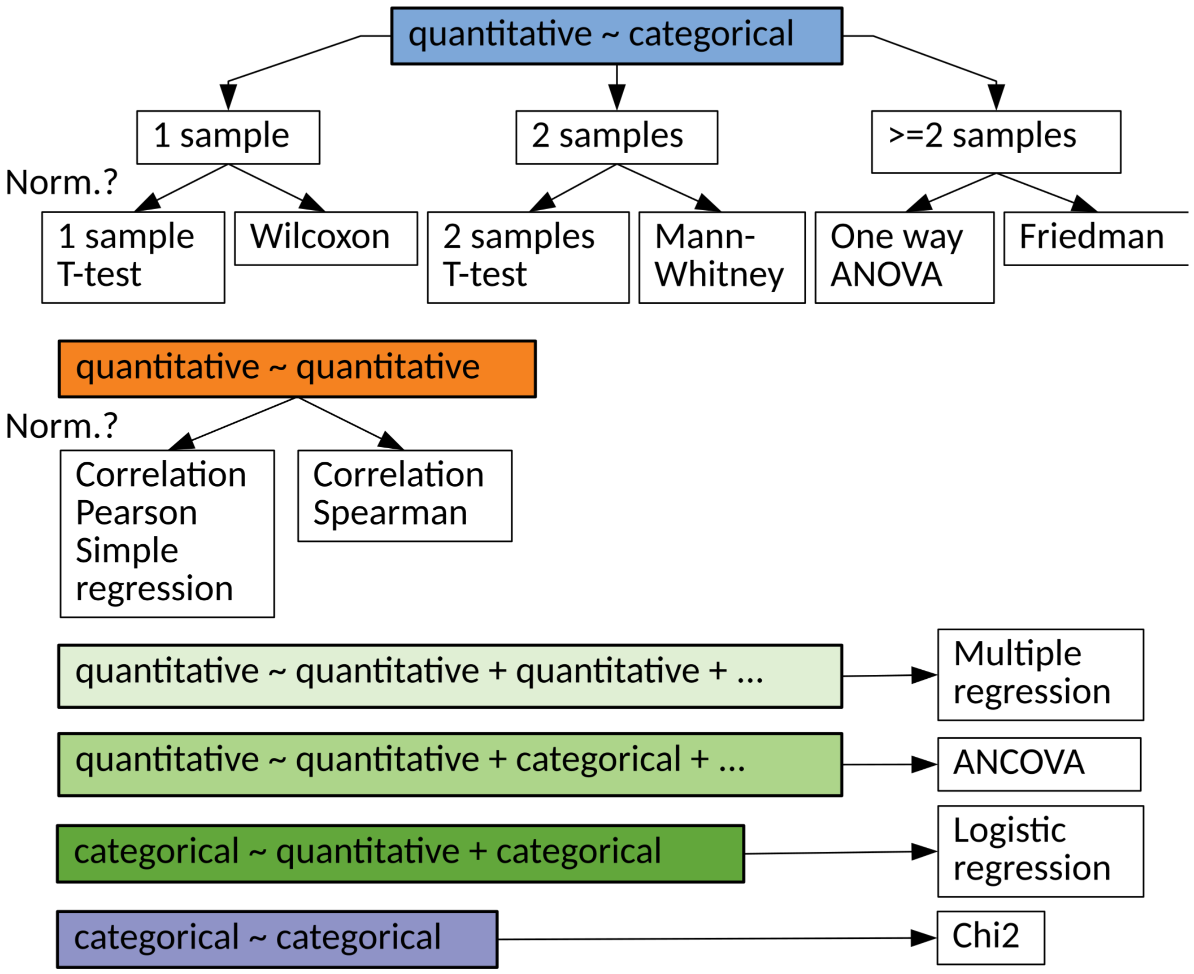

quantitative ~ categorical
1 sample
2 samples
>=2 samples
Norm.?
1 sample T-test
Wilcoxon
2 samples T-test
Mann-Whitney
One way ANOVA
Friedman
quantitative ~ quantitative
Norm.?
Correlation Pearson Simple regression
Correlation Spearman
quantitative ~ quantitative + quantitative + ...
Multiple regression
quantitative ~ quantitative + categorical + ...
ANCOVA
categorical ~ quantitative + categorical
Logistic regression
categorical ~ categorical
Chi2

Fig. 1: Statistical tests

### Pearson correlation test: test association between two quantitative variables

Test the correlation coefficient of two quantitative variables. The test calculates a Pearson correlation coefficient and the p-value for testing non-correlation.

Let x and y two quantitative variables, where n samples were obeserved. The linear correlation coeficient is defined as :

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

Under H_0, the test statistic $t = \sqrt{n-2}\frac{r}{\sqrt{1-r^2}}$ follow Student distribution with $n-2$ degrees of freedom.

```python
import numpy as np
import scipy.stats as stats
n = 50
x = np.random.normal(size=n)
y = 2 * x + np.random.normal(size=n)

# Compute with scipy
cor, pval = stats.pearsonr(x, y)
```

Two sample (Student) t-test: compare two means

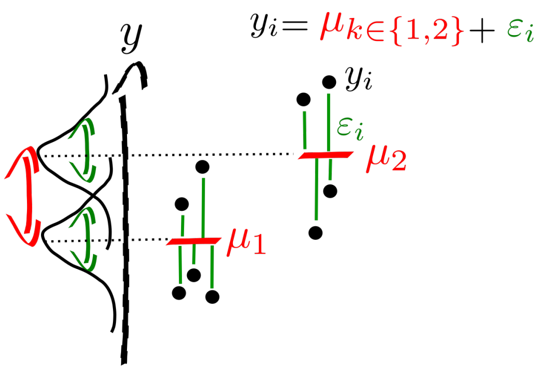


Fig. 2: Two-sample model

The two-sample t-test (Snedecor and Cochran, 1989) is used to determine if two population means are equal. There are several variations on this test. If data are paired (e.g. 2 measures, before and after treatment for each individual) use the one-sample t-test of the difference. The variances of the two samples may be assumed to be equal (a.k.a. homoscedasticity) or unequal (a.k.a. heteroscedasticity).

1. Model the data

Assume that the two random variables are normally distributed: $y_1 \sim \mathcal{N}(\mu_1, \sigma_1), y_2 \sim \mathcal{N}(\mu_2, \sigma_2)$.

2. Fit: estimate the model parameters

Estimate means and variances: $\bar{y}_1, s^2_{y_1}, \bar{y}_2, s^2_{y_2}$.

3. t-test

The general principle is

$$t = \frac{\text{difference of means}}{\text{its standard error}} \tag{4.8}$$
$$= \frac{\bar{y_1} - \bar{y_2}}{s_{\bar{y_1} - \bar{y_2}}} \tag{4.9}$$

Since y_1 and y_2 are independant:

$$s^2_{\bar{y_1} - \bar{y_2}} = s^2_{\bar{y_1}} + s^2_{\bar{y_2}} = \frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2} \tag{4.10}$$
$$\text{thus} \tag{4.11}$$
$$s_{\bar{y_1} - \bar{y_2}} = \sqrt{\frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2}} \tag{4.12}$$

Equal or unequal sample sizes, unequal variances (Welch's t-test)

Welch's t-test defines the t statistic as

$$t = \frac{\bar{y_1} - \bar{y_2}}{\sqrt{\frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2}}}.$$

To compute the p-value one needs the degrees of freedom associated with this variance estimate. It is approximated using the Welch–Satterthwaite equation:

$$\nu \approx \frac{\left(\frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2}\right)^2}{\frac{s^4_{y_1}}{n_1^2(n_1-1)} + \frac{s^4_{y_2}}{n_2^2(n_2-1)}}.$$

Equal or unequal sample sizes, equal variances

If we assume equal variance (ie, $s^2_{y_1} = s^2_{y_1} = s^2$), where s^2 is an estimator of the common variance of the two samples:

$$s^2 = \frac{s^2_{y_1}(n_1 - 1) + s^2_{y_2}(n_2 - 1)}{n_1 + n_2 - 2} \tag{4.13}$$
$$= \frac{\sum_i^{n_1}(y_{1i} - \bar{y_1})^2 + \sum_j^{n_2}(y_{2j} - \bar{y_2})^2}{(n_1 - 1) + (n_2 - 1)} \tag{4.14}$$

then

$$s_{\bar{y_1} - \bar{y_2}} = \sqrt{\frac{s^2}{n_1} + \frac{s^2}{n_2}} = s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

Therefore, the t statistic, that is used to test whether the means are different is:

$$t = \frac{\bar{y_1} - \bar{y_2}}{s \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}},$$

Equal sample sizes, equal variances

If we simplify the problem assuming equal samples of size $n_1 = n_2 = n$ we get

$$t = \frac{\bar{y_1} - \bar{y_2}}{s\sqrt{2}} \cdot \sqrt{n} \tag{4.15}$$

$$\approx \text{effect size} \cdot \sqrt{n} \tag{4.16}$$

$$\approx \frac{\text{difference of means}}{\text{standard deviation of the noise}} \cdot \sqrt{n} \tag{4.17}$$

Example

Given the following two samples, test whether their means are equal using the **standard t-test, assuming equal variance**.

```python
import scipy.stats as stats

height = np.array([ 1.83,  1.83,  1.73,  1.82,  1.83,  1.73,  1.99,  1.85,  1.68,  1.87,
                    1.66,  1.71,  1.73,  1.64,  1.70,  1.60,  1.79,  1.73,  1.62,  1.77])

grp = np.array(["M"] * 10 + ["F"] * 10)

# Compute with scipy
print(stats.ttest_ind(height[grp == "M"], height[grp == "F"], equal_var=True))
```

```
Ttest_indResult(statistic=3.5511519888466885, pvalue=0.00228208937112721)
```

ANOVA F-test (quantitative ~ categorial (>2 levels))

Analysis of variance (ANOVA) provides a statistical test of whether or not the means of several groups are equal, and therefore generalizes the t-test to more than two groups. ANOVAs are useful for comparing (testing) three or more means (groups or variables) for statistical significance. It is conceptually similar to multiple two-sample t-tests, but is less conservative.

Here we will consider one-way ANOVA with one independent variable, ie one-way anova.

Wikipedia:

- Test if any group is on average superior, or inferior, to the others versus the null hypothesis that all four strategies yield the same mean response
- Detect any of several possible differences.
- The advantage of the ANOVA F-test is that we do not need to pre-specify which strategies are to be compared, and we do not need to adjust for making multiple comparisons.

- The disadvantage of the ANOVA F-test is that if we reject the null hypothesis, we do not know which strategies can be said to be significantly different from the others.

1. Model the data

A company has applied three marketing strategies to three samples of customers in order increase their business volume. The marketing is asking whether the strategies led to different increases of business volume. Let y_1, y_2 and y_3 be the three samples of business volume increase.

Here we assume that the three populations were sampled from three random variables that are normally distributed. I.e., $Y_1 \sim N(\mu_1, \sigma_1), Y_2 \sim N(\mu_2, \sigma_2)$ and $Y_3 \sim N(\mu_3, \sigma_3)$.

2. Fit: estimate the model parameters

Estimate means and variances: $\bar{y}_i, \sigma_i, \quad \forall i \in \{1, 2, 3\}$.

3. F-test

The formula for the one-way ANOVA F-test statistic is

$$F = \frac{\text{Explained variance}}{\text{Unexplained variance}} \tag{4.18}$$

$$= \frac{\text{Between-group variability}}{\text{Within-group variability}} = \frac{s_B^2}{s_W^2}. \tag{4.19}$$

The "explained variance", or "between-group variability" is

$$s_B^2 = \sum_i n_i(\bar{y}_{i\cdot} - \bar{y})^2/(K-1),$$

where $\bar{y}_{i\cdot}$ denotes the sample mean in the ith group, n_i is the number of observations in the ith group, $\bar{y}$ denotes the overall mean of the data, and K denotes the number of groups.

The "unexplained variance", or "within-group variability" is

$$s_W^2 = \sum_{ij}(y_{ij} - \bar{y}_{i\cdot})^2/(N-K),$$

where y_{ij} is the jth observation in the ith out of K groups and N is the overall sample size. This F-statistic follows the F-distribution with $K-1$ and $N-K$ degrees of freedom under the null hypothesis. The statistic will be large if the between-group variability is large relative to the within-group variability, which is unlikely to happen if the population means of the groups all have the same value.

Note that when there are only two groups for the one-way ANOVA F-test, $F = t^2$ where t is the Student's t statistic.

Chi-square, χ^2 (categorial ~ categorial)

Computes the chi-square, χ^2, statistic and p-value for the hypothesis test of independence of frequencies in the observed contingency table (cross-table). The observed frequencies are tested against an expected contingency table obtained by computing expected frequencies based on the marginal sums under the assumption of independence.

Example: 20 participants: 10 exposed to some chemical product and 10 non exposed (exposed = 1 or 0). Among the 20 participants 10 had cancer 10 not (cancer = 1 or 0). χ^2 tests the association between those two variables.

```
import numpy as np
import pandas as pd
import scipy.stats as stats

# Dataset:
# 15 samples:
# 10 first exposed
exposed = np.array([1] * 10 + [0] * 10)
# 8 first with cancer, 10 without, the last two with.
cancer = np.array([1] * 8 + [0] * 10 + [1] * 2)

crosstab = pd.crosstab(exposed, cancer, rownames=['exposed'],
                       colnames=['cancer'])
print("Observed table:")
print("---------------")
print(crosstab)

chi2, pval, dof, expected = stats.chi2_contingency(crosstab)
print("Statistics:")
print("-----------")
print("Chi2 = %f, pval = %f" % (chi2, pval))
print("Expected table:")
print("---------------")
print(expected)
```

```
Observed table:
---------------
cancer   0  1
exposed
0        8  2
1        2  8
Statistics:
-----------
Chi2 = 5.000000, pval = 0.025347
Expected table:
---------------
[[5. 5.]
 [5. 5.]]
```

Computing expected cross-table

```
# Compute expected cross-table based on proportion
exposed_marg = crosstab.sum(axis=0)
exposed_freq = exposed_marg / exposed_marg.sum()

```

(continues on next page)

(continued from previous page)

```
cancer_marg = crosstab.sum(axis=1)
cancer_freq = cancer_marg / cancer_marg.sum()

print('Exposed frequency? Yes: %.2f' % exposed_freq[0],
      'No: %.2f' % exposed_freq[1])
print('Cancer frequency? Yes: %.2f' % cancer_freq[0],
      'No: %.2f' % cancer_freq[1])

print('Expected frequencies:')
print(np.outer(exposed_freq, cancer_freq))

print('Expected cross-table (frequencies * N): ')
print(np.outer(exposed_freq, cancer_freq) * len(exposed))
```

```
Exposed frequency? Yes: 0.50 No: 0.50
Cancer frequency? Yes: 0.50 No: 0.50
Expected frequencies:
[[0.25 0.25]
 [0.25 0.25]]
Expected cross-table (frequencies * N):
[[5. 5.]
 [5. 5.]]
```

4.1.5 Non-parametric test of pairwise associations

### Spearman rank-order correlation (quantitative $\sim$ quantitative)

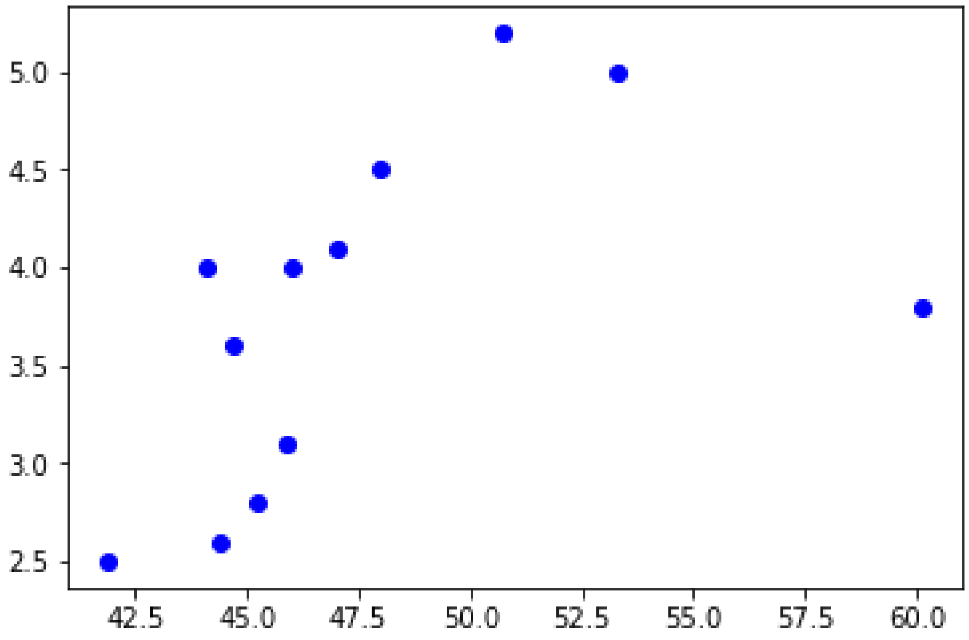
The Spearman correlation is a non-parametric measure of the monotonicity of the relationship between two datasets.

When to use it? Observe the data distribution: - presence of **outliers** - the distribution of the residuals is not Gaussian.

Like other correlation coefficients, this one varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases.

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

x = np.array([44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 46, 47, 48, 60.1])
y = np.array([2.6,  3.1,  2.5,  5.0,  3.6,  4.0,  5.2,  2.8, 4, 4.1, 4.5, 3.8])

plt.plot(x, y, "bo")

# Non-Parametric Spearman
cor, pval = stats.spearmanr(x, y)
print("Non-Parametric Spearman cor test, cor: %.4f, pval: %.4f" % (cor, pval))

# "Parametric Pearson cor test
cor, pval = stats.pearsonr(x, y)
print("Parametric Pearson cor test: cor: %.4f, pval: %.4f" % (cor, pval))
```

```
Non-Parametric Spearman cor test, cor: 0.7110, pval: 0.0095
Parametric Pearson cor test: cor: 0.5263, pval: 0.0788
```



Wilcoxon signed-rank test (quantitative ~ cte)

Source: https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ (i.e. it is a paired difference test). It is equivalent to one-sample test of the difference of paired samples.

It can be used as an alternative to the paired Student's t-test, t-test for matched pairs, or the t-test for dependent samples when the population cannot be assumed to be normally distributed.

When to use it? Observe the data distribution: - presence of outliers - the distribution of the residuals is not Gaussian

It has a lower sensitivity compared to t-test. May be problematic to use when the sample size is small.

Null hypothesis H_0: difference between the pairs follows a symmetric distribution around zero.

```python
import scipy.stats as stats
n = 20
# Buisness Volume time 0
bv0 = np.random.normal(loc=3, scale=.1, size=n)
# Buisness Volume time 1
bv1 = bv0 + 0.1 + np.random.normal(loc=0, scale=.1, size=n)

# create an outlier
bv1[0] -= 10

# Paired t-test
```

(continues on next page)

(continued from previous page)

```
print(stats.ttest_rel(bv0, bv1))

# Wilcoxon
print(stats.wilcoxon(bv0, bv1))
```

```
Ttest_relResult(statistic=0.7821450892478711, pvalue=0.4437681541620575)
WilcoxonResult(statistic=35.0, pvalue=0.008967599455194583)
```

Mann–Whitney U test (quantitative ~ categorial (2 levels))

In statistics, the Mann–Whitney U test (also called the Mann–Whitney–Wilcoxon, Wilcoxon rank-sum test or Wilcoxon–Mann–Whitney test) is a nonparametric test of the null hypothesis that two samples come from the same population against an alternative hypothesis, especially that a particular population tends to have larger values than the other.

It can be applied on unknown distributions contrary to e.g. a t-test that has to be applied only on normal distributions, and it is nearly as efficient as the t-test on normal distributions.

```
import scipy.stats as stats
n = 20
# Buismess Volume group 0
bv0 = np.random.normal(loc=1, scale=.1, size=n)

# Buismess Volume group 1
bv1 = np.random.normal(loc=1.2, scale=.1, size=n)

# create an outlier
bv1[0] -= 10

# Two-samples t-test
print(stats.ttest_ind(bv0, bv1))

# Wilcoxon
print(stats.mannwhitneyu(bv0, bv1))
```

```
Ttest_indResult(statistic=0.6725314683035514, pvalue=0.505314623871812)
MannwhitneyuResult(statistic=67.0, pvalue=0.0001690974050146689)
```

4.1.6 Linear model

Given n random samples $(y_i, x_{1i}, \ldots, x_{pi})$, $i = 1, \ldots, n$, the linear regression models the relation between the observations y_i and the independent variables x_i^p is formulated as

$$y_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} + \varepsilon_i \qquad i = 1, \ldots, n$$

- The β's are the model parameters, ie, the regression coeficients.
- β_0 is the intercept or the bias.
- ε_i are the **residuals**.
- **An independent variable (IV)**. It is a variable that stands alone and isn't changed by the other variables you are trying to measure. For example, someone's age might be an

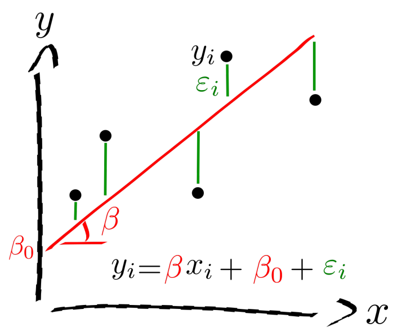


Fig. 3: Linear model

independent variable. Other factors (such as what they eat, how much they go to school, how much television they watch) aren't going to change a person's age. In fact, when you are looking for some kind of relationship between variables you are trying to see if the independent variable causes some kind of change in the other variables, or dependent variables. In Machine Learning, these variables are also called the **predictors**.

- A **dependent variable**. It is something that depends on other factors. For example, a test score could be a dependent variable because it could change depending on several factors such as how much you studied, how much sleep you got the night before you took the test, or even how hungry you were when you took it. Usually when you are looking for a relationship between two things you are trying to find out what makes the dependent variable change the way it does. In Machine Learning this variable is called a **target variable**.

Simple regression: test association between two quantitative variables

Using the dataset "salary", explore the association between the dependant variable (e.g. Salary) and the independent variable (e.g.: Experience is quantitative).

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
salary = pd.read_csv(url)
```

1. Model the data

Model the data on some **hypothesis** e.g.: salary is a linear function of the experience.

$$\text{salary}_i = \beta\ \text{experience}_i + \beta_0 + \epsilon_i,$$

more generally

$$y_i = \beta\ x_i + \beta_0 + \epsilon_i$$

- β: the slope or coefficient or parameter of the model,
- β_0: the **intercept** or **bias** is the second parameter of the model,

- ϵ_i: is the ith error, or residual with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

The simple regression is equivalent to the Pearson correlation.

2. Fit: estimate the model parameters

The goal it so estimate β, β_0 and σ^2.

Minimizes the **mean squared error (MSE)** or the **Sum squared error (SSE)**. The so-called **Ordinary Least Squares (OLS)** finds β, β_0 that minimizes the $SSE = \sum_i \epsilon_i^2$

$$SSE = \sum_i (y_i - \beta\ x_i - \beta_0)^2$$

Recall from calculus that an extreme point can be found by computing where the derivative is zero, i.e. to find the intercept, we perform the steps:

$$\begin{aligned}\frac{\partial SSE}{\partial \beta_0} &= \sum_i (y_i - \beta\ x_i - \beta_0) = 0\\ \sum_i y_i &= \beta\ \sum_i x_i + n\ \beta_0\\ n\ \bar{y} &= n\ \beta\ \bar{x} + n\ \beta_0\\ \beta_0 &= \bar{y} - \beta\ \bar{x}\end{aligned}$$

To find the regression coefficient, we perform the steps:

$$\frac{\partial SSE}{\partial \beta} = \sum_i x_i (y_i - \beta\ x_i - \beta_0) = 0$$

Plug in β_0:

$$\begin{aligned}\sum_i x_i (y_i - \beta\ x_i - \bar{y} + \beta \bar{x}) = 0\\ \sum_i x_i y_i - \bar{y} \sum_i x_i = \beta \sum_i (x_i - \bar{x})\end{aligned}$$

Divide both sides by n:

$$\begin{aligned}\frac{1}{n} \sum_i x_i y_i - \bar{y}\bar{x} = \frac{1}{n} \beta \sum_i (x_i - \bar{x})\\ \beta = \frac{\frac{1}{n} \sum_i x_i y_i - \bar{y}\bar{x}}{\frac{1}{n} \sum_i (x_i - \bar{x})} = \frac{Cov(x, y)}{Var(x)}.\end{aligned}$$

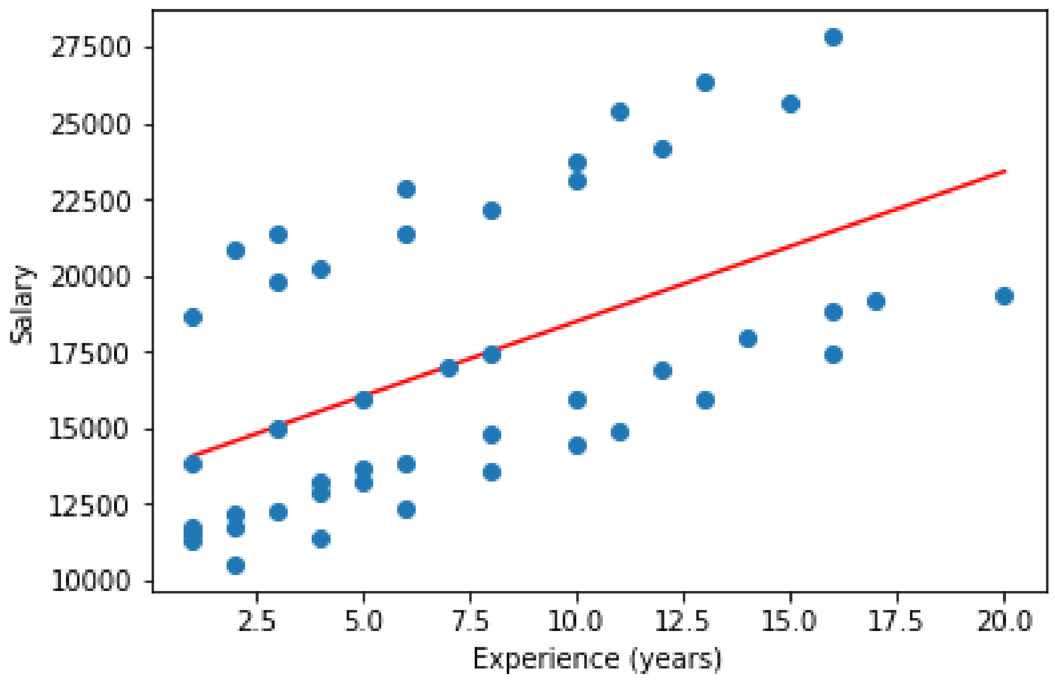
```
from scipy import stats
import numpy as np
y, x = salary.salary, salary.experience
beta, beta0, r_value, p_value, std_err = stats.linregress(x,y)
print("y = %f x + %f,  r: %f, r-squared: %f,\np-value: %f, std_err: %f"
      % (beta, beta0, r_value, r_value**2, p_value, std_err))

print("Regression line with the scatterplot")
yhat = beta * x  +  beta0 # regression line
```

(continues on next page)

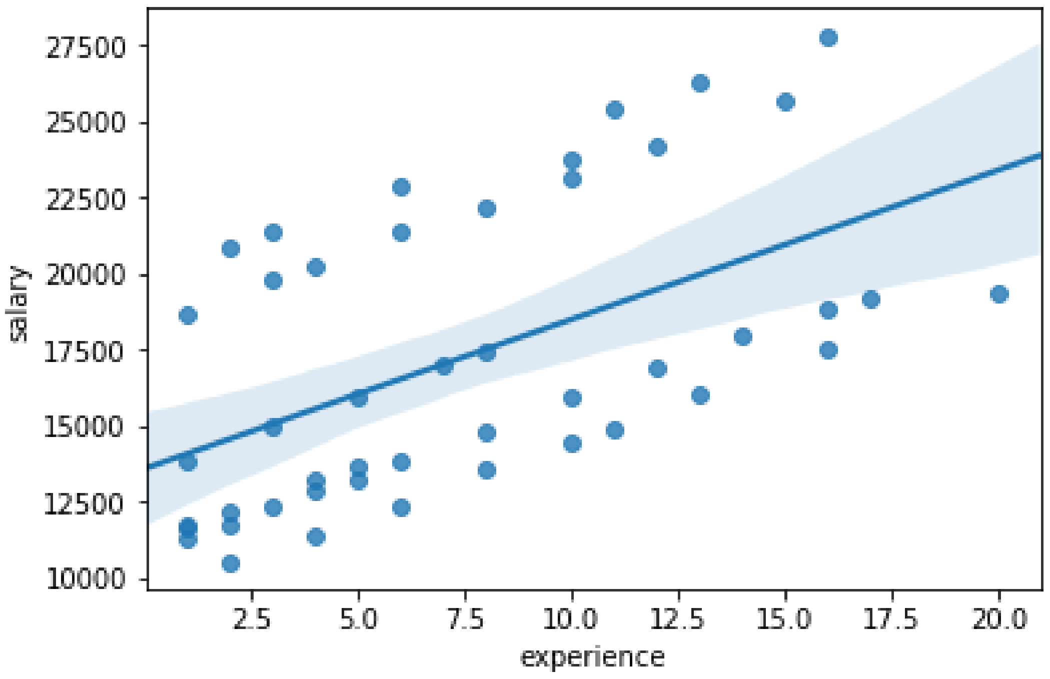
(continued from previous page)

```
plt.plot(x, yhat, 'r-', x, y,'o')
plt.xlabel('Experience (years)')
plt.ylabel('Salary')
plt.show()

print("Using seaborn")
import seaborn as sns
sns.regplot(x="experience", y="salary", data=salary);
```

```
y = 491.486913 x + 13584.043803,  r: 0.538886, r-squared: 0.290398,
p-value: 0.000112, std_err: 115.823381
Regression line with the scatterplot
```



```
Using seaborn
```



3. F-Test

3.1 Goodness of fit

The goodness of fit of a statistical model describes how well it fits a set of observations. Measures of goodness of fit typically summarize the discrepancy between observed values and the values expected under the model in question. We will consider the **explained variance** also known as the coefficient of determination, denoted R^2 pronounced **R-squared**.

The total sum of squares, SS_{tot} is the sum of the sum of squares explained by the regression, SS_{reg}, plus the sum of squares of residuals unexplained by the regression, SS_{res}, also called the SSE, i.e. such that

$$SS_{\text{tot}} = SS_{\text{reg}} + SS_{\text{res}}$$

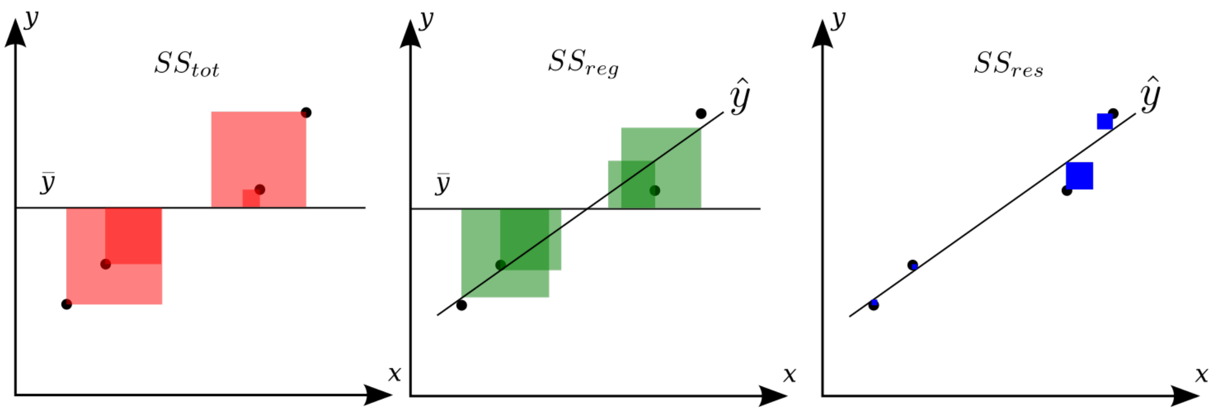


Fig. 4: title

The mean of y is

$$\bar{y} = \frac{1}{n}\sum_i y_i.$$

The total sum of squares is the total squared sum of deviations from the mean of y, i.e.

$$SS_\text{tot} = \sum_i (y_i - \bar{y})^2$$

The regression sum of squares, also called the explained sum of squares:

$$SS_\text{reg} = \sum_i (\hat{y}_i - \bar{y})^2,$$

where $\hat{y}_i = \beta x_i + \beta_0$ is the estimated value of salary $\hat{y}_i$ given a value of experience x_i.

The sum of squares of the residuals, also called the residual sum of squares (RSS) is:

$$SS_\text{res} = \sum_i (y_i - \hat{y}_i)^2.$$

R^2 is the explained sum of squares of errors. It is the variance explain by the regression divided by the total variance, i.e.

$$R^2 = \frac{\text{explained SS}}{\text{total SS}} = \frac{SS_\text{reg}}{SS_{tot}} = 1 - \frac{SS_{res}}{SS_{tot}}.$$

3.2 Test

Let $\hat{\sigma}^2 = SS_\text{res}/(n-2)$ be an estimator of the variance of ϵ. The 2 in the denominator stems from the 2 estimated parameters: intercept and coefficient.

- **Unexplained variance:** $\frac{SS_\text{res}}{\hat{\sigma}^2} \sim \chi^2_{n-2}$
- **Explained variance:** $\frac{SS_\text{reg}}{\hat{\sigma}^2} \sim \chi^2_1$. The single degree of freedom comes from the difference between $\frac{SS_\text{tot}}{\hat{\sigma}^2} (\sim \chi^2_{n-1})$ and $\frac{SS_\text{res}}{\hat{\sigma}^2} (\sim \chi^2_{n-2})$, i.e. $(n-1) - (n-2)$ degree of freedom.

The Fisher statistics of the ratio of two variances:

$$F = \frac{\text{Explained variance}}{\text{Unexplained variance}} = \frac{SS_\text{reg}/1}{SS_\text{res}/(n-2)} \sim F(1, n-2)$$

Using the F-distribution, compute the probability of observing a value greater than F under H_0, i.e.: $P(x > F|H_0)$, i.e. the survival function $(1 - \text{Cumulative Distribution Function})$ at x of the given F-distribution.

Multiple regression

Theory

Muliple Linear Regression is the most basic supervised learning algorithm.

Given: a set of training data $\{x_1, ..., x_N\}$ with corresponding targets $\{y_1, ..., y_N\}$.

In linear regression, we assume that the model that generates the data involves only a linear combination of the input variables, i.e.

$$y(x_i, \beta) = \beta^0 + \beta^1 x_i^1 + ... + \beta^P x_i^P,$$

or, simplified

$$y(x_i, \beta) = \beta_0 + \sum_{j=1}^{P-1} \beta_j x_i^j.$$

Extending each sample with an intercept, $x_i := [1, x_i] \in R^{P+1}$ allows us to use a more general notation based on linear algebra and write it as a simple dot product:

$$y(x_i, \beta) = x_i^T \beta,$$

where $\beta \in R^{P+1}$ is a vector of weights that define the $P+1$ parameters of the model. From now we have P regressors + the intercept.

Minimize the Mean Squared Error MSE loss:

$$MSE(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - y(x_i, \beta))^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i^T\beta)^2$$

Let $X = [x_0^T, ..., x_N^T]$ be a $N \times P+1$ matrix of N samples of P input features with one column of one and let be $y = [y_1, ..., y_N]$ be a vector of the N targets. Then, using linear algebra, the **mean squared error (MSE) loss can be rewritten**:

$$MSE(\beta) = \frac{1}{N}||y - X\beta||_2^2.$$

The β that minimises the MSE can be found by:

$$\nabla_\beta \left(\frac{1}{N}||y - X\beta||_2^2\right) = 0 \tag{4.20}$$

$$\frac{1}{N}\nabla_\beta (y - X\beta)^T(y - X\beta) = 0 \tag{4.21}$$

$$\frac{1}{N}\nabla_\beta (y^Ty - 2\beta^TX^Ty + \beta X^TX\beta) = 0 \tag{4.22}$$

$$-2X^Ty + 2X^TX\beta = 0 \tag{4.23}$$

$$X^TX\beta = X^Ty \tag{4.24}$$

$$\beta = (X^TX)^{-1}X^Ty, \tag{4.25}$$

where $(X^TX)^{-1}X^T$ is a pseudo inverse of X.

Fit with numpy

```
import numpy as np
from scipy import linalg
np.random.seed(seed=42)  # make the example reproducible
```

(continues on next page)

(continued from previous page)

```
# Dataset
N, P = 50, 4
X = np.random.normal(size= N * P).reshape((N, P))
## Our model needs an intercept so we add a column of 1s:
X[:, 0] = 1
print(X[:5, :])

betastar = np.array([10, 1., .5, 0.1])
e = np.random.normal(size=N)
y = np.dot(X, betastar) + e

# Estimate the parameters
Xpinv = linalg.pinv2(X)
betahat = np.dot(Xpinv, y)
print("Estimated beta:\n", betahat)
```

```
[[ 1.         -0.1382643   0.64768854  1.52302986]
 [ 1.         -0.23413696  1.57921282  0.76743473]
 [ 1.          0.54256004 -0.46341769 -0.46572975]
 [ 1.         -1.91328024 -1.72491783 -0.56228753]
 [ 1.          0.31424733 -0.90802408 -1.4123037 ]]
Estimated beta:
 [10.14742501  0.57938106  0.51654653  0.17862194]
```

4.1.7 Linear model with statsmodels

Sources: http://statsmodels.sourceforge.net/devel/examples/

Multiple regression

Interface with Numpy

```
import statsmodels.api as sm

## Fit and summary:
model = sm.OLS(y, X).fit()
print(model.summary())

# prediction of new values
ypred = model.predict(X)

# residuals + prediction == true values
assert np.all(ypred + model.resid == y)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.363
Model:                            OLS   Adj. R-squared:                  0.322
Method:                 Least Squares   F-statistic:                     8.748
Date:                Thu, 16 May 2019   Prob (F-statistic):           0.000106
```

(continues on next page)

(continued from previous page)

```
Time:                        20:15:04   Log-Likelihood:                -71.271
No. Observations:                  50   AIC:                             150.5
Df Residuals:                      46   BIC:                             158.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         10.1474      0.150     67.520      0.000       9.845      10.450
x1             0.5794      0.160      3.623      0.001       0.258       0.901
x2             0.5165      0.151      3.425      0.001       0.213       0.820
x3             0.1786      0.144      1.240      0.221      -0.111       0.469
==============================================================================
Omnibus:                        2.493   Durbin-Watson:                   2.369
Prob(Omnibus):                  0.288   Jarque-Bera (JB):                1.544
Skew:                           0.330   Prob(JB):                        0.462
Kurtosis:                       3.554   Cond. No.                         1.27
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
```

Interface with Pandas

Use R language syntax for data.frame. For an additive model: $y_i = \beta^0 + x_i^1\beta^1 + x_i^2\beta^2 + \epsilon_i \equiv$ y ~ x1 + x2.

```python
import statsmodels.formula.api as smfrmla

df = pd.DataFrame(np.column_stack([X, y]), columns=['inter', 'x1','x2', 'x3', 'y'])
print(df.columns, df.shape)
# Build a model excluding the intercept, it is implicit
model = smfrmla.ols("y~x1 + x2 + x3", df).fit()
print(model.summary())
```

```
Index(['inter', 'x1', 'x2', 'x3', 'y'], dtype='object') (50, 5)
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.363
Model:                            OLS   Adj. R-squared:                  0.322
Method:                 Least Squares   F-statistic:                     8.748
Date:                Thu, 16 May 2019   Prob (F-statistic):           0.000106
Time:                        20:15:04   Log-Likelihood:                -71.271
No. Observations:                  50   AIC:                             150.5
Df Residuals:                      46   BIC:                             158.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     10.1474      0.150     67.520      0.000       9.845      10.450
x1             0.5794      0.160      3.623      0.001       0.258       0.901
```

(continues on next page)

(continued from previous page)

```
x2             0.5165      0.151      3.425      0.001       0.213       0.820
x3             0.1786      0.144      1.240      0.221      -0.111       0.469
==============================================================================
Omnibus:                        2.493   Durbin-Watson:                   2.369
Prob(Omnibus):                  0.288   Jarque-Bera (JB):                1.544
Skew:                           0.330   Prob(JB):                        0.462
Kurtosis:                       3.554   Cond. No.                         1.27
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
```

Multiple regression with categorical independent variables or factors: Analysis of covariance (ANCOVA)

Analysis of covariance (ANCOVA) is a linear model that blends ANOVA and linear regression. ANCOVA evaluates whether population means of a dependent variable (DV) are equal across levels of a categorical independent variable (IV) often called a treatment, while statistically controlling for the effects of other quantitative or continuous variables that are not of primary interest, known as covariates (CV).

```python
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

try:
    salary = pd.read_csv("../datasets/salary_table.csv")
except:
    url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
    salary = pd.read_csv(url)
```

One-way AN(C)OVA

- ANOVA: one categorical independent variable, i.e. one factor.
- ANCOVA: ANOVA with some covariates.

```python
import statsmodels.formula.api as smfrmla

oneway = smfrmla.ols('salary ~ management + experience', salary).fit()
print(oneway.summary())
aov = sm.stats.anova_lm(oneway, typ=2) # Type 2 ANOVA DataFrame
print(aov)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 salary   R-squared:                       0.865
Model:                            OLS   Adj. R-squared:                  0.859
Method:                 Least Squares   F-statistic:                     138.2
Date:                Thu, 16 May 2019   Prob (F-statistic):           1.90e-19
```

(continues on next page)

(continued from previous page)

```
Time:                        20:15:04   Log-Likelihood:                -407.76
No. Observations:                  46   AIC:                             821.5
Df Residuals:                      43   BIC:                             827.0
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept       1.021e+04    525.999     19.411      0.000    9149.578    1.13e+04
management[T.Y]  7145.0151    527.320     13.550      0.000    6081.572    8208.458
experience       527.1081     51.106     10.314      0.000     424.042     630.174
==============================================================================
Omnibus:                       11.437   Durbin-Watson:                   2.193
Prob(Omnibus):                  0.003   Jarque-Bera (JB):               11.260
Skew:                          -1.131   Prob(JB):                      0.00359
Kurtosis:                       3.872   Cond. No.                         22.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
                  sum_sq    df           F        PR(>F)
management  5.755739e+08   1.0  183.593466  4.054116e-17
experience  3.334992e+08   1.0  106.377768  3.349662e-13
Residual    1.348070e+08  43.0         NaN           NaN
```

Two-way AN(C)OVA

Ancova with two categorical independent variables, i.e. two factors.

```
import statsmodels.formula.api as smfrmla

twoway = smfrmla.ols('salary ~ education + management + experience', salary).fit()
print(twoway.summary())
aov = sm.stats.anova_lm(twoway, typ=2) # Type 2 ANOVA DataFrame
print(aov)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 salary   R-squared:                       0.957
Model:                            OLS   Adj. R-squared:                  0.953
Method:                 Least Squares   F-statistic:                     226.8
Date:                Thu, 16 May 2019   Prob (F-statistic):           2.23e-27
Time:                        20:15:04   Log-Likelihood:                -381.63
No. Observations:                  46   AIC:                             773.3
Df Residuals:                      41   BIC:                             782.4
Df Model:                           4
Covariance Type:            nonrobust
======================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
Intercept           8035.5976    386.689     20.781      0.000    7254.663    8816.532
education[T.Master]  3144.0352    361.968      8.686      0.000    2413.025    3875.045
```

(continues on next page)

(continued from previous page)

```
education[T.Ph.D]    2996.2103    411.753      7.277      0.000     2164.659    3827.762
management[T.Y]      6883.5310    313.919     21.928      0.000     6249.559    7517.503
experience            546.1840     30.519     17.896      0.000      484.549     607.819
==============================================================================
Omnibus:                        2.293   Durbin-Watson:                   2.237
Prob(Omnibus):                  0.318   Jarque-Bera (JB):                1.362
Skew:                          -0.077   Prob(JB):                        0.506
Kurtosis:                       2.171   Cond. No.                         33.5
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
                  sum_sq    df           F        PR(>F)
education   9.152624e+07   2.0   43.351589  7.672450e-11
management  5.075724e+08   1.0  480.825394  2.901444e-24
experience  3.380979e+08   1.0  320.281524  5.546313e-21
Residual    4.328072e+07  41.0         NaN           NaN
```

Comparing two nested models

oneway is nested within twoway. Comparing two nested models tells us if the additional predictors (i.e. education) of the full model significantly decrease the residuals. Such comparison can be done using an F-test on residuals:

```
print(twoway.compare_f_test(oneway))  # return F, pval, df
```

```
(43.35158945918107, 7.672449570495418e-11, 2.0)
```

Factor coding

See http://statsmodels.sourceforge.net/devel/contrasts.html

By default Pandas use "dummy coding". Explore:

```
print(twoway.model.data.param_names)
print(twoway.model.data.exog[:10, :])
```

```
['Intercept', 'education[T.Master]', 'education[T.Ph.D]', 'management[T.Y]', 'experience']
[[1. 0. 0. 1. 1.]
 [1. 0. 1. 0. 1.]
 [1. 0. 1. 1. 1.]
 [1. 1. 0. 0. 1.]
 [1. 0. 1. 0. 1.]
 [1. 1. 0. 1. 2.]
 [1. 1. 0. 0. 2.]
 [1. 0. 0. 0. 2.]
 [1. 0. 1. 0. 2.]
 [1. 1. 0. 0. 3.]]
```

Contrasts and post-hoc tests

```
# t-test of the specific contribution of experience:
ttest_exp = twoway.t_test([0, 0, 0, 0, 1])
ttest_exp.pvalue, ttest_exp.tvalue
print(ttest_exp)

# Alternatively, you can specify the hypothesis tests using a string
twoway.t_test('experience')

# Post-hoc is salary of Master different salary of Ph.D?
# ie. t-test salary of Master = salary of Ph.D.
print(twoway.t_test('education[T.Master] = education[T.Ph.D]'))
```

```
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0           546.1840     30.519     17.896      0.000     484.549     607.819
==============================================================================
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0           147.8249    387.659      0.381      0.705    -635.069     930.719
==============================================================================
```

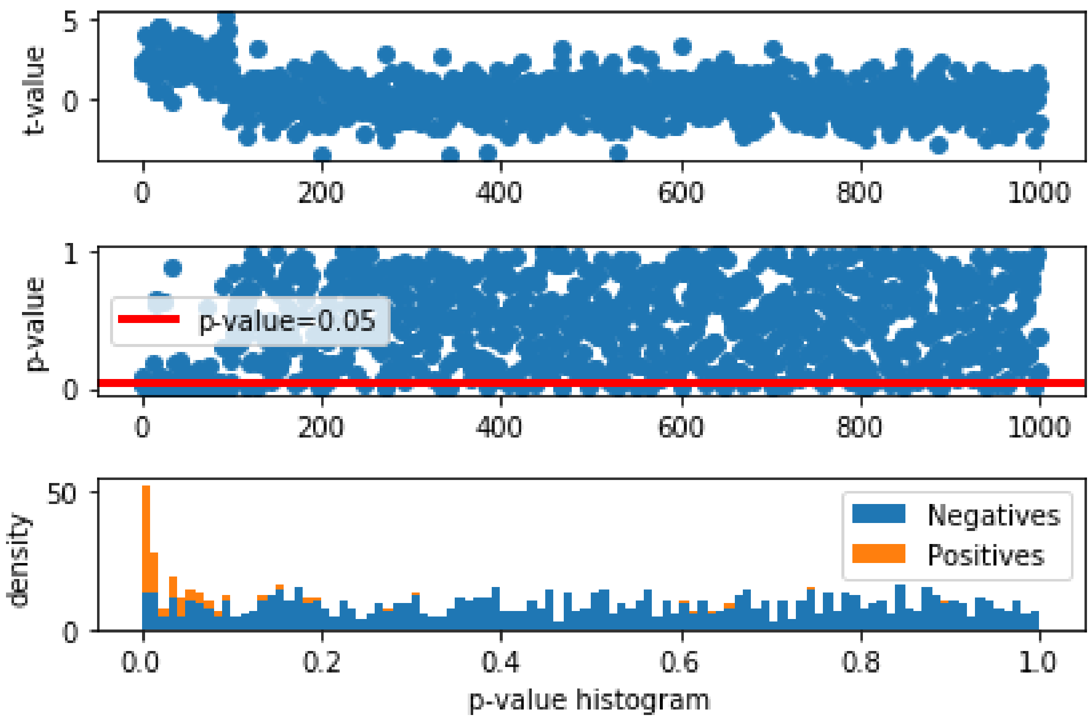
4.1.8 Multiple comparisons

```
import numpy as np
np.random.seed(seed=42)  # make example reproducible

# Dataset
n_samples, n_features = 100, 1000
n_info = int(n_features/10)  # number of features with information
n1, n2 = int(n_samples/2), n_samples - int(n_samples/2)
snr = .5
Y = np.random.randn(n_samples, n_features)
grp = np.array(["g1"] * n1 + ["g2"] * n2)

# Add some group effect for Pinfo features
Y[grp=="g1", :n_info] += snr

#
import scipy.stats as stats
import matplotlib.pyplot as plt
tvals, pvals = np.full(n_features, np.NAN), np.full(n_features, np.NAN)
for j in range(n_features):
    tvals[j], pvals[j] = stats.ttest_ind(Y[grp=="g1", j], Y[grp=="g2", j],
                                         equal_var=True)

fig, axis = plt.subplots(3, 1)#, sharex='col')

axis[0].plot(range(n_features), tvals, 'o')
```

(continues on next page)

(continued from previous page)

```
axis[0].set_ylabel("t-value")

axis[1].plot(range(n_features), pvals, 'o')
axis[1].axhline(y=0.05, color='red', linewidth=3, label="p-value=0.05")
#axis[1].axhline(y=0.05, label="toto", color='red')
axis[1].set_ylabel("p-value")
axis[1].legend()

axis[2].hist([pvals[n_info:], pvals[:n_info]],
    stacked=True, bins=100, label=["Negatives", "Positives"])
axis[2].set_xlabel("p-value histogram")
axis[2].set_ylabel("density")
axis[2].legend()

plt.tight_layout()
```



Note that under the null hypothesis the distribution of the *p*-values is uniform.

Statistical measures:

- **True Positive (TP)** equivalent to a hit. The test correctly concludes the presence of an effect.
- True Negative (TN). The test correctly concludes the absence of an effect.
- **False Positive (FP)** equivalent to a false alarm, **Type I error**. The test improperly concludes the presence of an effect. Thresholding at p-value < 0.05 leads to 47 FP.
- False Negative (FN) equivalent to a miss, Type II error. The test improperly concludes the absence of an effect.

```
P, N = n_info, n_features - n_info  # Positives, Negatives
TP = np.sum(pvals[:n_info ] < 0.05)  # True Positives
```

(continues on next page)

(continued from previous page)

```
FP = np.sum(pvals[n_info: ] < 0.05)  # False Positives
print("No correction, FP: %i (expected: %.2f), TP: %i" % (FP, N * 0.05, TP))
```

```
No correction, FP: 47 (expected: 45.00), TP: 71
```

Bonferroni correction for multiple comparisons

The Bonferroni correction is based on the idea that if an experimenter is testing P hypotheses, then one way of maintaining the familywise error rate (FWER) is to test each individual hypothesis at a statistical significance level of $1/P$ times the desired maximum overall level.

So, if the desired significance level for the whole family of tests is α (usually 0.05), then the Bonferroni correction would test each individual hypothesis at a significance level of α/P. For example, if a trial is testing $P = 8$ hypotheses with a desired $\alpha = 0.05$, then the Bonferroni correction would test each individual hypothesis at $\alpha = 0.05/8 = 0.00625$.

```
import statsmodels.sandbox.stats.multicomp as multicomp
_, pvals_fwer, _, _  = multicomp.multipletests(pvals, alpha=0.05,
                                               method='bonferroni')
TP = np.sum(pvals_fwer[:n_info ] < 0.05)  # True Positives
FP = np.sum(pvals_fwer[n_info: ] < 0.05)  # False Positives
print("FWER correction, FP: %i, TP: %i" % (FP, TP))
```

```
FWER correction, FP: 0, TP: 6
```

The False discovery rate (FDR) correction for multiple comparisons

FDR-controlling procedures are designed to control the expected proportion of rejected null hypotheses that were incorrect rejections ("false discoveries"). FDR-controlling procedures provide less stringent control of Type I errors compared to the familywise error rate (FWER) controlling procedures (such as the Bonferroni correction), which control the probability of at least one Type I error. Thus, FDR-controlling procedures have greater power, at the cost of increased rates of Type I errors.

```
import statsmodels.sandbox.stats.multicomp as multicomp
_, pvals_fdr, _, _  = multicomp.multipletests(pvals, alpha=0.05,
                                               method='fdr_bh')
TP = np.sum(pvals_fdr[:n_info ] < 0.05)  # True Positives
FP = np.sum(pvals_fdr[n_info: ] < 0.05)  # False Positives

print("FDR correction, FP: %i, TP: %i" % (FP, TP))
```

```
FDR correction, FP: 3, TP: 20
```

4.1.9 Exercises

Simple linear regression and correlation (application)

Load the dataset: birthwt Risk Factors Associated with Low Infant Birth Weight at `ftp://ftp.cea.fr/pub/unati/people/educhesnay/pystatml/datasets/birthwt.csv`

1. Test the association of mother's age and birth weight using the correlation test and linear regeression.
2. Test the association of mother's weight and birth weight using the correlation test and linear regeression.
3. Produce two scatter plot of: (i) age by birth weight; (ii) mother's weight by birth weight.

Conclusion ?

Simple linear regression (maths)

Considering the salary and the experience of the salary table.

https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv

Compute:

- Estimate the model paramters β, β_0 using scipy `stats.linregress(x,y)`
- Compute the predicted values $\hat{y}$

Compute:

- $\bar{y}$: `y_mu`
- SS_{tot}: `ss_tot`
- SS_{reg}: `ss_reg`
- SS_{res}: `ss_res`
- Check partition of variance formula based on sum of squares by using `assert np.allclose(val1, val2, atol=1e-05)`
- Compute R^2 and compare it with the `r_value` above
- Compute the F score
- Compute the p-value:
- Plot the $F(1, n)$ distribution for 100 f values within $[10, 25]$. Draw $P(F(1, n) > F)$, i.e. color the surface defined by the x values larger than F below the $F(1, n)$.
- $P(F(1, n) > F)$ is the p-value, compute it.

Multiple regression

Considering the simulated data used below:

1. What are the dimensions of $\text{pinv}(X)$?
2. Compute the MSE between the predicted values and the true values.

```
import numpy as np
from scipy import linalg
np.random.seed(seed=42)  # make the example reproducible

# Dataset
N, P = 50, 4
X = np.random.normal(size= N * P).reshape((N, P))
## Our model needs an intercept so we add a column of 1s:
X[:, 0] = 1
print(X[:5, :])

betastar = np.array([10, 1., .5, 0.1])
e = np.random.normal(size=N)
y = np.dot(X, betastar) + e

# Estimate the parameters
Xpinv = linalg.pinv2(X)
betahat = np.dot(Xpinv, y)
print("Estimated beta:\n", betahat)
```

```
[[ 1.         -0.1382643   0.64768854  1.52302986]
 [ 1.         -0.23413696  1.57921282  0.76743473]
 [ 1.          0.54256004 -0.46341769 -0.46572975]
 [ 1.         -1.91328024 -1.72491783 -0.56228753]
 [ 1.          0.31424733 -0.90802408 -1.4123037 ]]
Estimated beta:
 [10.14742501  0.57938106  0.51654653  0.17862194]
```

Two sample t-test (maths)

Given the following two sample, test whether their means are equals.

```
height = np.array([ 1.83,  1.83,  1.73,  1.82,  1.83,
                    1.73,1.99,  1.85,  1.68,  1.87,
                    1.66,  1.71,  1.73,  1.64,  1.70,
                    1.60,  1.79,  1.73,  1.62,  1.77])
grp = np.array(["M"] * 10 + ["F"] * 10)
```

- Compute the means/std-dev per groups.
- Compute the t-value (standard two sample t-test with equal variances).
- Compute the p-value.
- The p-value is one-sided: a two-sided test would test `P(T > tval)` and `P(T < -tval)`. What would the two sided p-value be?
- Compare the two-sided p-value with the one obtained by `stats.ttest_ind` using `assert np.allclose(arr1, arr2)`.

Two sample t-test (application)

Risk Factors Associated with Low Infant Birth Weight:

`https://raw.github.com/neurospin/pystatsml/master/datasets/birthwt.csv`

1. Explore the data
2. Recode smoke factor
3. Compute the means/std-dev per groups.
4. Plot birth weight by smoking (box plot, violin plot or histogram)
5. Test the effect of smoking on birth weight

Two sample t-test and random permutations

Generate 100 samples following the model:

$$y = g + \varepsilon$$

Where the noise $\varepsilon \sim N(1, 1)$ and $g \in \{0, 1\}$ is a group indicator variable with 50 ones and 50 zeros.

- Write a function `tstat(y, g)` that compute the two samples t-test of y splited in two groups defined by g.
- Sample the t-statistic distribution under the null hypothesis using random permutations.
- Assess the p-value.

Univariate associations (developpement)

Write a function `univar_stat(df, target, variables)` that computes the parametric statistics and p-values between the `target` variable (provided as as string) and all `variables` (provided as a list of string) of the pandas DataFrame `df`. The target is a quantitative variable but variables may be quantitative or qualitative. The function returns a DataFrame with four columns: `variable`, `test`, `value`, `p_value`.

Apply it to the salary dataset available at https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv, with target being `S`: salaries for IT staff in a corporation.

Multiple comparisons

This exercise has 2 goals: apply you knowledge of statistics using vectorized numpy operations. Given the dataset provided for multiple comparisons, compute the two-sample t-test (assuming equal variance) for each (column) feature of the `Y` array given the two groups defined by `grp` variable. You should return two vectors of size `n_features`: one for the t-values and one for the p-values.

ANOVA

Perform an ANOVA dataset described bellow

- Compute between and within variances
- Compute F-value: `fval`
- Compare the p-value with the one obtained by `stats.f_oneway` using `assert np.allclose(arr1, arr2)`

```
# dataset
mu_k = np.array([1, 2, 3])    # means of 3 samples
sd_k = np.array([1, 1, 1])    # sd of 3 samples
n_k = np.array([10, 20, 30])  # sizes of 3 samples
grp = [0, 1, 2]               # group labels
n = np.sum(n_k)
label = np.hstack([[k] * n_k[k] for k in [0, 1, 2]])

y = np.zeros(n)
for k in grp:
    y[label == k] = np.random.normal(mu_k[k], sd_k[k], n_k[k])

# Compute with scipy
fval, pval = stats.f_oneway(y[label == 0], y[label == 1], y[label == 2])
```

Note: Click *here* to download the full example code

4.2 Lab 1: Brain volumes study

The study provides the brain volumes of grey matter (gm), white matter (wm) and cerebrospinal fluid) (csf) of 808 anatomical MRI scans. Manipulate data ————

Set the working directory within a directory called "brainvol"

Create 2 subdirectories: *data* that will contain downloaded data and *reports* for results of the analysis.

```
import os
import os.path
import pandas as pd
import tempfile
import urllib.request

WD = os.path.join(tempfile.gettempdir(), "brainvol")
os.makedirs(WD, exist_ok=True)
#os.chdir(WD)

# use cookiecutter file organization
# https://drivendata.github.io/cookiecutter-data-science/
os.makedirs(os.path.join(WD, "data"), exist_ok=True)
#os.makedirs("reports", exist_ok=True)
```

Fetch data

- Demographic data *demo.csv* (columns: *participant_id*, *site*, *group*, *age*, *sex*) and tissue volume data: *group* is Control or Patient. *site* is the recruiting site.
- Gray matter volume *gm.csv* (columns: *participant_id*, *session*, *gm_vol*)
- White matter volume *wm.csv* (columns: *participant_id*, *session*, *wm_vol*)
- Cerebrospinal Fluid *csf.csv* (columns: *participant_id*, *session*, *csf_vol*)

```
base_url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/brain_volumes/%s'
data = dict()
for file in ["demo.csv", "gm.csv", "wm.csv", "csf.csv"]:
    urllib.request.urlretrieve(base_url % file, os.path.join(WD, "data", file))

demo = pd.read_csv(os.path.join(WD, "data", "demo.csv"))
gm = pd.read_csv(os.path.join(WD, "data", "gm.csv"))
wm = pd.read_csv(os.path.join(WD, "data", "wm.csv"))
csf = pd.read_csv(os.path.join(WD, "data", "csf.csv"))

print("tables can be merge using shared columns")
print(gm.head())
```

Out:

```
tables can be merge using shared columns
  participant_id session    gm_vol
0    sub-S1-0002  ses-01  0.672506
1    sub-S1-0002  ses-02  0.678772
2    sub-S1-0002  ses-03  0.665592
3    sub-S1-0004  ses-01  0.890714
4    sub-S1-0004  ses-02  0.881127
```

Merge tables according to *participant_id*

```
brain_vol = pd.merge(pd.merge(pd.merge(demo, gm), wm), csf)
assert brain_vol.shape == (808, 9)
```

Drop rows with missing values

```
brain_vol = brain_vol.dropna()
assert brain_vol.shape == (766, 9)
```

Compute Total Intra-cranial volume $tiv\_vol = gm\_vol + csf\_vol + wm\_vol$.

```
brain_vol["tiv_vol"] = brain_vol["gm_vol"] + brain_vol["wm_vol"] + brain_vol["csf_vol"]
```

Compute tissue fractions $gm\_f = gm\_vol / tiv\_vol$, $wm\_f = wm\_vol / tiv\_vol$.

```
brain_vol["gm_f"] = brain_vol["gm_vol"] / brain_vol["tiv_vol"]
brain_vol["wm_f"] = brain_vol["wm_vol"] / brain_vol["tiv_vol"]
```

Save in a excel file *brain_vol.xlsx*

```
brain_vol.to_excel(os.path.join(WD, "data", "brain_vol.xlsx"),
                   sheet_name='data', index=False)
```

4.2.1 Descriptive Statistics

Load excel file *brain_vol.xlsx*

```
import os
import pandas as pd
import seaborn as sns
```

(continues on next page)

(continued from previous page)

```
import statsmodels.formula.api as smfrmla
import statsmodels.api as sm

brain_vol = pd.read_excel(os.path.join(WD, "data", "brain_vol.xlsx"),
                          sheet_name='data')
# Round float at 2 decimals when printing
pd.options.display.float_format = '{:,.2f}'.format
```

Descriptive statistics Most of participants have several MRI sessions (column *session*) Select on rows from session one "ses-01"

```
brain_vol1 = brain_vol[brain_vol.session == "ses-01"]
# Check that there are no duplicates
assert len(brain_vol1.participant_id.unique()) == len(brain_vol1.participant_id)
```

Global descriptives statistics of numerical variables

```
desc_glob_num = brain_vol1.describe()
print(desc_glob_num)
```

Out:

```
age  gm_vol  wm_vol  csf_vol  tiv_vol   gm_f   wm_f
count 244.00  244.00  244.00   244.00   244.00 244.00 244.00
mean   34.54    0.71    0.44     0.31     1.46   0.49   0.30
std    12.09    0.08    0.07     0.08     0.17   0.04   0.03
min    18.00    0.48    0.05     0.12     0.83   0.37   0.06
25%    25.00    0.66    0.40     0.25     1.34   0.46   0.28
50%    31.00    0.70    0.43     0.30     1.45   0.49   0.30
75%    44.00    0.77    0.48     0.37     1.57   0.52   0.31
max    61.00    1.03    0.62     0.63     2.06   0.60   0.36
```

Global Descriptive statistics of categorical variable

```
desc_glob_cat = brain_vol1[["site", "group", "sex"]].describe(include='all')
print(desc_glob_cat)

print("Get count by level")
desc_glob_cat = pd.DataFrame({col: brain_vol1[col].value_counts().to_dict()
                              for col in ["site", "group", "sex"]})
print(desc_glob_cat)
```

Out:

```
site    group  sex
count   244       244  244
unique    7         2    2
top      S7  Patient    M
freq     65       157  155
Get count by level
         site  group    sex
Control   nan  87.00    nan
F         nan    nan  89.00
M         nan    nan 155.00
Patient   nan 157.00    nan
```

(continues on next page)

(continued from previous page)

```
S1      13.00    nan    nan
S3      29.00    nan    nan
S4      15.00    nan    nan
S5      62.00    nan    nan
S6       1.00    nan    nan
S7      65.00    nan    nan
S8      59.00    nan    nan
```

Remove the single participant from site 6

```
brain_vol = brain_vol[brain_vol.site != "S6"]
brain_vol1 = brain_vol[brain_vol.session == "ses-01"]
desc_glob_cat = pd.DataFrame({col: brain_vol1[col].value_counts().to_dict()
                              for col in ["site", "group", "sex"]})
print(desc_glob_cat)
```

Out:

```
site  group    sex
Control    nan  86.00    nan
F          nan    nan  88.00
M          nan    nan 155.00
Patient    nan 157.00    nan
S1      13.00    nan    nan
S3      29.00    nan    nan
S4      15.00    nan    nan
S5      62.00    nan    nan
S7      65.00    nan    nan
S8      59.00    nan    nan
```

Descriptives statistics of numerical variables per clinical status

```
desc_group_num = brain_vol1[["group", 'gm_vol']].groupby("group").describe()
print(desc_group_num)
```

Out:

```
gm_vol
         count mean  std  min  25%  50%  75%  max
group
Control  86.00 0.72 0.09 0.48 0.66 0.71 0.78 1.03
Patient 157.00 0.70 0.08 0.53 0.65 0.70 0.76 0.90
```

4.2.2 Statistics

Objectives:

1. Site effect of gray matter atrophy
2. Test the association between the age and gray matter atrophy in the control and patient population independently.
3. Test for differences of atrophy between the patients and the controls
4. Test for interaction between age and clinical status, ie: is the brain atrophy process in patient population faster than in the control population.

5. The effect of the medication in the patient population.

```
import statsmodels.api as sm
import statsmodels.formula.api as smfrmla
import scipy.stats
import seaborn as sns
```

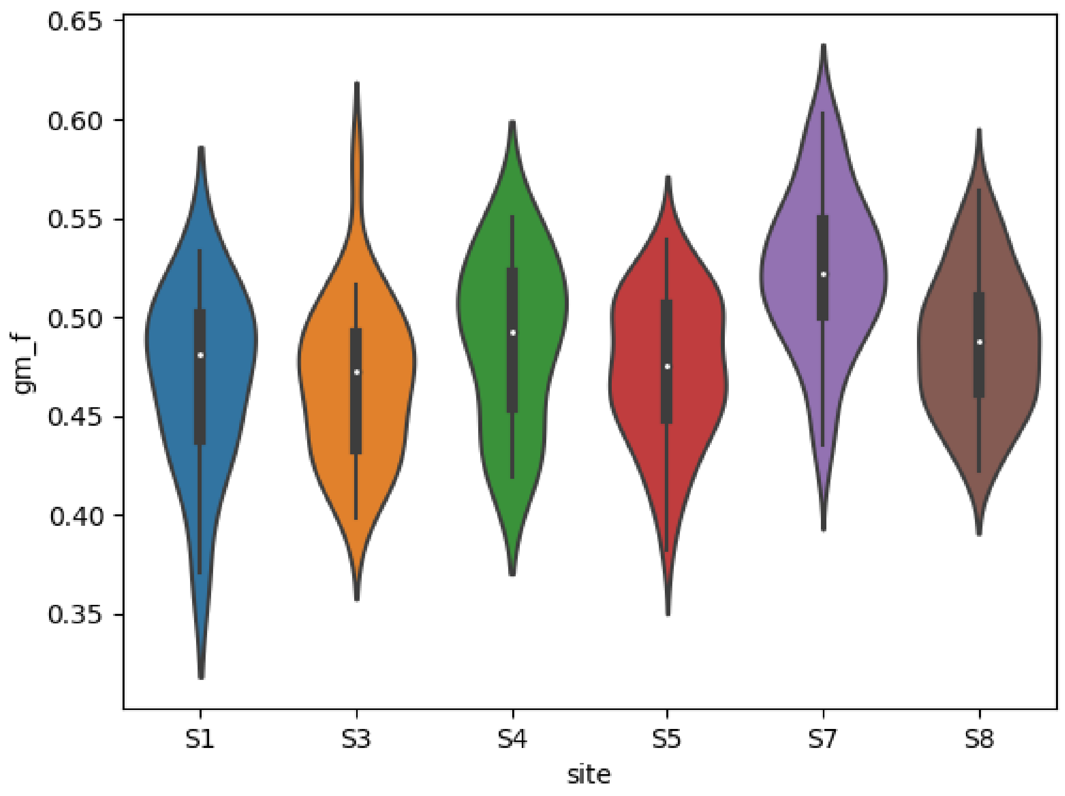
1 Site effect on Grey Matter atrophy

The model is Oneway Anova gm_f ~ site The ANOVA test has important assumptions that must be satisfied in order for the associated p-value to be valid.

- The samples are independent.
- Each sample is from a normally distributed population.
- The population standard deviations of the groups are all equal. This property is known as homoscedasticity.

Plot

```
sns.violinplot("site", "gm_f", data=brain_vol1)
```



Stats with scipy

```
fstat, pval = scipy.stats.f_oneway(*[brain_vol1.gm_f[brain_vol1.site == s]
                                   for s in brain_vol1.site.unique()])
print("Oneway Anova gm_f ~ site F=%.2f, p-value=%E" % (fstat, pval))
```

Out:

```
Oneway Anova gm_f ~ site F=14.82, p-value=1.188136E-12
```

Stats with statsmodels

```
anova = smfrmla.ols("gm_f ~ site", data=brain_vol1).fit()
# print(anova.summary())
print("Site explains %.2f%% of the grey matter fraction variance" %
      (anova.rsquared * 100))

print(sm.stats.anova_lm(anova, typ=2))
```

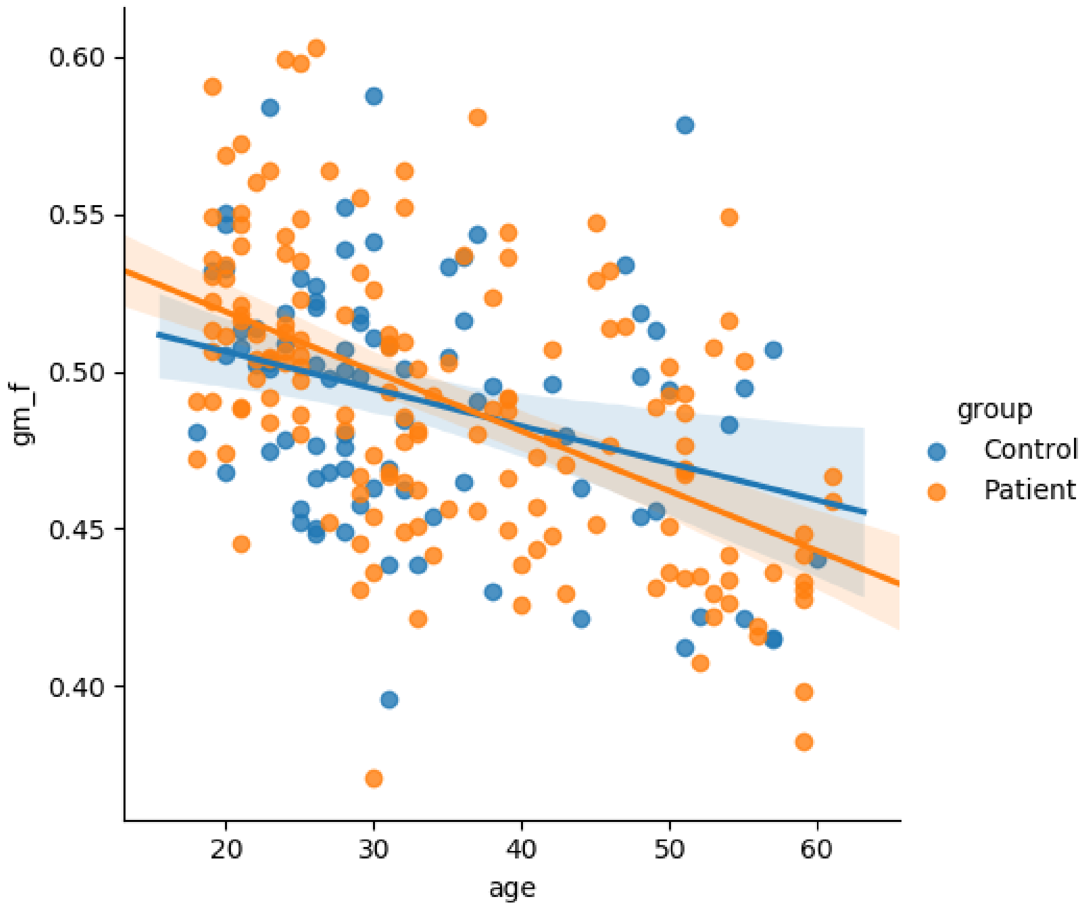
Out:

```
Site explains 23.82% of the grey matter fraction variance
          sum_sq     df     F  PR(>F)
site        0.11   5.00 14.82    0.00
Residual    0.35 237.00   nan     nan
```

2. Test the association between the age and gray matter atrophy in the control and patient population independently.

Plot

```
sns.lmplot("age", "gm_f", hue="group", data=brain_vol1)

brain_vol1_ctl = brain_vol1[brain_vol1.group == "Control"]
brain_vol1_pat = brain_vol1[brain_vol1.group == "Patient"]
```



Stats with scipy

```
print("--- In control population ---")
beta, beta0, r_value, p_value, std_err = \
    scipy.stats.linregress(x=brain_vol1_ctl.age, y=brain_vol1_ctl.gm_f)

print("gm_f = %f * age + %f" % (beta, beta0))
print("Corr: %f, r-squared: %f, p-value: %f, std_err: %f"\
      % (r_value, r_value**2, p_value, std_err))

print("--- In patient population ---")
beta, beta0, r_value, p_value, std_err = \
    scipy.stats.linregress(x=brain_vol1_pat.age, y=brain_vol1_pat.gm_f)

print("gm_f = %f * age + %f" % (beta, beta0))
print("Corr: %f, r-squared: %f, p-value: %f, std_err: %f"\
      % (r_value, r_value**2, p_value, std_err))

print("Decrease seems faster in patient than in control population")
```

Out:

```
--- In control population ---
gm_f = -0.001181 * age + 0.529829
Corr: -0.325122, r-squared: 0.105704, p-value: 0.002255, std_err: 0.000375
--- In patient population ---
```

(continues on next page)

(continued from previous page)

```
gm_f = -0.001899 * age + 0.556886
Corr: -0.528765, r-squared: 0.279592, p-value: 0.000000, std_err: 0.000245
Decrease seems faster in patient than in control population
```

Stats with statsmodels

```
print("--- In control population ---")
lr = smfrmla.ols("gm_f ~ age", data=brain_vol1_ctl).fit()
print(lr.summary())
print("Age explains %.2f%% of the grey matter fraction variance" %
      (lr.rsquared * 100))

print("--- In patient population ---")
lr = smfrmla.ols("gm_f ~ age", data=brain_vol1_pat).fit()
print(lr.summary())
print("Age explains %.2f%% of the grey matter fraction variance" %
      (lr.rsquared * 100))
```

Out:

```
--- In control population ---
                            OLS Regression Results
==============================================================================
Dep. Variable:                   gm_f   R-squared:                       0.106
Model:                            OLS   Adj. R-squared:                  0.095
Method:                 Least Squares   F-statistic:                     9.929
Date:                jeu., 16 mai 2019   Prob (F-statistic):            0.00226
Time:                        20:18:24   Log-Likelihood:                 159.34
No. Observations:                  86   AIC:                            -314.7
Df Residuals:                      84   BIC:                            -309.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5298      0.013     40.350      0.000       0.504       0.556
age           -0.0012      0.000     -3.151      0.002      -0.002      -0.000
==============================================================================
Omnibus:                        0.946   Durbin-Watson:                   1.628
Prob(Omnibus):                  0.623   Jarque-Bera (JB):                0.782
Skew:                           0.233   Prob(JB):                        0.676
Kurtosis:                       2.962   Cond. No.                         111.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
Age explains 10.57% of the grey matter fraction variance
--- In patient population ---
                            OLS Regression Results
==============================================================================
Dep. Variable:                   gm_f   R-squared:                       0.280
Model:                            OLS   Adj. R-squared:                  0.275
Method:                 Least Squares   F-statistic:                     60.16
Date:                jeu., 16 mai 2019   Prob (F-statistic):           1.09e-12
Time:                        20:18:24   Log-Likelihood:                 289.38
```

(continues on next page)

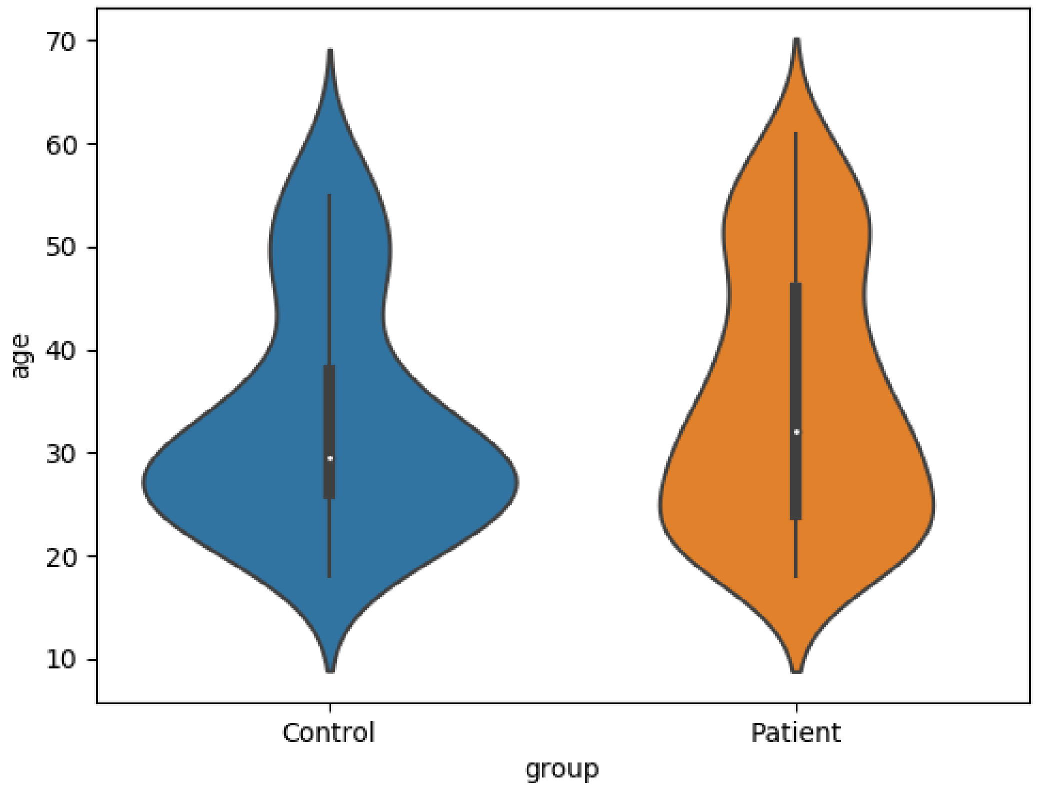
(continued from previous page)

```
No. Observations:                 157   AIC:                            -574.8
Df Residuals:                     155   BIC:                            -568.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5569      0.009     60.817      0.000       0.539       0.575
age           -0.0019      0.000     -7.756      0.000      -0.002      -0.001
==============================================================================
Omnibus:                        2.310   Durbin-Watson:                   1.325
Prob(Omnibus):                  0.315   Jarque-Bera (JB):                1.854
Skew:                           0.230   Prob(JB):                        0.396
Kurtosis:                       3.268   Cond. No.                         111.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
Age explains 27.96% of the grey matter fraction variance
```

Before testing for differences of atrophy between the patients ans the controls **Preliminary tests for age x group effect** (patients would be older or younger than Controls)

Plot

```
sns.violinplot("group", "age", data=brain_vol1)
```



Stats with scipy

```
print(scipy.stats.ttest_ind(brain_vol1_ctl.age, brain_vol1_pat.age))
```

Out:

```
Ttest_indResult(statistic=-1.2155557697674162, pvalue=0.225343592508479)
```

Stats with statsmodels

```
print(smfrmla.ols("age ~ group", data=brain_vol1).fit().summary())
print("No significant difference in age between patients and controls")
```

Out:

```
OLS Regression Results
==============================================================================
Dep. Variable:                    age   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                  0.002
Method:                 Least Squares   F-statistic:                     1.478
Date:                jeu., 16 mai 2019   Prob (F-statistic):              0.225
Time:                        20:18:24   Log-Likelihood:                -949.69
No. Observations:                 243   AIC:                             1903.
Df Residuals:                     241   BIC:                             1910.
Df Model:                           1
Covariance Type:            nonrobust
==================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept           33.2558      1.305     25.484      0.000      30.685      35.826
group[T.Patient]     1.9735      1.624      1.216      0.225      -1.225       5.172
==============================================================================
Omnibus:                       35.711   Durbin-Watson:                   2.096
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.726
Skew:                           0.569   Prob(JB):                     3.16e-05
Kurtosis:                       2.133   Cond. No.                         3.12
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
No significant difference in age between patients and controls
```

Preliminary tests for sex x group (more/less males in patients than in Controls)

```
crosstab = pd.crosstab(brain_vol1.sex, brain_vol1.group)
print("Obeserved contingency table")
print(crosstab)

chi2, pval, dof, expected = scipy.stats.chi2_contingency(crosstab)

print("Chi2 = %f, pval = %f" % (chi2, pval))
print("Expected contingency table under the null hypothesis")
print(expected)
print("No significant difference in sex between patients and controls")
```

Out:

```
Obeserved contingency table
group  Control  Patient
sex
F           33       55
M           53      102
Chi2 = 0.143253, pval = 0.705068
Expected contingency table under the null hypothesis
[[ 31.14403292  56.85596708]
 [ 54.85596708 100.14403292]]
No significant difference in sex between patients and controls
```

3. Test for differences of atrophy between the patients and the controls

```
print(sm.stats.anova_lm(smfrmla.ols("gm_f ~ group", data=brain_vol1).fit(), typ=2))
print("No significant difference in age between patients and controls")
```

Out:

```
sum_sq     df     F  PR(>F)
group       0.00   1.00 0.01    0.92
Residual    0.46 241.00  nan     nan
No significant difference in age between patients and controls
```

This model is simplistic we should adjust for age and site

```
print(sm.stats.anova_lm(smfrmla.ols(
        "gm_f ~ group + age + site", data=brain_vol1).fit(), typ=2))
print("No significant difference in age between patients and controls")
```

Out:

```
sum_sq     df      F  PR(>F)
group       0.00   1.00  1.82    0.18
site        0.11   5.00 19.79    0.00
age         0.09   1.00 86.86    0.00
Residual    0.25 235.00   nan     nan
No significant difference in age between patients and controls
```

4. Test for interaction between age and clinical status, ie: is the brain atrophy process in patient population faster than in the control population.

```
ancova = smfrmla.ols("gm_f ~ group:age + age + site", data=brain_vol1).fit()
print(sm.stats.anova_lm(ancova, typ=2))

print("= Parameters =")
print(ancova.params)

print("%.3f%% of grey matter loss per year (almost %.1f%% per decade)" %\
      (ancova.params.age * 100, ancova.params.age * 100 * 10))

print("grey matter loss in patients is accelerated by %.3f%% per decade" %
      (ancova.params['group[T.Patient]:age'] * 100 * 10))
```

Out:

```
sum_sq     df      F  PR(>F)
site         0.11   5.00 20.28    0.00
age          0.10   1.00 89.37    0.00
group:age    0.00   1.00  3.28    0.07
Residual     0.25 235.00   nan     nan
= Parameters =
Intercept              0.52
site[T.S3]             0.01
site[T.S4]             0.03
site[T.S5]             0.01
site[T.S7]             0.06
site[T.S8]             0.02
age                   -0.00
group[T.Patient]:age  -0.00
dtype: float64
-0.148% of grey matter loss per year (almost -1.5% per decade)
grey matter loss in patients is accelerated by -0.232% per decade
```

Total running time of the script: (0 minutes 5.184 seconds)

4.3 Multivariate statistics

Multivariate statistics includes all statistical techniques for analyzing samples made of two or more variables. The data set (a $N \times P$ matrix $\mathbf{X}$) is a collection of N independent samples column **vectors** $[\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_N]$ of length P

$$\mathbf{X} = \begin{bmatrix} -\mathbf{x}_1^T- \\ \vdots \\ -\mathbf{x}_i^T- \\ \vdots \\ -\mathbf{x}_P^T- \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1j} & \cdots & x_{1P} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{iP} \\ \vdots & & \vdots & & \vdots \\ x_{N1} & \cdots & x_{Nj} & \cdots & x_{NP} \end{bmatrix} = \begin{bmatrix} x_{11} & \ldots & x_{1P} \\ \vdots & & \vdots \\ & \mathbf{X} & \\ \vdots & & \vdots \\ x_{N1} & \ldots & x_{NP} \end{bmatrix}_{N \times P}.$$

4.3.1 Linear Algebra

Euclidean norm and distance

The Euclidean norm of a vector $\mathbf{a} \in \mathbb{R}^P$ is denoted

$$\|\mathbf{a}\|_2 = \sqrt{\sum_i^P {a_i}^2}$$

The Euclidean distance between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^P$ is

$$\|\mathbf{a} - \mathbf{b}\|_2 = \sqrt{\sum_i^P (a_i - b_i)^2}$$

Dot product and projection

Source: Wikipedia

Algebraic definition

The dot product, denoted ''·'' of two P-dimensional vectors $\mathbf{a} = [a_1, a_2, ..., a_P]$ and $\mathbf{a} = [b_1, b_2, ..., b_P]$ is defined as

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b} = \sum_i a_i b_i = \begin{bmatrix} a_1 & \ldots & \mathbf{a}^T & \ldots & a_P \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ \mathbf{b} \\ \vdots \\ b_P \end{bmatrix}.$$

The Euclidean norm of a vector can be computed using the dot product, as

$$\left\|\mathbf{a}\right\|_2 = \sqrt{\mathbf{a} \cdot \mathbf{a}}.$$

Geometric definition: projection

In Euclidean space, a Euclidean vector is a geometrical object that possesses both a magnitude and a direction. A vector can be pictured as an arrow. Its magnitude is its length, and its direction is the direction that the arrow points. The magnitude of a vector $\mathbf{a}$ is denoted by $\|\mathbf{a}\|_2$. The dot product of two Euclidean vectors $\mathbf{a}$ and $\mathbf{b}$ is defined by

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2\ \|\mathbf{b}\|_2 \cos\theta,$$

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$.

In particular, if $\mathbf{a}$ and $\mathbf{b}$ are orthogonal, then the angle between them is 90° and

$$\mathbf{a} \cdot \mathbf{b} = 0.$$

At the other extreme, if they are codirectional, then the angle between them is 0° and

$$\mathbf{a} \cdot \mathbf{b} = \left\|\mathbf{a}\right\|_2\ \left\|\mathbf{b}\right\|_2$$

This implies that the dot product of a vector $\mathbf{a}$ by itself is

$$\mathbf{a} \cdot \mathbf{a} = \left\|\mathbf{a}\right\|_2^2.$$

The scalar projection (or scalar component) of a Euclidean vector $\mathbf{a}$ in the direction of a Euclidean vector $\mathbf{b}$ is given by

$$a_b = \left\|\mathbf{a}\right\|_2 \cos\theta,$$

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$.

In terms of the geometric definition of the dot product, this can be rewritten

$$a_b = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{b}\|_2},$$

```python
import numpy as np
np.random.seed(42)

a = np.random.randn(10)
b = np.random.randn(10)

np.dot(a, b)
```

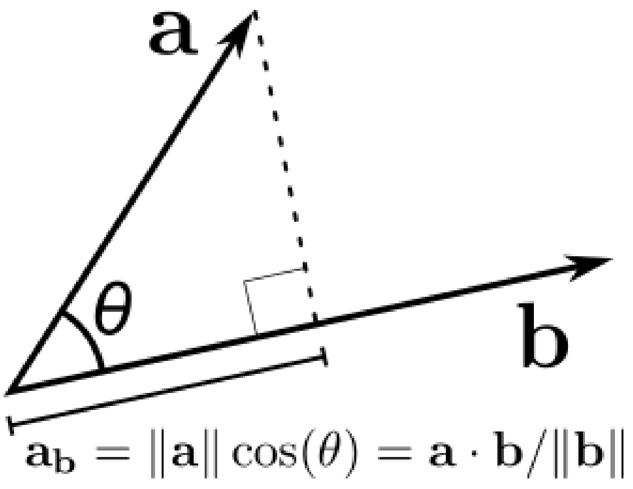


Fig. 5: Projection.

```
-4.085788532659924
```

4.3.2 Mean vector

The mean $(P \times 1)$ column-vector μ whose estimator is

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x_i} = \frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix} x_{i1} \\ \vdots \\ x_{ij} \\ \vdots \\ x_{iP} \end{bmatrix} = \begin{bmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_j \\ \vdots \\ \bar{x}_P \end{bmatrix}.$$

4.3.3 Covariance matrix

- The covariance matrix $\mathbf{\Sigma_{XX}}$ is a **symmetric** positive semi-definite matrix whose element in the j, k position is the covariance between the j^{th} and k^{th} elements of a random vector i.e. the j^{th} and k^{th} columns of $\mathbf{X}$.
- The covariance matrix generalizes the notion of covariance to multiple dimensions.
- The covariance matrix describe the shape of the sample distribution around the mean assuming an elliptical distribution:

$$\mathbf{\Sigma_{XX}} = E(\mathbf{X} - E(\mathbf{X}))^T E(\mathbf{X} - E(\mathbf{X})),$$

whose estimator $\mathbf{S_{XX}}$ is a $P \times P$ matrix given by

$$\mathbf{S_{XX}} = \frac{1}{N-1}(\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T)^T(\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T).$$

If we assume that $\mathbf{X}$ is centered, i.e. $\mathbf{X}$ is replaced by $\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T$ then the estimator is

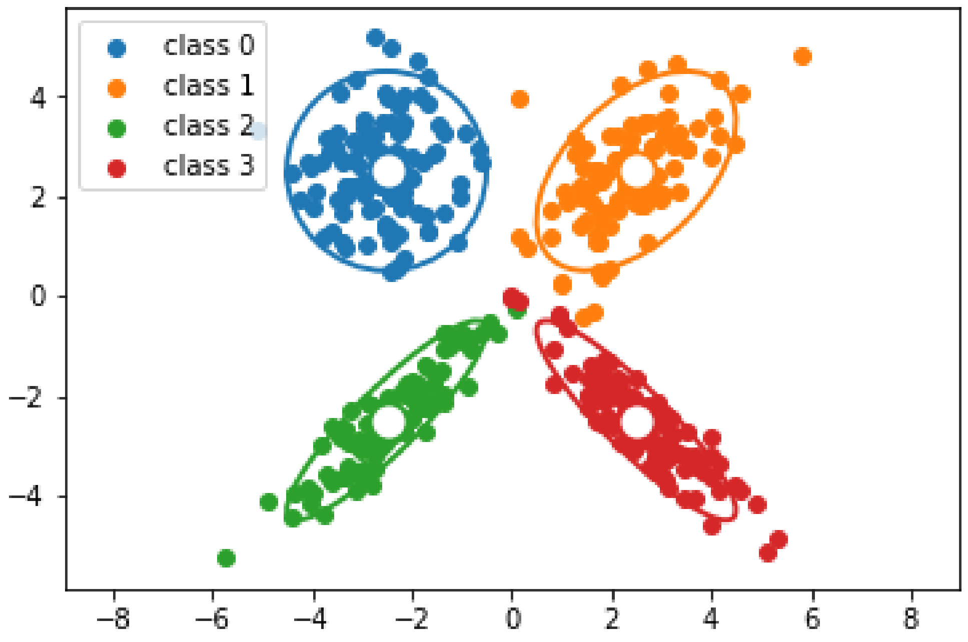
$$\mathbf{S_{XX}} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X} = \frac{1}{N-1}\begin{bmatrix} x_{11} & \cdots & x_{N1} \\ x_{1j} & \cdots & x_{Nj} \\ \vdots & & \vdots \\ x_{1P} & \cdots & x_{NP} \end{bmatrix}\begin{bmatrix} x_{11} & \cdots & x_{1k} & x_{1P} \\ \vdots & & \vdots & \vdots \\ x_{N1} & \cdots & x_{Nk} & x_{NP} \end{bmatrix} = \begin{bmatrix} s_1 & \ldots & s_{1k} & s_{1P} \\ & \ddots & s_{jk} & \vdots \\ & & s_k & s_{kP} \\ & & & s_P \end{bmatrix},$$

where

$$s_{jk} = s_{kj} = \frac{1}{N-1}\mathbf{x_j}^T\mathbf{x_k} = \frac{1}{N-1}\sum_{i=1}^{N} x_{ij}x_{ik}$$

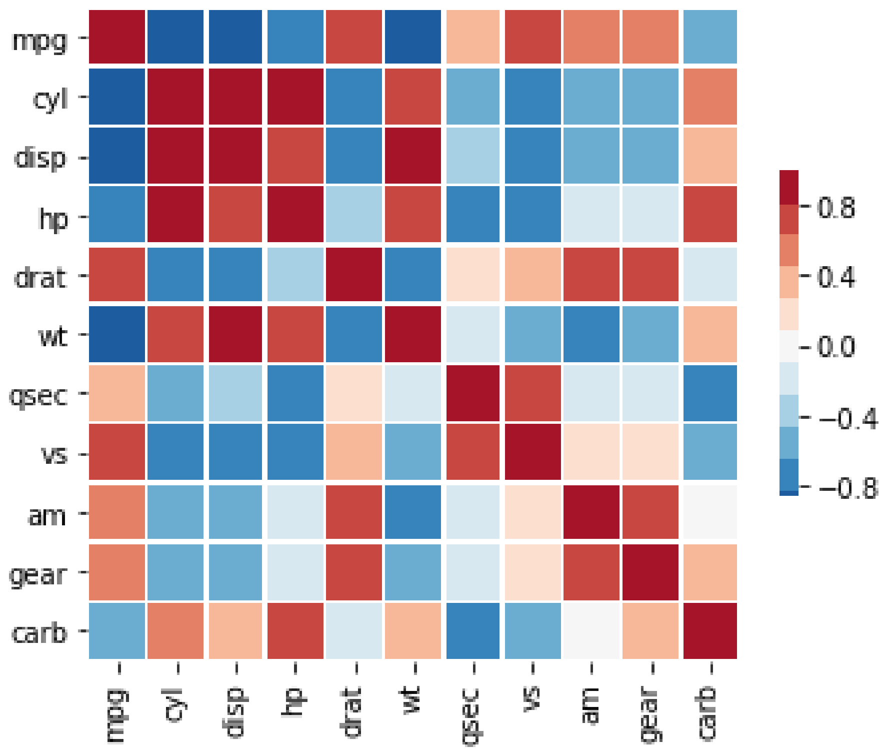
is an estimator of the covariance between the j^{th} and k^{th} variables.

```
## Avoid warnings and force inline plot
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
##
import numpy as np
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
import pystatsml.plot_utils
import seaborn as sns  # nice color

np.random.seed(42)
colors = sns.color_palette()

n_samples, n_features = 100, 2

mean, Cov, X = [None] * 4, [None] * 4, [None] * 4
mean[0] = np.array([-2.5, 2.5])
Cov[0] = np.array([[1, 0],
                   [0, 1]])

mean[1] = np.array([2.5, 2.5])
Cov[1] = np.array([[1, .5],
                   [.5, 1]])

mean[2] = np.array([-2.5, -2.5])
Cov[2] = np.array([[1, .9],
                   [.9, 1]])

mean[3] = np.array([2.5, -2.5])
Cov[3] = np.array([[1, -.9],
                   [-.9, 1]])

# Generate dataset
for i in range(len(mean)):
    X[i] = np.random.multivariate_normal(mean[i], Cov[i], n_samples)

# Plot
for i in range(len(mean)):
    # Points
    plt.scatter(X[i][:, 0], X[i][:, 1], color=colors[i], label="class %i" % i)
    # Means
    plt.scatter(mean[i][0], mean[i][1], marker="o", s=200, facecolors='w',
                edgecolors=colors[i], linewidth=2)
    # Ellipses representing the covariance matrices
    pystatsml.plot_utils.plot_cov_ellipse(Cov[i], pos=mean[i], facecolor='none',
                                          linewidth=2, edgecolor=colors[i])

plt.axis('equal')
_ = plt.legend(loc='upper left')
```



4.3.4 Correlation matrix

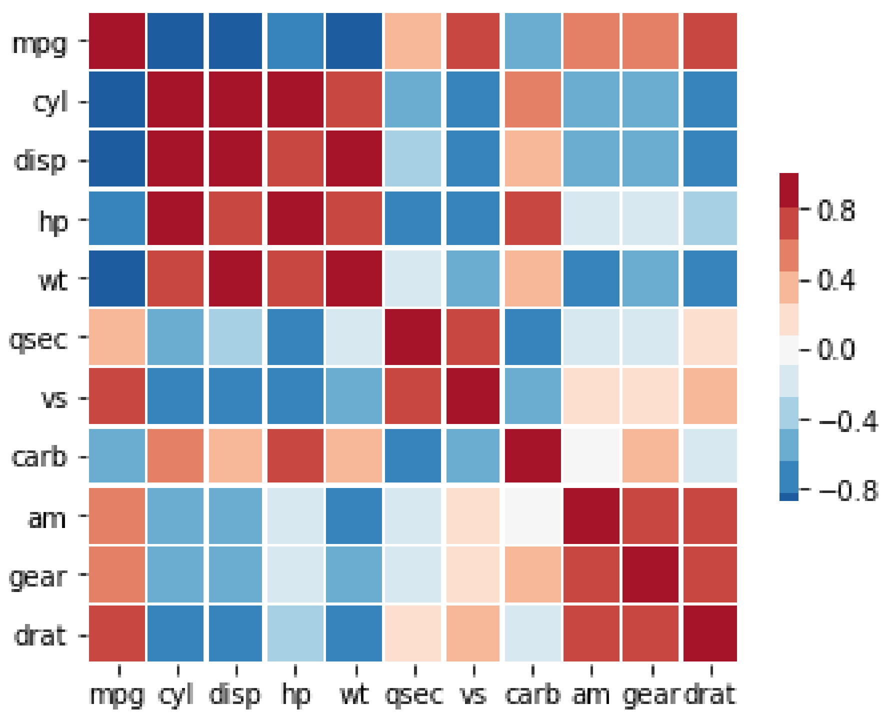
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

url = 'https://python-graph-gallery.com/wp-content/uploads/mtcars.csv'
df = pd.read_csv(url)

# Compute the correlation matrix
corr = df.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(5.5, 4.5))
cmap = sns.color_palette("RdBu_r", 11)
# Draw the heatmap with the mask and correct aspect ratio
_ = sns.heatmap(corr, mask=None, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



Re-order correlation matrix using AgglomerativeClustering

```
# convert correlation to distances
d = 2 * (1 - np.abs(corr))

from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=3, linkage='single', affinity="precomputed
↪").fit(d)
lab=0

clusters = [list(corr.columns[clustering.labels_==lab]) for lab in set(clustering.labels_
↪)]
print(clusters)

reordered = np.concatenate(clusters)

R = corr.loc[reordered, reordered]

f, ax = plt.subplots(figsize=(5.5, 4.5))
# Draw the heatmap with the mask and correct aspect ratio
_ = sns.heatmap(R, mask=None, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
[['mpg', 'cyl', 'disp', 'hp', 'wt', 'qsec', 'vs', 'carb'], ['am', 'gear'], ['drat']]
```



4.3.5 Precision matrix

In statistics, precision is the reciprocal of the variance, and the precision matrix is the matrix inverse of the covariance matrix.

It is related to **partial correlations** that measures the degree of association between two variables, while controlling the effect of other variables.

```
import numpy as np

Cov = np.array([[1.0, 0.9, 0.9, 0.0, 0.0, 0.0],
                [0.9, 1.0, 0.9, 0.0, 0.0, 0.0],
                [0.9, 0.9, 1.0, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 1.0, 0.9, 0.0],
                [0.0, 0.0, 0.0, 0.9, 1.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]])

print("# Precision matrix:")
Prec = np.linalg.inv(Cov)
print(Prec.round(2))

print("# Partial correlations:")
Pcor = np.zeros(Prec.shape)
Pcor[::] = np.NaN

for i, j in zip(*np.triu_indices_from(Prec, 1)):
    Pcor[i, j] = - Prec[i, j] / np.sqrt(Prec[i, i] * Prec[j, j])

print(Pcor.round(2))
```

```
# Precision matrix:
[[ 6.79 -3.21 -3.21  0.    0.    0.  ]
 [-3.21  6.79 -3.21  0.    0.    0.  ]
 [-3.21 -3.21  6.79  0.    0.    0.  ]
 [ 0.   -0.   -0.    5.26 -4.74 -0.  ]
 [ 0.    0.    0.   -4.74  5.26  0.  ]
 [ 0.    0.    0.    0.    0.    1.  ]]
# Partial correlations:
[[  nan  0.47  0.47 -0.   -0.   -0.  ]
 [  nan   nan  0.47 -0.   -0.   -0.  ]
 [  nan   nan   nan -0.   -0.   -0.  ]
 [  nan   nan   nan   nan  0.9   0.  ]
 [  nan   nan   nan   nan   nan -0.  ]
 [  nan   nan   nan   nan   nan   nan]]
```

4.3.6 Mahalanobis distance

- The Mahalanobis distance is a measure of the distance between two points $\mathbf{x}$ and μ where the dispersion (i.e. the covariance structure) of the samples is taken into account.
- The dispersion is considered through covariance matrix.

This is formally expressed as

$$D_M(\mathbf{x}, \mu) = \sqrt{(\mathbf{x} - \mu)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu)}.$$

Intuitions

- Distances along the principal directions of dispersion are contracted since they correspond to likely dispersion of points.
- Distances othogonal to the principal directions of dispersion are dilated since they correspond to unlikely dispersion of points.

For example

$$D_M(\mathbf{1}) = \sqrt{\mathbf{1}^T \mathbf{\Sigma}^{-1} \mathbf{1}}.$$

```python
ones  = np.ones(Cov.shape[0])
d_euc = np.sqrt(np.dot(ones, ones))
d_mah = np.sqrt(np.dot(np.dot(ones, Prec), ones))

print("Euclidean norm of ones=%.2f. Mahalanobis norm of ones=%.2f" % (d_euc, d_mah))
```

```
Euclidean norm of ones=2.45. Mahalanobis norm of ones=1.77
```

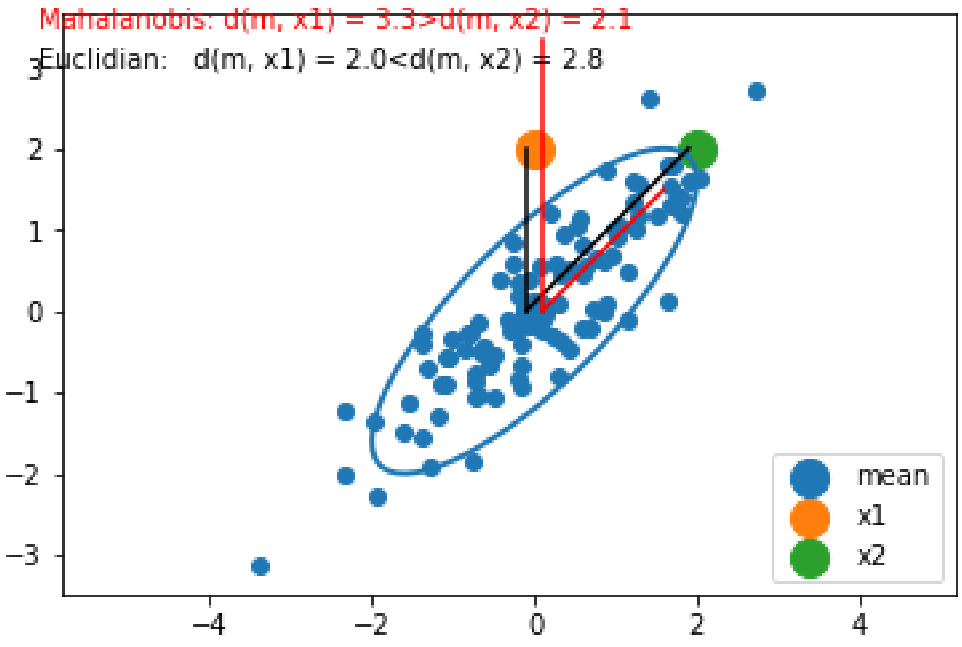
The first dot product that distances along the principal directions of dispersion are contracted:

```python
print(np.dot(ones, Prec))
```

```
[0.35714286 0.35714286 0.35714286 0.52631579 0.52631579 1.        ]
```

```python
import numpy as np
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
import pystatsml.plot_utils
%matplotlib inline
np.random.seed(40)
colors = sns.color_palette()

mean = np.array([0, 0])
Cov = np.array([[1, .8],
                [.8, 1]])
samples = np.random.multivariate_normal(mean, Cov, 100)
x1 = np.array([0, 2])
x2 = np.array([2, 2])

plt.scatter(samples[:, 0], samples[:, 1], color=colors[0])
plt.scatter(mean[0], mean[1], color=colors[0], s=200, label="mean")
plt.scatter(x1[0], x1[1], color=colors[1], s=200, label="x1")
plt.scatter(x2[0], x2[1], color=colors[2], s=200, label="x2")

# plot covariance ellipsis
pystatsml.plot_utils.plot_cov_ellipse(Cov, pos=mean, facecolor='none',
                                      linewidth=2, edgecolor=colors[0])
# Compute distances
d2_m_x1 = scipy.spatial.distance.euclidean(mean, x1)
d2_m_x2 = scipy.spatial.distance.euclidean(mean, x2)

Covi = scipy.linalg.inv(Cov)
dm_m_x1 = scipy.spatial.distance.mahalanobis(mean, x1, Covi)
dm_m_x2 = scipy.spatial.distance.mahalanobis(mean, x2, Covi)

# Plot distances
vm_x1 = (x1 - mean) / d2_m_x1
vm_x2 = (x2 - mean) / d2_m_x2
jitter = .1
plt.plot([mean[0] - jitter, d2_m_x1 * vm_x1[0] - jitter],
         [mean[1], d2_m_x1 * vm_x1[1]], color='k')
plt.plot([mean[0] - jitter, d2_m_x2 * vm_x2[0] - jitter],
         [mean[1], d2_m_x2 * vm_x2[1]], color='k')

plt.plot([mean[0] + jitter, dm_m_x1 * vm_x1[0] + jitter],
         [mean[1], dm_m_x1 * vm_x1[1]], color='r')
plt.plot([mean[0] + jitter, dm_m_x2 * vm_x2[0] + jitter],
         [mean[1], dm_m_x2 * vm_x2[1]], color='r')

plt.legend(loc='lower right')
plt.text(-6.1, 3,
         'Euclidian:   d(m, x1) = %.1f<d(m, x2) = %.1f' % (d2_m_x1, d2_m_x2), color='k')
plt.text(-6.1, 3.5,
         'Mahalanobis: d(m, x1) = %.1f>d(m, x2) = %.1f' % (dm_m_x1, dm_m_x2), color='r')

plt.axis('equal')
print('Euclidian   d(m, x1) = %.2f < d(m, x2) = %.2f' % (d2_m_x1, d2_m_x2))
print('Mahalanobis d(m, x1) = %.2f > d(m, x2) = %.2f' % (dm_m_x1, dm_m_x2))
```

```
Euclidian   d(m, x1) = 2.00 < d(m, x2) = 2.83
Mahalanobis d(m, x1) = 3.33 > d(m, x2) = 2.11
```



If the covariance matrix is the identity matrix, the Mahalanobis distance reduces to the Euclidean distance. If the covariance matrix is diagonal, then the resulting distance measure is called a normalized Euclidean distance.

More generally, the Mahalanobis distance is a measure of the distance between a point x and a distribution $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma})$. It is a multi-dimensional generalization of the idea of measuring how many standard deviations away x is from the mean. This distance is zero if x is at the mean, and grows as x moves away from the mean: along each principal component axis, it measures the number of standard deviations from x to the mean of the distribution.

4.3.7 Multivariate normal distribution

The distribution, or probability density function (PDF) (sometimes just density), of a continuous random variable is a function that describes the relative likelihood for this random variable to take on a given value.

The multivariate normal distribution, or multivariate Gaussian distribution, of a P-dimensional random vector $\mathbf{x} = [x_1, x_2, \ldots, x_P]^T$ is

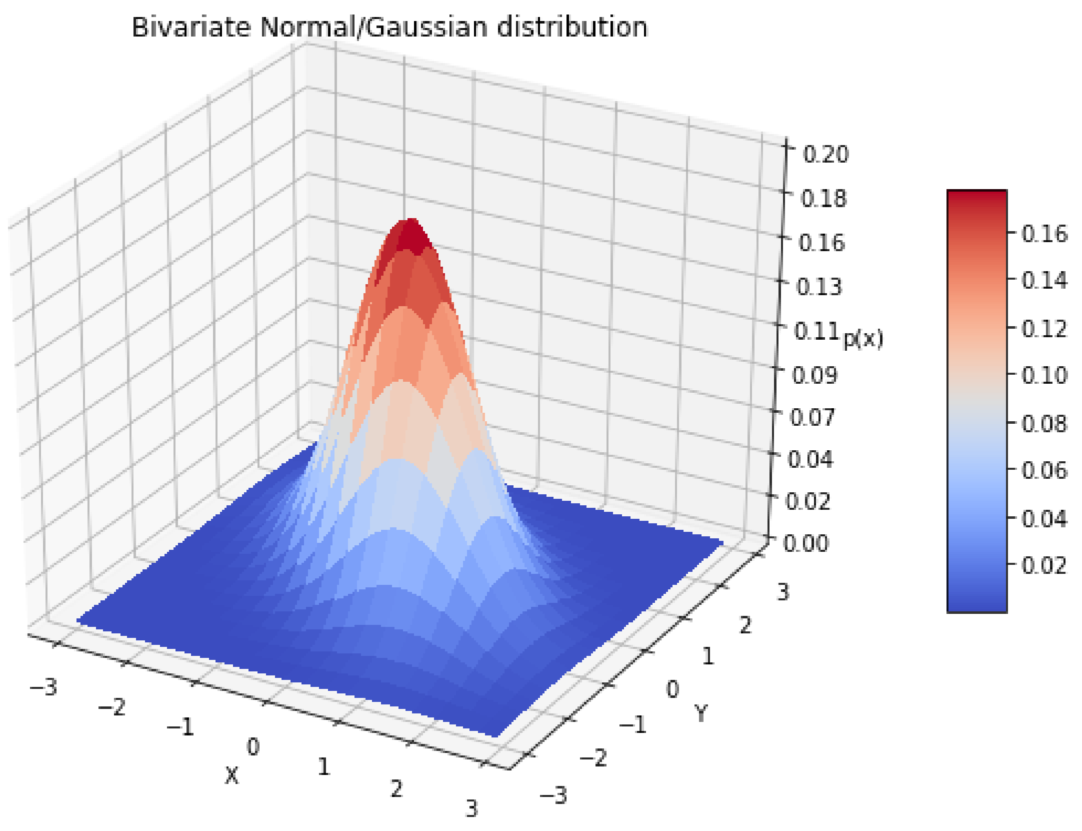
$$\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{P/2}|\mathbf{\Sigma}|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\}.$$

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
```

(continues on next page)

(continued from previous page)

```
def multivariate_normal_pdf(X, mean, sigma):
    """Multivariate normal probability density function over X (n_samples x n_features)"""
    P = X.shape[1]
    det = np.linalg.det(sigma)
    norm_const = 1.0 / (((2*np.pi) ** (P/2)) * np.sqrt(det))
    X_mu = X - mu
    inv = np.linalg.inv(sigma)
    d2 = np.sum(np.dot(X_mu, inv) * X_mu, axis=1)
    return norm_const * np.exp(-0.5 * d2)

# mean and covariance
mu = np.array([0, 0])
sigma = np.array([[1, -.5],
                  [-.5, 1]])

# x, y grid
x, y = np.mgrid[-3:3:.1, -3:3:.1]
X = np.stack((x.ravel(), y.ravel())).T
norm = multivariate_normal_pdf(X, mean, sigma).reshape(x.shape)

# Do it with scipy
norm_scpy = multivariate_normal(mu, sigma).pdf(np.stack((x, y), axis=2))
assert np.allclose(norm, norm_scpy)

# Plot
fig = plt.figure(figsize=(10, 7))
ax = fig.gca(projection='3d')
surf = ax.plot_surface(x, y, norm, rstride=3,
        cstride=3, cmap=plt.cm.coolwarm,
        linewidth=1, antialiased=False
    )

ax.set_zlim(0, 0.2)
ax.zaxis.set_major_locator(plt.LinearLocator(10))
ax.zaxis.set_major_formatter(plt.FormatStrFormatter('%.02f'))

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('p(x)')

plt.title('Bivariate Normal/Gaussian distribution')
fig.colorbar(surf, shrink=0.5, aspect=7, cmap=plt.cm.coolwarm)
plt.show()
```



4.3.8 Exercises

Dot product and Euclidean norm

Given $\mathbf{a} = [2, 1]^T$ and $\mathbf{b} = [1, 1]^T$

1. Write a function `euclidean(x)` that computes the Euclidean norm of vector, $\mathbf{x}$.
2. Compute the Euclidean norm of $\mathbf{a}$.
3. Compute the Euclidean distance of $\|\mathbf{a} - \mathbf{b}\|_2$.
4. Compute the projection of $\mathbf{b}$ in the direction of vector $\mathbf{a}$: b_a.
5. Simulate a dataset $\mathbf{X}$ of $N = 100$ samples of 2-dimensional vectors.
6. Project all samples in the direction of the vector $\mathbf{a}$.

Covariance matrix and Mahalanobis norm

1. Sample a dataset $\mathbf{X}$ of $N = 100$ samples of 2-dimensional vectors from the bivariate normal distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$ where $\mu = [1, 1]^T$ and $\mathbf{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8, 1 \end{bmatrix}$.
2. Compute the mean vector $\bar{\mathbf{x}}$ and center $\mathbf{X}$. Compare the estimated mean $\bar{\mathbf{x}}$ to the true mean, μ.
3. Compute the empirical covariance matrix $\mathbf{S}$. Compare the estimated covariance matrix $\mathbf{S}$ to the true covariance matrix, $\mathbf{\Sigma}$.

4. Compute $\mathbf{S}^{-1}$ (`Sinv`) the inverse of the covariance matrix by using `scipy.linalg.inv(S)`.
5. Write a function `mahalanobis(x, xbar, Sinv)` that computes the Mahalanobis distance of a vector $\mathbf{x}$ to the mean, $\bar{\mathbf{x}}$.
6. Compute the Mahalanobis and Euclidean distances of each sample $\mathbf{x}_i$ to the mean $\bar{\mathbf{x}}$. Store the results in a 100×2 dataframe.

4.4 Time Series in python

Two libraries:

- Pandas: https://pandas.pydata.org/pandas-docs/stable/timeseries.html
- scipy http://www.statsmodels.org/devel/tsa.html

4.4.1 Stationarity

A TS is said to be stationary if its statistical properties such as mean, variance remain constant over time.

- constant mean
- constant variance
- an autocovariance that does not depend on time.

what is making a TS non-stationary. There are 2 major reasons behind non-stationaruty of a TS:

1. Trend – varying mean over time. For eg, in this case we saw that on average, the number of passengers was growing over time.
2. Seasonality – variations at specific time-frames. eg people might have a tendency to buy cars in a particular month because of pay increment or festivals.

4.4.2 Pandas Time Series Data Structure

A Series is similar to a list or an array in Python. It represents a series of values (numeric or otherwise) such as a column of data. It provides additional functionality, methods, and operators, which make it a more powerful version of a list.

```
import pandas as pd
import numpy as np

# Create a Series from a list
ser = pd.Series([1, 3])
print(ser)

# String as index
prices = {'apple': 4.99,
         'banana': 1.99,
         'orange': 3.99}
ser = pd.Series(prices)
```

(continues on next page)

(continued from previous page)

```
print(ser)

x = pd.Series(np.arange(1,3), index=[x for x in 'ab'])
print(x)
print(x['b'])
```

```
0    1
1    3
dtype: int64
apple     4.99
banana    1.99
orange    3.99
dtype: float64
a    1
b    2
dtype: int64
2
```

4.4.3 Time Series Analysis of Google Trends

source: https://www.datacamp.com/community/tutorials/time-series-analysis-tutorial

Get Google Trends data of keywords such as 'diet' and 'gym' and see how they vary over time while learning about trends and seasonality in time series data.

In the Facebook Live code along session on the 4th of January, we checked out Google trends data of keywords 'diet', 'gym' and 'finance' to see how they vary over time. We asked ourselves if there could be more searches for these terms in January when we're all trying to turn over a new leaf?

In this tutorial, you'll go through the code that we put together during the session step by step. You're not going to do much mathematics but you are going to do the following:

- Read data
- Recode data
- Exploratory Data Analysis

4.4.4 Read data

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

# Plot appears on its own windows
%matplotlib inline
# Tools / Preferences / Ipython Console  / Graphics  / Graphics Backend / Backend:␣
↪“automatic”
# Interactive Matplotlib Jupyter Notebook
# %matplotlib inline
```

(continues on next page)

(continued from previous page)

```
try:
    url = "https://raw.githubusercontent.com/datacamp/datacamp_facebook_live_ny_
↪resolution/master/datasets/multiTimeline.csv"
    df = pd.read_csv(url, skiprows=2)
except:
    df = pd.read_csv("../datasets/multiTimeline.csv", skiprows=2)

print(df.head())

# Rename columns
df.columns = ['month', 'diet', 'gym', 'finance']

# Describe
print(df.describe())
```

```
     Month  diet: (Worldwide)  gym: (Worldwide)  finance: (Worldwide)
0  2004-01                100                31                    48
1  2004-02                 75                26                    49
2  2004-03                 67                24                    47
3  2004-04                 70                22                    48
4  2004-05                 72                22                    43
             diet         gym     finance
count  168.000000  168.000000  168.000000
mean    49.642857   34.690476   47.148810
std      8.033080    8.134316    4.972547
min     34.000000   22.000000   38.000000
25%     44.000000   28.000000   44.000000
50%     48.500000   32.500000   46.000000
75%     53.000000   41.000000   50.000000
max    100.000000   58.000000   73.000000
```

4.4.5 Recode data

Next, you'll turn the 'month' column into a DateTime data type and make it the index of the DataFrame.

Note that you do this because you saw in the result of the .info() method that the 'Month' column was actually an of data type object. Now, that generic data type encapsulates everything from strings to integers, etc. That's not exactly what you want when you want to be looking at time series data. That's why you'll use .to_datetime() to convert the 'month' column in your DataFrame to a DateTime.

Be careful! Make sure to include the inplace argument when you're setting the index of the DataFrame df so that you actually alter the original index and set it to the 'month' column.

```
df.month = pd.to_datetime(df.month)
df.set_index('month', inplace=True)

print(df.head())
```

```
            diet  gym  finance
month
2004-01-01   100   31       48
```

(continues on next page)

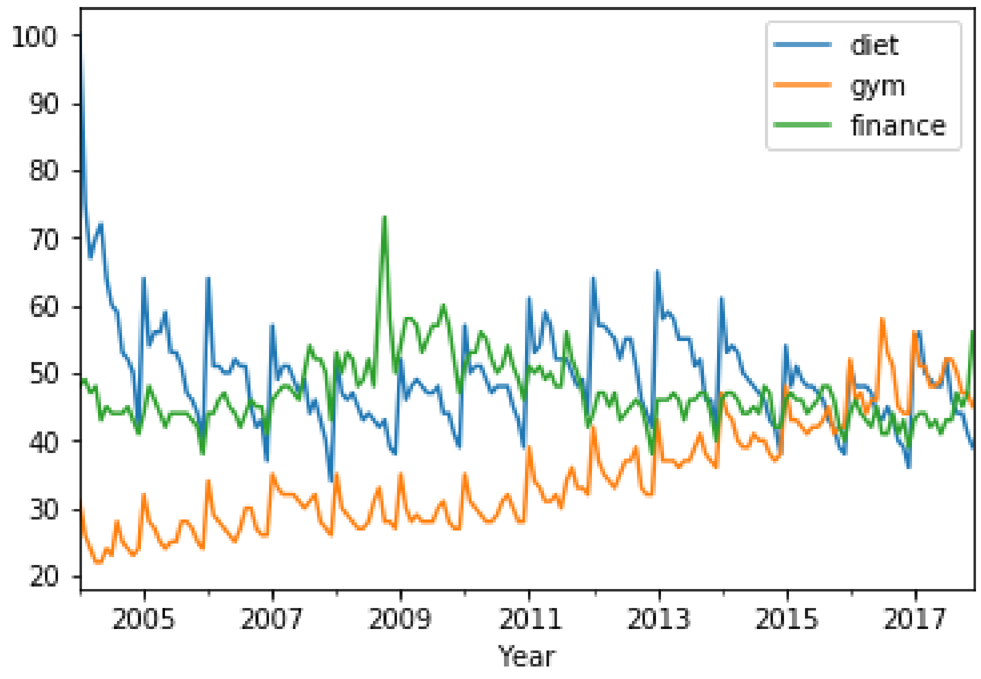
(continued from previous page)

```
2004-02-01    75   26        49
2004-03-01    67   24        47
2004-04-01    70   22        48
2004-05-01    72   22        43
```

4.4.6 Exploratory Data Analysis

You can use a built-in pandas visualization method .plot() to plot your data as 3 line plots on a single figure (one for each column, namely, 'diet', 'gym', and 'finance').

```
df.plot()
plt.xlabel('Year');

# change figure parameters
# df.plot(figsize=(20,10), linewidth=5, fontsize=20)

# Plot single column
# df[['diet']].plot(figsize=(20,10), linewidth=5, fontsize=20)
# plt.xlabel('Year', fontsize=20);
```



Note that this data is relative. As you can read on Google trends:

Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. Likewise a score of 0 means the term was less than 1% as popular as the peak.

4.4.7 Resampling, Smoothing, Windowing, Rolling average: Trends

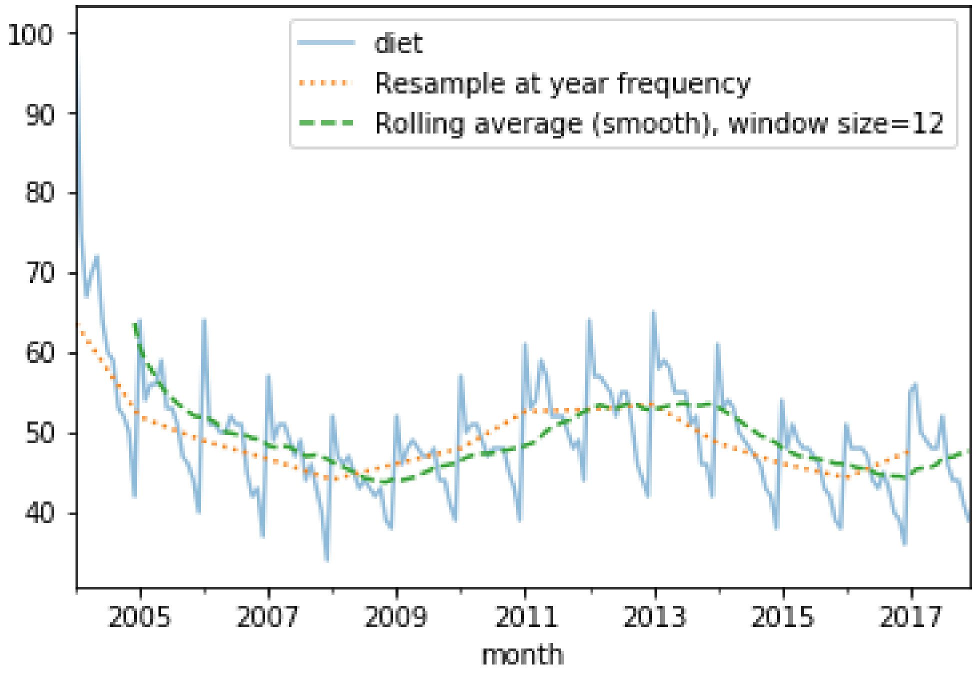
Rolling average, for each time point, take the average of the points on either side of it. Note that the number of points is specified by a window size.

Remove Seasonality with pandas Series.

See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html A: 'year end frequency' year frequency

```
diet = df['diet']

diet_resamp_yr = diet.resample('A').mean()
diet_roll_yr = diet.rolling(12).mean()

ax = diet.plot(alpha=0.5, style='-') # store axis (ax) for latter plots
diet_resamp_yr.plot(style=':', label='Resample at year frequency', ax=ax)
diet_roll_yr.plot(style='--', label='Rolling average (smooth), window size=12', ax=ax)
ax.legend()
```

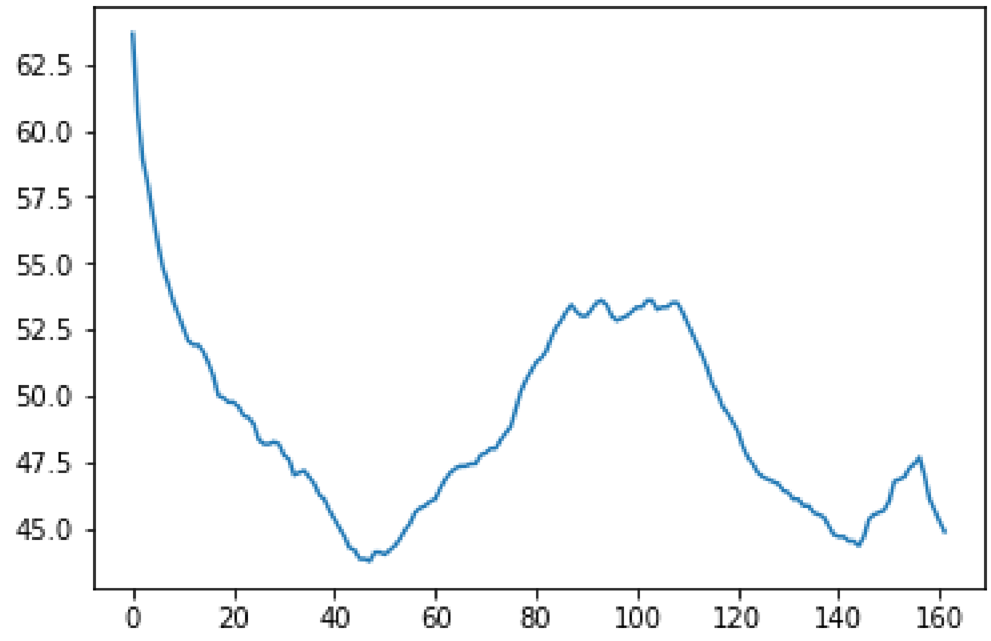
```
<matplotlib.legend.Legend at 0x7f12ec1eec50>
```



Rolling average (smoothing) with Numpy

```
x = np.asarray(df[['diet']])
win = 12
win_half = int(win / 2)
# print([((idx-win_half), (idx+win_half)) for idx in np.arange(win_half, len(x))])

diet_smooth = np.array([x[(idx-win_half):(idx+win_half)].mean() for idx in np.arange(win_
↪half, len(x))])
plt.plot(diet_smooth)
```

```
[<matplotlib.lines.Line2D at 0x7f12ec163f98>]
```

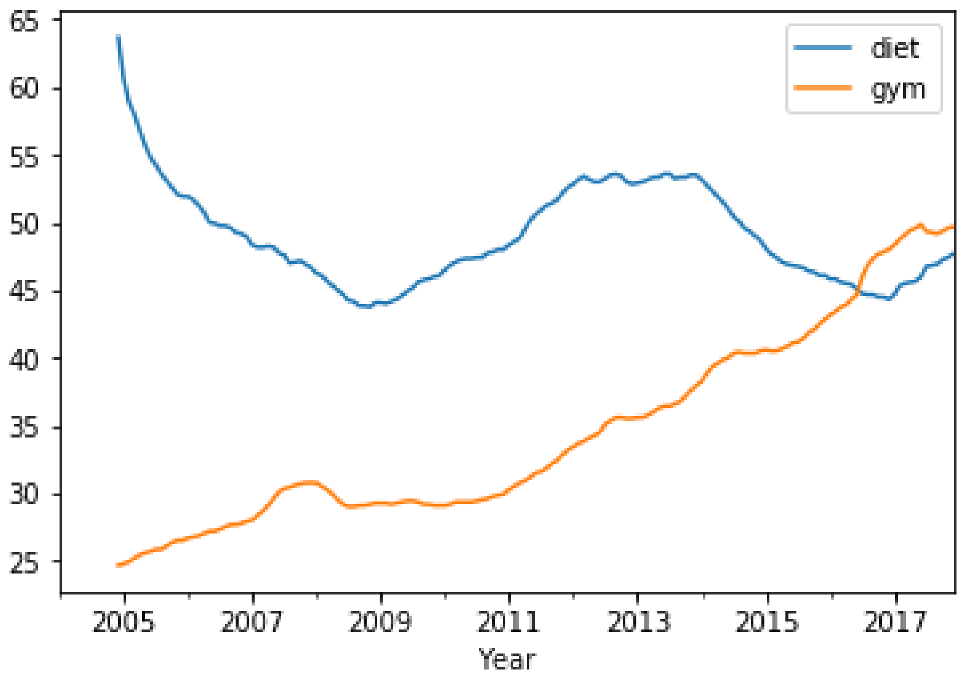


Trends Plot Diet and Gym

Build a new DataFrame which is the concatenation diet and gym smoothed data

```
gym = df['gym']

df_avg = pd.concat([diet.rolling(12).mean(), gym.rolling(12).mean()], axis=1)
df_avg.plot()
plt.xlabel('Year')
```

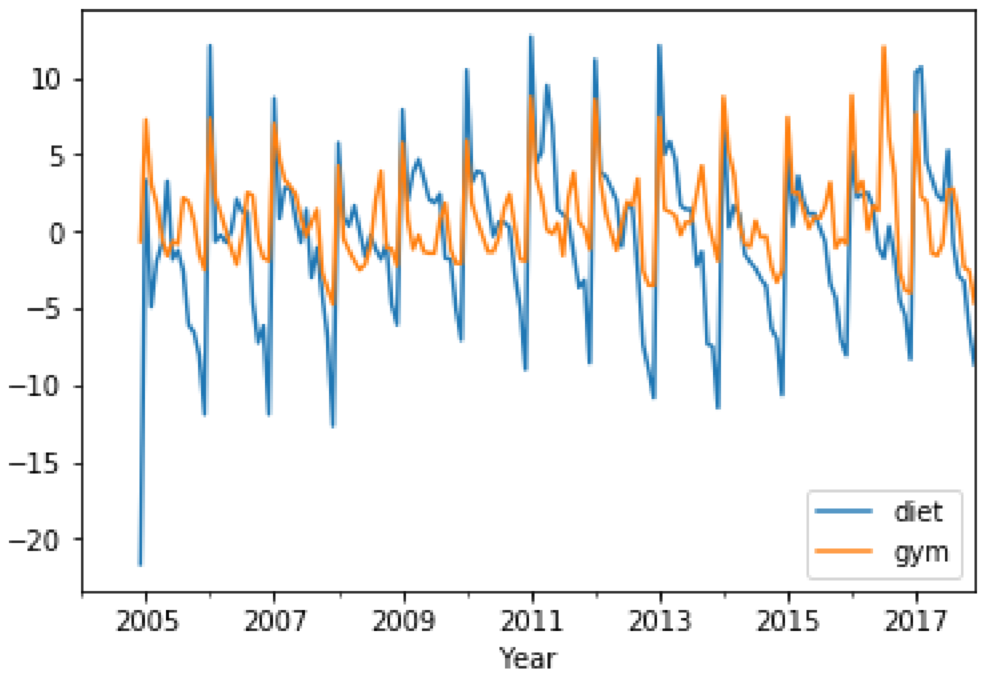
```
Text(0.5, 0, 'Year')
```



Detrending

```
df_dtrend = df[["diet", "gym"]] - df_avg
df_dtrend.plot()
plt.xlabel('Year')
```

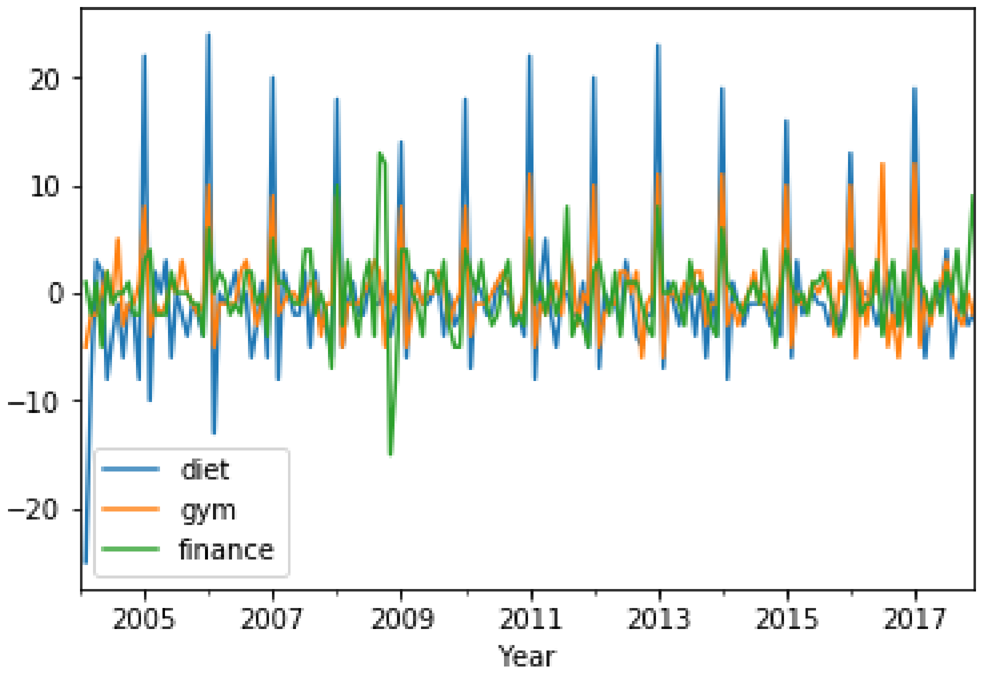
```
Text(0.5, 0, 'Year')
```



4.4.8 First-order differencing: Seasonal Patterns

```
# diff = original - shiftted data
# (exclude first term for some implementation details)
assert np.all((diet.diff() == diet - diet.shift())[1:])

df.diff().plot()
plt.xlabel('Year')
```

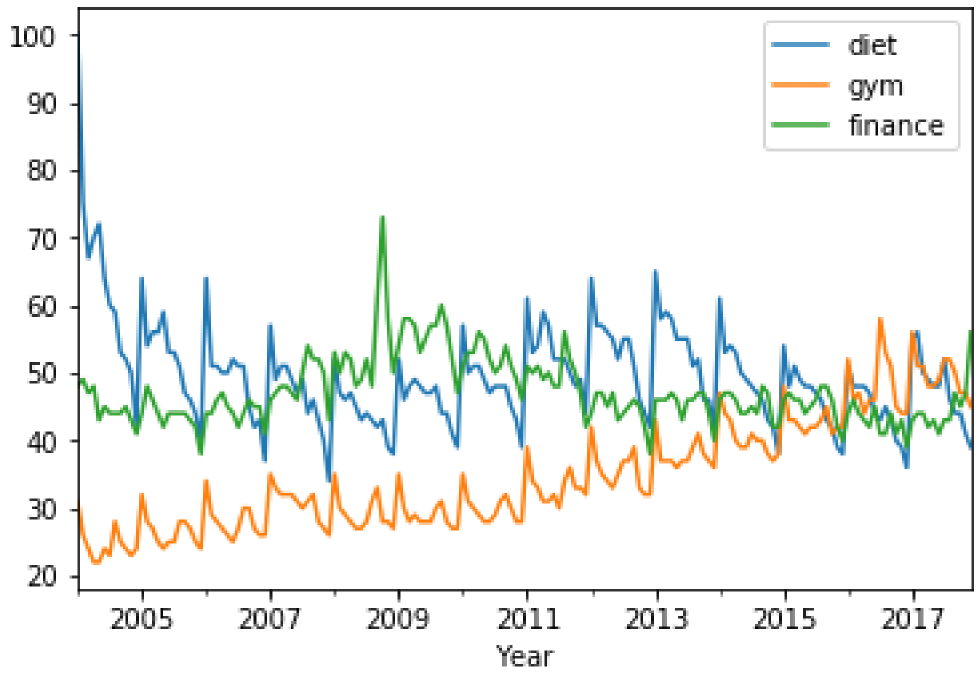
```
Text(0.5, 0, 'Year')
```



4.4.9 Periodicity and Correlation

```
df.plot()
plt.xlabel('Year');
print(df.corr())
```

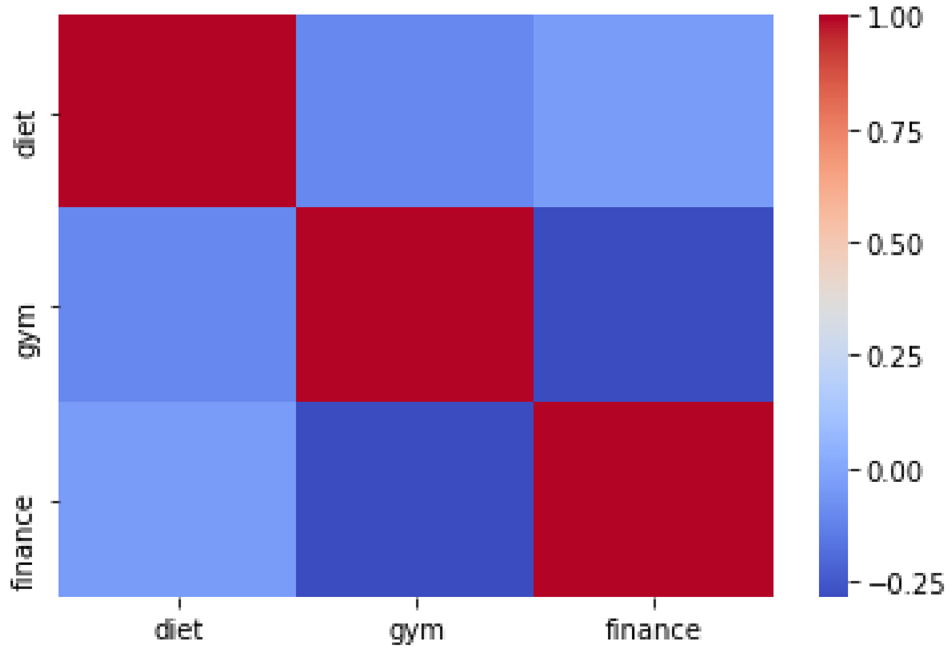
```
             diet       gym   finance
diet     1.000000 -0.100764 -0.034639
gym     -0.100764  1.000000 -0.284279
finance -0.034639 -0.284279  1.000000
```



Plot correlation matrix

```
sns.heatmap(df.corr(), cmap="coolwarm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f12e9e05a90>
```



'diet' and 'gym' are negatively correlated! Remember that you have a seasonal and a trend component. From the correlation coefficient, 'diet' and 'gym' are negatively correlated:

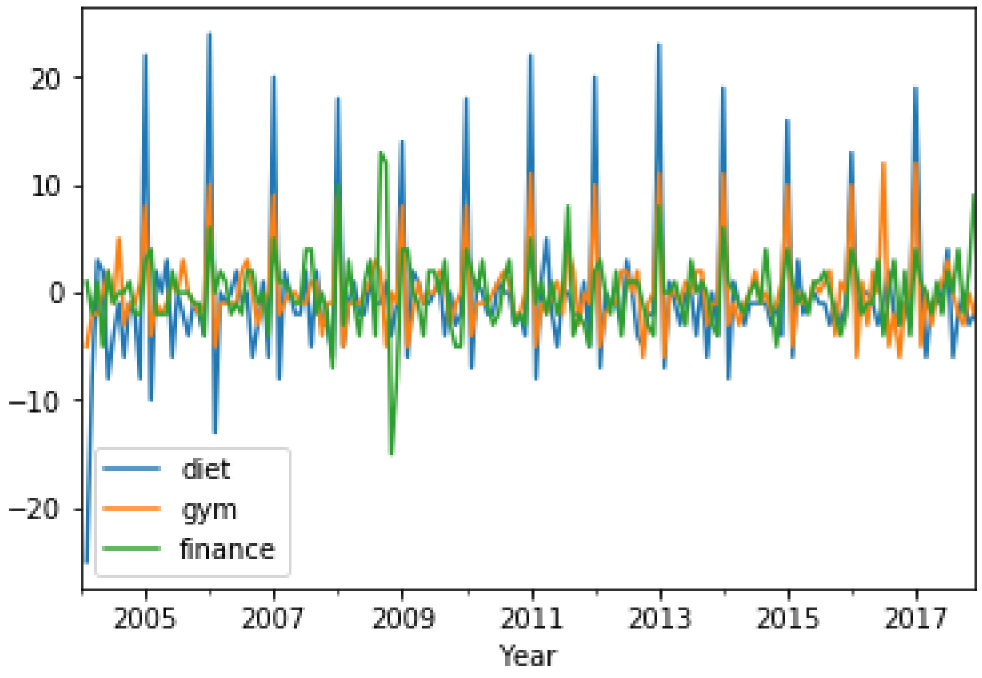
- trends components are negatively correlated.
- seasonal components would positively correlated and their

The actual correlation coefficient is actually capturing both of those.

Seasonal correlation: correlation of the first-order differences of these time series

```
df.diff().plot()
plt.xlabel('Year');

print(df.diff().corr())
```

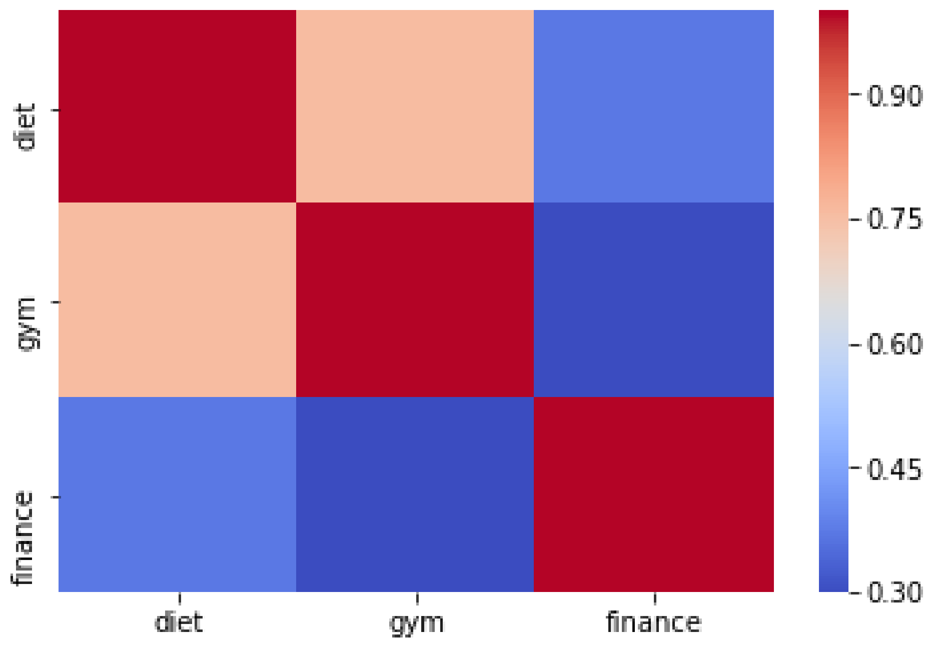
```
             diet       gym   finance
diet     1.000000  0.758707  0.373828
gym      0.758707  1.000000  0.301111
finance  0.373828  0.301111  1.000000
```



Plot correlation matrix

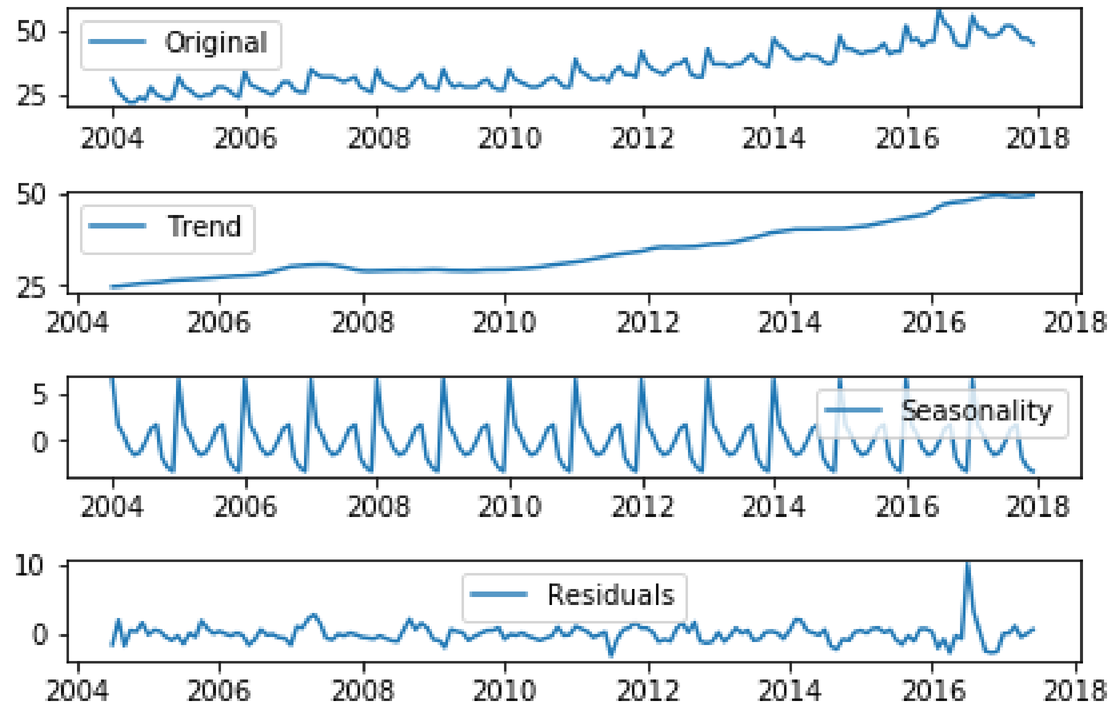
```
sns.heatmap(df.diff().corr(), cmap="coolwarm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f12ec06c438>
```



Decomposing time serie in trend, seasonality and residuals

```
from statsmodels.tsa.seasonal import seasonal_decompose

x = gym

x = x.astype(float) # force float
decomposition = seasonal_decompose(x)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(x, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



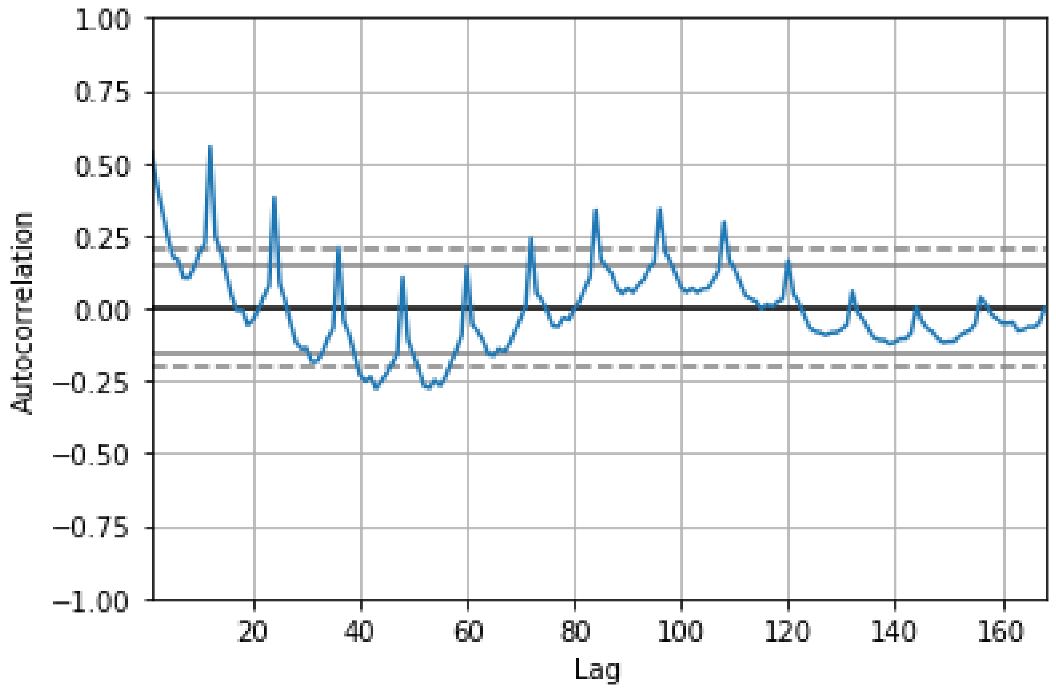
4.4.10 Autocorrelation

A time series is periodic if it repeats itself at equally spaced intervals, say, every 12 months. Autocorrelation Function (ACF): It is a measure of the correlation between the TS with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant t1…t2 with series at instant t1-5…t2-5 (t1-5 and t2 being end points).

Plot

```
# from pandas.plotting import autocorrelation_plot
from pandas.plotting import autocorrelation_plot

x = df["diet"].astype(float)
autocorrelation_plot(x)
```

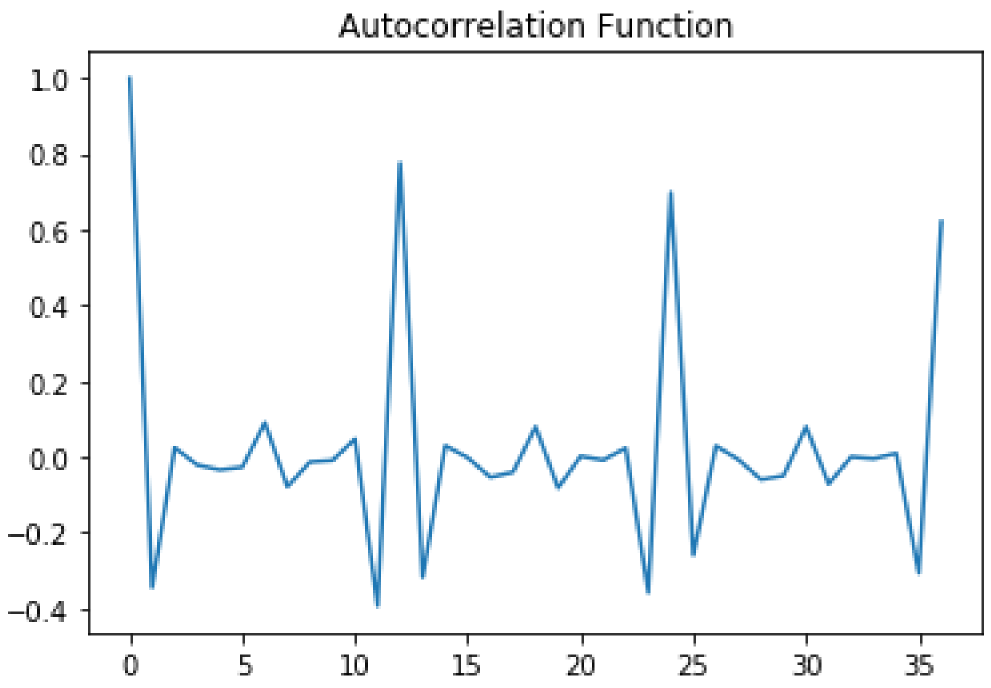
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f12e9a6d4e0>
```



Compute Autocorrelation Function (ACF)

```
from statsmodels.tsa.stattools import acf

x_diff = x.diff().dropna() # first item is NA
lag_acf = acf(x_diff, nlags=36)
plt.plot(lag_acf)
plt.title('Autocorrelation Function')
```

```
Text(0.5, 1.0, 'Autocorrelation Function')
```



ACF peaks every 12 months: Time series is correlated with itself shifted by 12 months.

4.4.11 Time Series Forecasting with Python using Autoregressive Moving Average (ARMA) models

Source:

- https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781783553358/7/ch07lvl1sec77/arma-models
- http://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model
- ARIMA: https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

ARMA models are often used to forecast a time series. These models combine autoregressive and moving average models. In moving average models, we assume that a variable is the sum of the mean of the time series and a linear combination of noise components.

The autoregressive and moving average models can have different orders. In general, we can define an ARMA model with p autoregressive terms and q moving average terms as follows:

$$x_t = \sum_i^p a_i x_{t-i} + \sum_i^q b_i \varepsilon_{t-i} + \varepsilon_t$$

Choosing p and q

Plot the partial autocorrelation functions for an estimate of p, and likewise using the autocorrelation functions for an estimate of q.

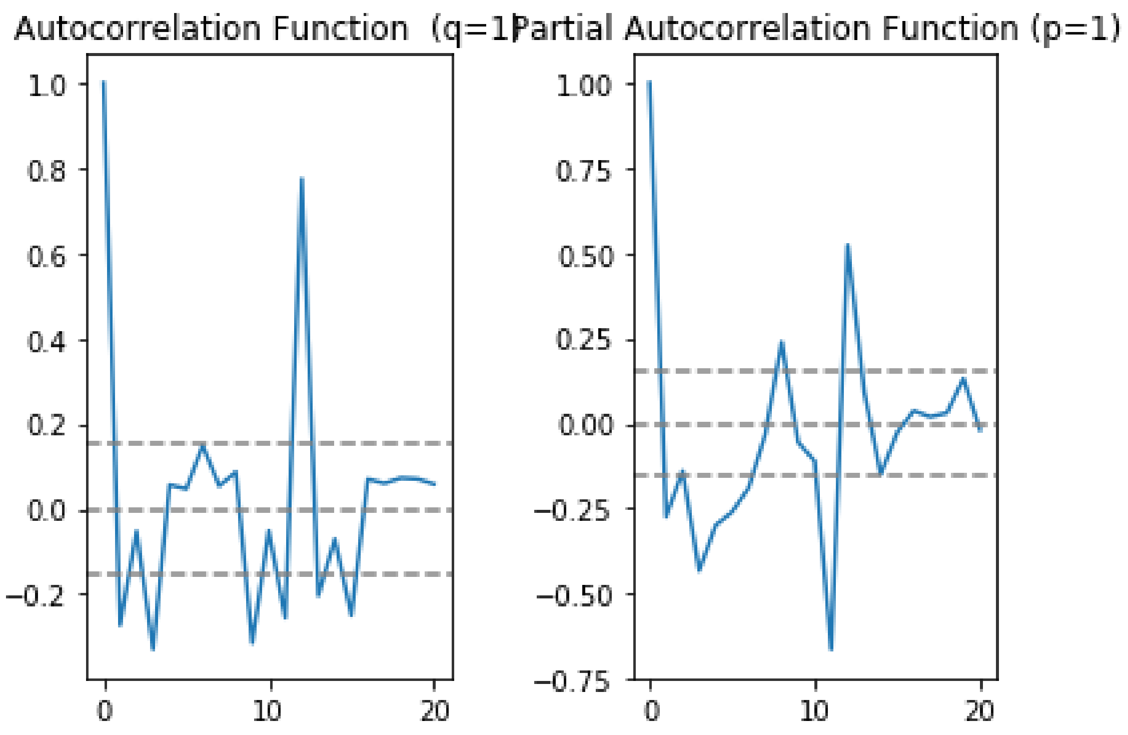
Partial Autocorrelation Function (PACF): This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Eg at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

```python
from statsmodels.tsa.stattools import acf, pacf

x = df["gym"].astype(float)

x_diff = x.diff().dropna() # first item is NA
# ACF and PACF plots:

lag_acf = acf(x_diff, nlags=20)
lag_pacf = pacf(x_diff, nlags=20, method='ols')

#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function  (q=1)')

#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
```

(continues on next page)

(continued from previous page)

```
plt.axhline(y=-1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function (p=1)')
plt.tight_layout()
```



In this plot, the two dotted lines on either sides of 0 are the confidence interevals. These can be used to determine the p and q values as:

- p: The lag value where the PACF chart crosses the upper confidence interval for the first time, in this case p=1.
- q: The lag value where the ACF chart crosses the upper confidence interval for the first time, in this case q=1.

Fit ARMA model with statsmodels

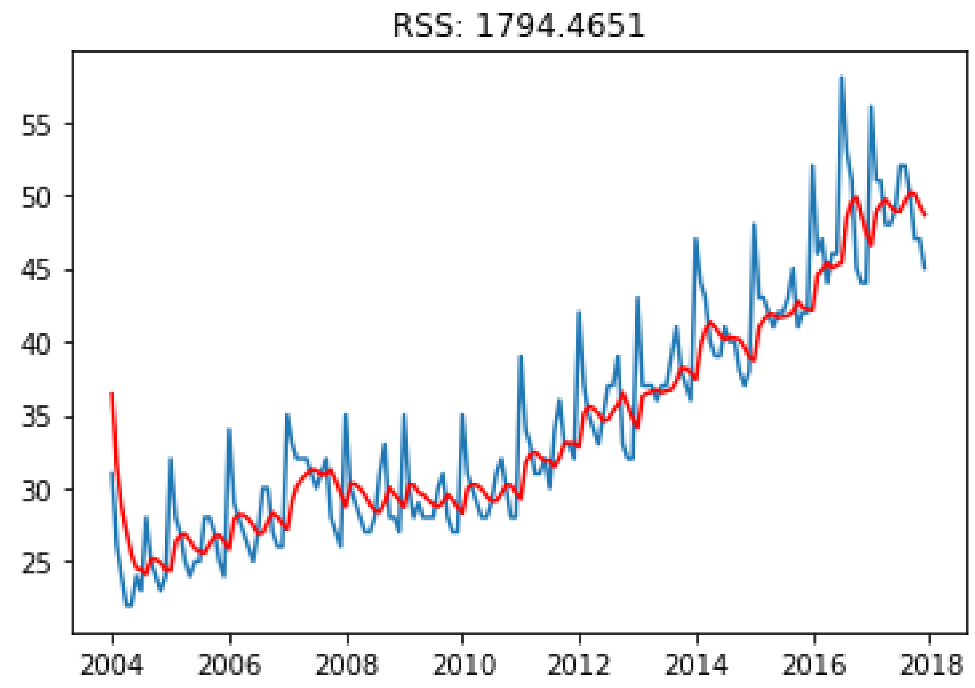
1. Define the model by calling `ARMA()` and passing in the p and q parameters.
2. The model is prepared on the training data by calling the `fit()` function.
3. Predictions can be made by calling the `predict()` function and specifying the index of the time or times to be predicted.

```
from statsmodels.tsa.arima_model import ARMA

model = ARMA(x, order=(1, 1)).fit() # fit model

print(model.summary())
plt.plot(x)
plt.plot(model.predict(), color='red')
plt.title('RSS: %.4f'% sum((model.fittedvalues-x)**2))
```

```
/home/edouard/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_
↪model.py:171: ValueWarning: No frequency information was provided, so inferred␣
↪frequency MS will be used.
  % freq, ValueWarning)
/home/edouard/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/
↪tsa_model.py:191: FutureWarning: Creating a DatetimeIndex by passing range␣
↪endpoints is deprecated.  Use pandas.date_range instead.
  start=index[0], end=index[-1], freq=freq)
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                    gym   No. Observations:                  168
Model:                     ARMA(1, 1)   Log Likelihood                -436.852
Method:                       css-mle   S.D. of innovations              3.229
Date:                Thu, 16 May 2019   AIC                            881.704
Time:                        20:15:20   BIC                            894.200
Sample:                    01-01-2004   HQIC                           886.776
                         - 12-01-2017
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         36.4315      8.827      4.127      0.000      19.131      53.732
ar.L1.gym      0.9967      0.005    220.566      0.000       0.988       1.006
ma.L1.gym     -0.7494      0.054    -13.931      0.000      -0.855      -0.644
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0033           +0.0000j            1.0033            0.0000
MA.1            1.3344           +0.0000j            1.3344            0.0000
-----------------------------------------------------------------------------
```

```
Text(0.5, 1.0, 'RSS: 1794.4651')
```


RSS: 1794.4651