# Lecture 11:

# Data Fitting and Regression

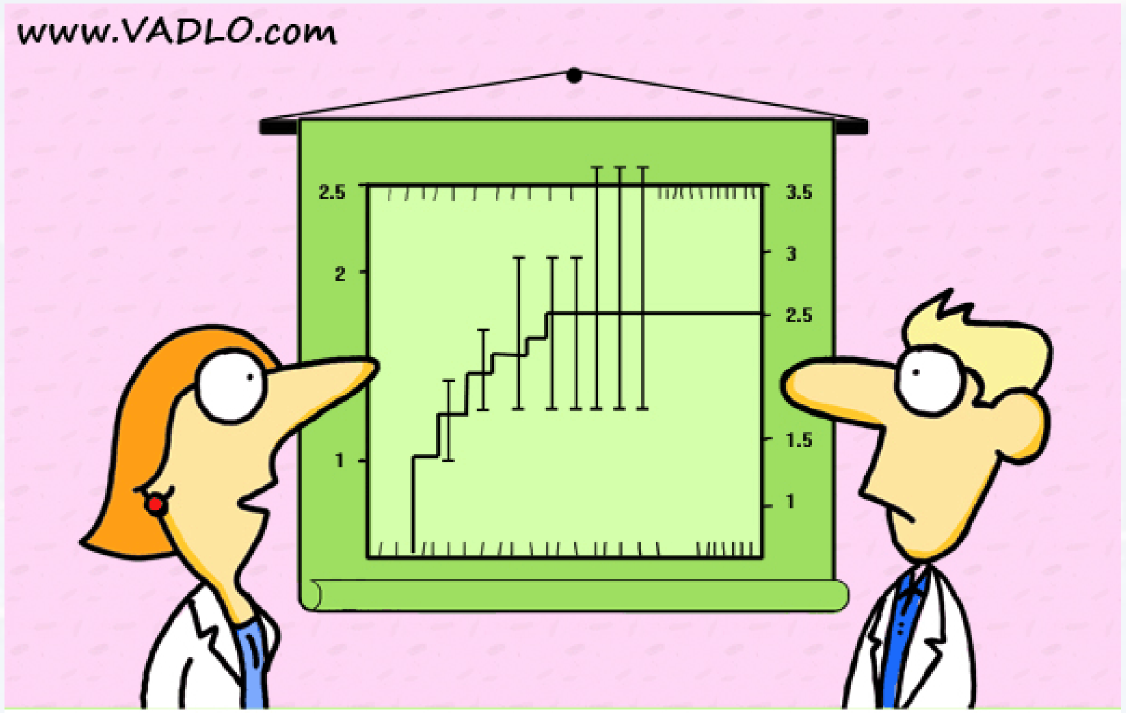
Markus Hohle

University California, Berkeley

**Numerical Methods for Computational Science**

## Course Map

## Outline



“Did you really have to show the error bars?”

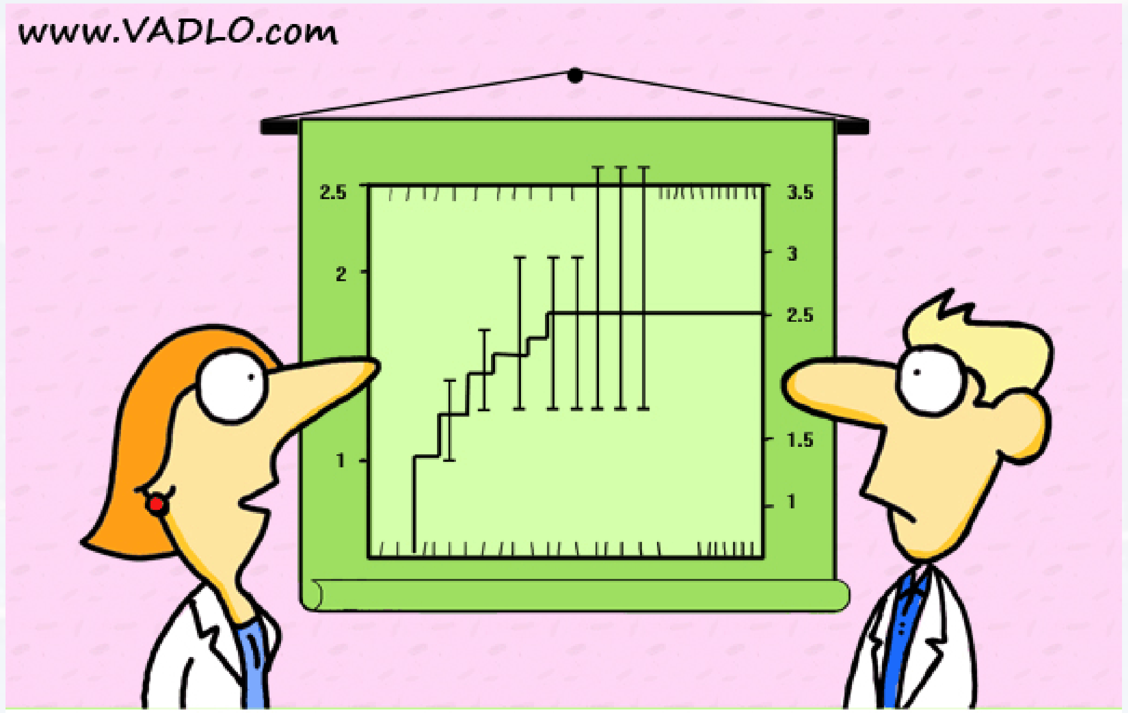
**Linear Regression**

- The Math
- What is Linear?
- Stats
- a Python Example

**Logistic Regression**

**Curve Fitting**

## Outline



“Did you really have to show the error bars?”

**Linear Regression**

- The Math

- What is Linear?

- Stats

- a Python Example

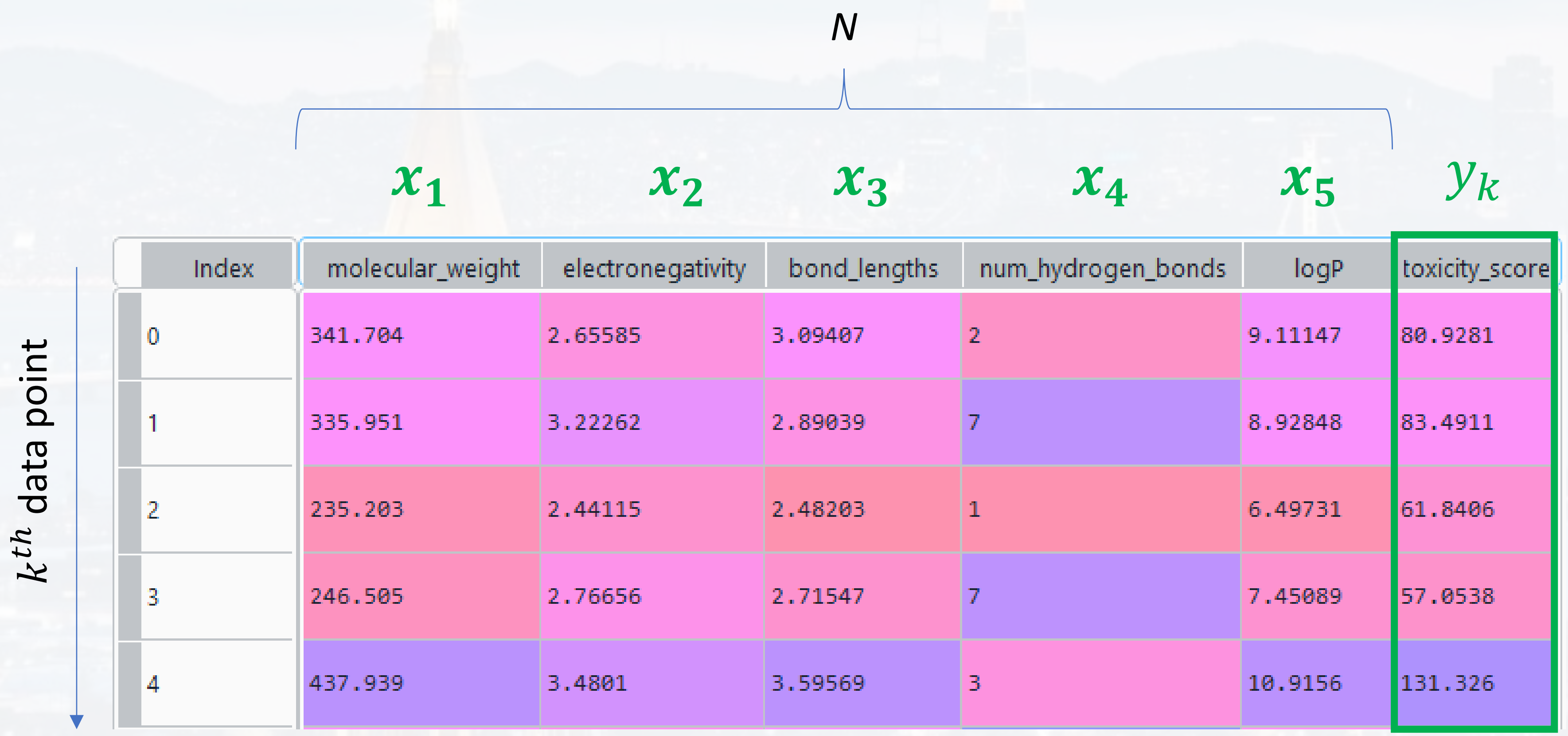
**Logistic Regression**

**Curve Fitting**

## Data Fitting and Regression:

**problem:** a parameter *y* depends on many **regressors** $x_n$

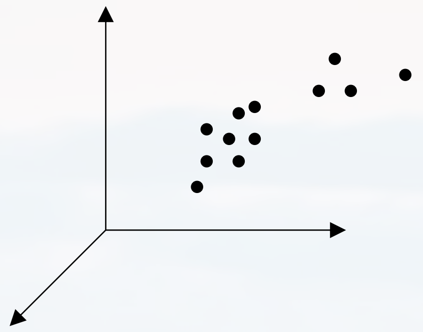
**goal:** finding a model that tells us **how** *y* depends on each $x_n$
deriving a model to **predict** *y* from new data points


$N$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$y_k$
$k^{th}$ data point

| Index | molecular_weight | electronegativity | bond_lengths | num_hydrogen_bonds | logP | toxicity_score |
|---|---|---|---|---|---|---|
| 0 | 341.704 | 2.65585 | 3.09407 | 2 | 9.11147 | 80.9281 |
| 1 | 335.951 | 3.22262 | 2.89039 | 7 | 8.92848 | 83.4911 |
| 2 | 235.203 | 2.44115 | 2.48203 | 1 | 6.49731 | 61.8406 |
| 3 | 246.505 | 2.76656 | 2.71547 | 7 | 7.45089 | 57.0538 |
| 4 | 437.939 | 3.4801 | 3.59569 | 3 | 10.9156 | 131.326 |

## Data Fitting and Regression:

**The Math**
What is Linear?
Stats
a Python Example

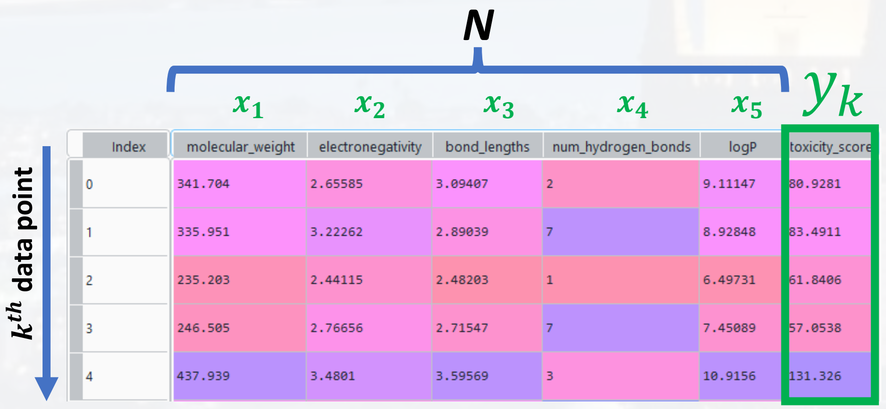


idea: data point $y_k$ in $N$ dimensional space

$\rightarrow y_k = f(x_1, \dots x_n, \dots x_N) + \epsilon$ for each data point $k$

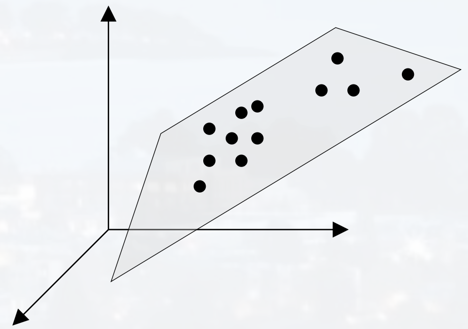
ansatz: $$\boxed{y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon}$$ ***linear*** combination

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors **(how a regressor contributes to the response)** |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |



$N$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$y_k$
$k^{th}$ data point

| Index | molecular_weight | electronegativity | bond_lengths | num_hydrogen_bonds | logP | toxicity_score |
|---|---|---|---|---|---|---|
| 0 | 341.704 | 2.65585 | 3.09407 | 2 | 9.11147 | 80.9281 |
| 1 | 335.951 | 3.22262 | 2.89039 | 7 | 8.92848 | 83.4911 |
| 2 | 235.203 | 2.44115 | 2.48203 | 1 | 6.49731 | 61.8406 |
| 3 | 246.505 | 2.76656 | 2.71547 | 7 | 7.45089 | 57.0538 |
| 4 | 437.939 | 3.4801 | 3.59569 | 3 | 10.9156 | 131.326 |



## Data Fitting and Regression:

linear ≠ not curved

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n \, x_n^n + \epsilon$$

...is still linear    just define: $\bar{x}_n := x_n^n$

$$y_k = \beta_1 \, x_n^{\beta_2}$$

...is still linear

just use log: $\bar{y}_k = \log(y_k) = \log(\beta_1) + \beta_2 \log(x_n)$
$= \bar{\beta}_1 + \beta_2 \bar{x}_n$

As long as we can recover the linear structure by any transformation → it is linear

in part. log scaling is quite common    examples:

- log fold change (DESeq/RNASeq)
- log odds ratio (comparing models, HMM)
- sound → dB is a log unit
- log incidence rates (medical studies)
- percentiles (medical studies)
-.....

| | |
|---|---|
| **y:** | **response** |
| **x:** | **regressors** |
| **β:** | **factors** |
| $\boldsymbol{\beta_0}$**:** | **intercept** |
| **ε:** | **error** |

## Data Fitting and Regression:

...what is **not** linear?

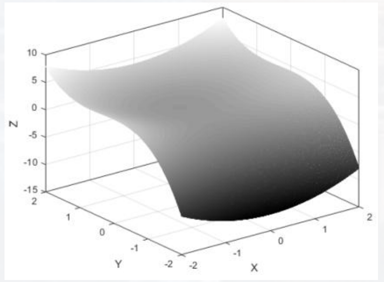
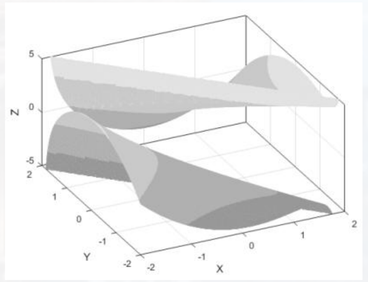
$y_k = \beta_0 + \beta_1 x_n^{\boxed{\beta_2}}$ log trick does not work here

general: linear refers to the **factors**

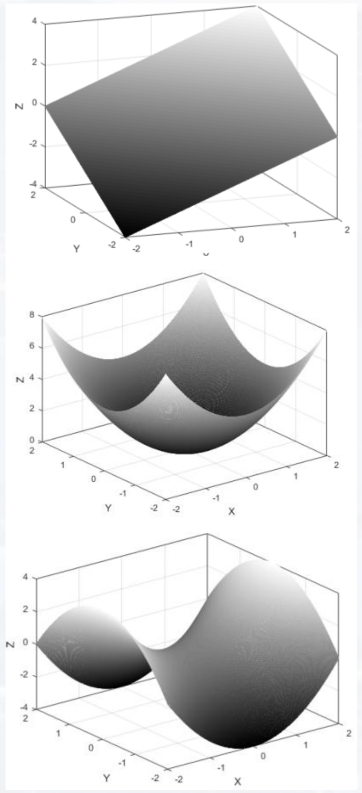
$y_k = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ 2*D* plane in 3*D* space

$y_k = \beta_0 + \beta_1 {x_1}^2 + \beta_2 {x_2}^2$ 2*D* parabolic

$y_k = \beta_0 + \beta_1 {x_1}^2 - \beta_2 {x_2}^2$ 2*D* hyperbolic

...and many more...



**all linear**

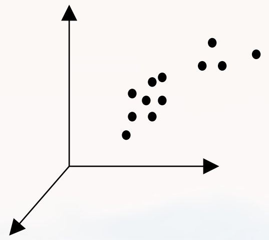
$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

| | |
|---|---|
| **y:** | **response** |
| **x:** | **regressors** |
| **β:** | **factors** |
| $\boldsymbol{\beta_0}$**:** | **intercept** |
| **ε:** | **error** |

# Data Fitting and Regression:

for *K* data points in *N* dimensional space

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

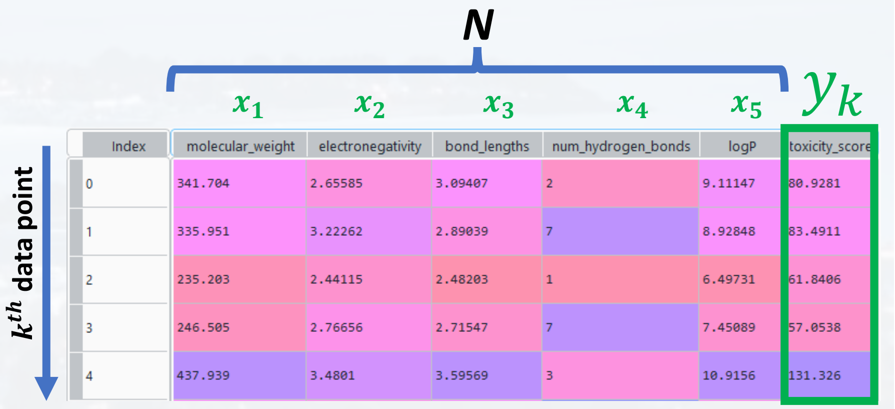


The Math
What is Linear?
**Stats**
a Python Example

| | |
|---|---|
| **y:** | **response** |
| **x:** | **regressors** |
| **β:** | **factors** |
| **$\beta_0$:** | **intercept** |
| **ε:** | **error** |

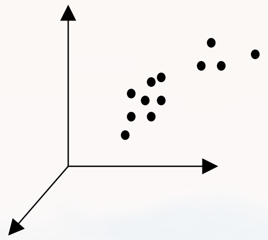
$$\underbrace{\begin{pmatrix} y_1 \\ \dots \\ y_k \\ \dots \\ \dots \\ y_K \end{pmatrix}}_{Y} = \underbrace{\begin{pmatrix} 1 & x_{11} & x_{12} \dots & x_{1n} \dots & x_{1N} \\ \dots & \dots & & \dots & \\ 1 & x_{k1} & & x_{kn} & \\ 1 & \dots & & \dots & \\ 1 & x_{K1} & x_{K2} \dots & x_{Kn} \dots & x_{KN} \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_n \\ \dots \\ \beta_N \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_k \\ \dots \\ \varepsilon_K \end{pmatrix}}_{\varepsilon}$$

$$\boxed{Y = X\beta + \varepsilon}$$



| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $y_k$ |
|---|---|---|---|---|---|---|
| Index | molecular_weight | electronegativity | bond_lengths | num_hydrogen_bonds | logP | toxicity_score |
| 0 | 341.704 | 2.65585 | 3.09407 | 2 | 9.11147 | 80.9281 |
| 1 | 335.951 | 3.22262 | 2.89039 | 7 | 8.92848 | 83.4911 |
| 2 | 235.203 | 2.44115 | 2.48203 | 1 | 6.49731 | 61.8406 |
| 3 | 246.505 | 2.76656 | 2.71547 | 7 | 7.45089 | 57.0538 |
| 4 | 437.939 | 3.4801 | 3.59569 | 3 | 10.9156 | 131.326 |

## Data Fitting and Regression:



for *K* data points in *N* dimensional space

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

$$\underbrace{\begin{pmatrix} y_1 \\ \dots \\ y_k \\ \dots \\ \dots \\ y_K \end{pmatrix}}_{Y} = \underbrace{\begin{pmatrix} 1 & x_{11} & x_{12} \dots x_{1n} \dots x_{1N} \\ \dots & \dots & \dots \\ 1 & x_{k1} & x_{kn} \\ 1 & \dots & \dots \\ 1 & x_{K1} & x_{K2} \dots x_{Kn} \dots x_{KN} \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_n \\ \dots \\ \beta_N \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_k \\ \dots \\ \varepsilon_K \end{pmatrix}}_{\varepsilon}$$

$$\boxed{\boldsymbol{Y = X\beta + \varepsilon}}$$

| | |
|---|---|
| **y:** | **response** |
| **x:** | **regressors** |
| **β:** | **factors** |
| **$\beta_0$:** | **intercept** |
| **ε:** | **error** |

<u>fitting:</u> finding the best β in terms of minimizing the errors

$$(Y - X\beta)^T (Y - X\beta) = \sum_k {\varepsilon_k}^2$$

$$\frac{\partial}{\partial \beta} \sum_k {\varepsilon_k}^2 = 0 \longrightarrow \beta_{best} = \hat{\beta} = (X^T X)^{-1} X^T Y \longrightarrow \hat{Y} = X\hat{\beta} = X(X^T X)^{-1} X^T Y$$

**the model**

X and Y are all observables

## Data Fitting and Regression:

$$\boxed{Y = X\beta + \varepsilon}$$

finding the best β in terms of minimizing the errors

$$(Y - X\beta)^T(Y - X\beta) = \sum_k {\varepsilon_k}^2$$

**the model**

$$\hat{Y} = X\hat{\beta} = \underbrace{X\left(X^TX\right)^{-1}X^T}_{\text{hat matrix } H}Y$$

| | |
|---|---|
| **y:** | **response** |
| **x:** | **regressors** |
| **β:** | **factors** |
| **$\beta_0$:** | **intercept** |
| **ε:** | **error** |

**<u>some properties of the hat matrix:</u>**

- $H = H^T$ (symmetry)
- $HH = H \rightarrow H^n = H$ (idempotency)

<u>evaluating the fit:</u>

$$\hat{\varepsilon} = Y - X\hat{\beta} = Y - \hat{Y} = (I - H)Y$$

$$\hat{\varepsilon}^T\hat{\varepsilon} = [(I - H)Y]^T(I - H)Y = Y^T(I - H)^T(I - H)Y = Y^T(I - H)Y$$

sum of squared errors (SSE)

$$\hat{\sigma}^2 = \frac{\hat{\varepsilon}^T\hat{\varepsilon}}{K - N} \longleftarrow$$

**degrees of freedom**

**variance or** mean of squared errors (MSE)

## Data Fitting and Regression:

| | |
|---|---|
| **y:** | **response** |
| **x:** | **regressors** |
| **β:** | **factors** |
| $\boldsymbol{\beta_0}$**:** | **intercept** |
| **ε:** | **error** |

summary:

the model: $Y = X\beta + \varepsilon$

the fit: $\hat{Y} = X\hat{\beta} = X(X^TX)^{-1}X^TY$

SSE: $\hat{\varepsilon}^T\hat{\varepsilon} = Y^T(I-H)Y$ (after the fit)

variance or MSE: $\hat{\sigma}^2 = \frac{\hat{\varepsilon}^T\hat{\varepsilon}}{K-N}$ (after the fit)

often fit quality is judged by $R^2 := 1 - \frac{\sum_k(\hat{y}_k - y_k)^2}{\sum_k(y_k - \langle y\rangle)^2}$

or adjusted $R^2$ $\bar{R}^2 := R^2 - (1-R^2)\frac{N}{K-N-1}$

and it is said that the fit is good if $R^2$ is close to one....

**...but that is not true...**

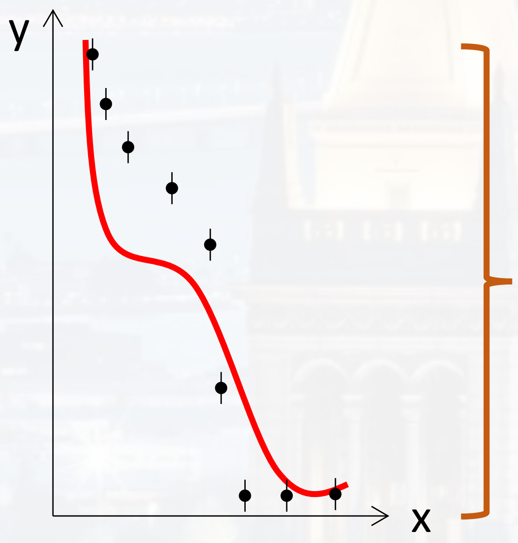
## Data Fitting and Regression:

$$R^2 = 1 - \frac{\boxed{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}}{\boxed{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

variance data vs model
*(aka residual sum of squares)*

variance of the data
*(aka total sum of squares)*

**Note:** do not confuse $R^2$ with Pearsons coefficient: $\rho = \frac{cov(x,y)}{\sqrt{var(x)var(y)}}$



data variance can be huge
(i. e. exponential functions)
→ $R^2$ could be around 1.0
even if fit is completely off!

$\bar{y}$: mean of the data point values

| | |
|---|---|
| $\bar{y}$: | mean of the data point values |
| $y_i$: | measured value of data point |
| $\sigma_i$: | statistical error of $y_i$ (often aka $ey_i$) |
| $\hat{y}_i$: | prediction by the model *after the fit* |
| N : | number of data points |
| p: | number of fit parameter |

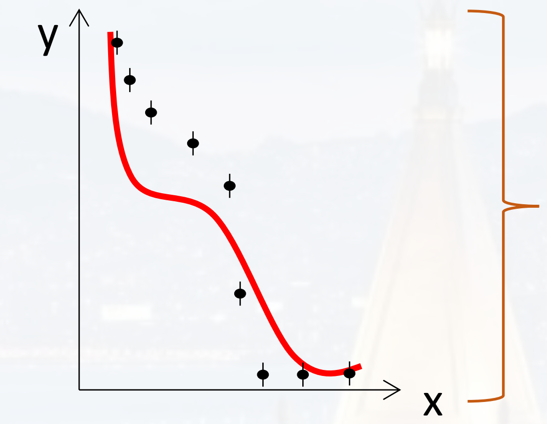
## Data Fitting and Regression:

$$R^2 = 1 - \frac{\boxed{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}}{\boxed{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

variance data vs model
*(aka residual sum of squares)*

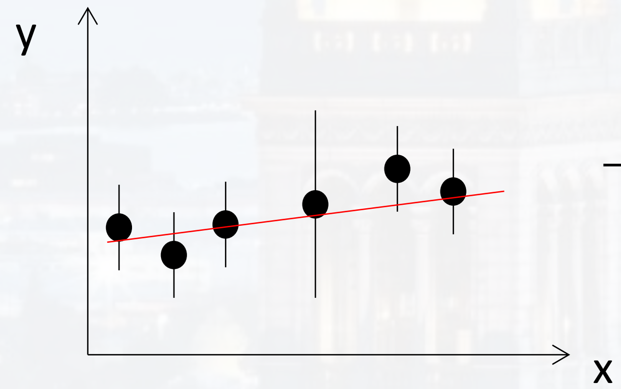
variance of the data
*(aka total sum of squares)*

The Math
What is Linear?
**Stats**
a Python Example



$\bar{y}$: mean of the data point values

**data variance can be huge (i. e. exponential functions) → $R^2$ could be around 1.0 even if fit is completely off!**



$$\frac{\text{variance data vs model } \textit{(aka residual sum of squares)}}{\text{variance of the data } \textit{(aka total sum of squares)}} \approx 1 \rightarrow R^2 = 0$$

**although the fit is good!**

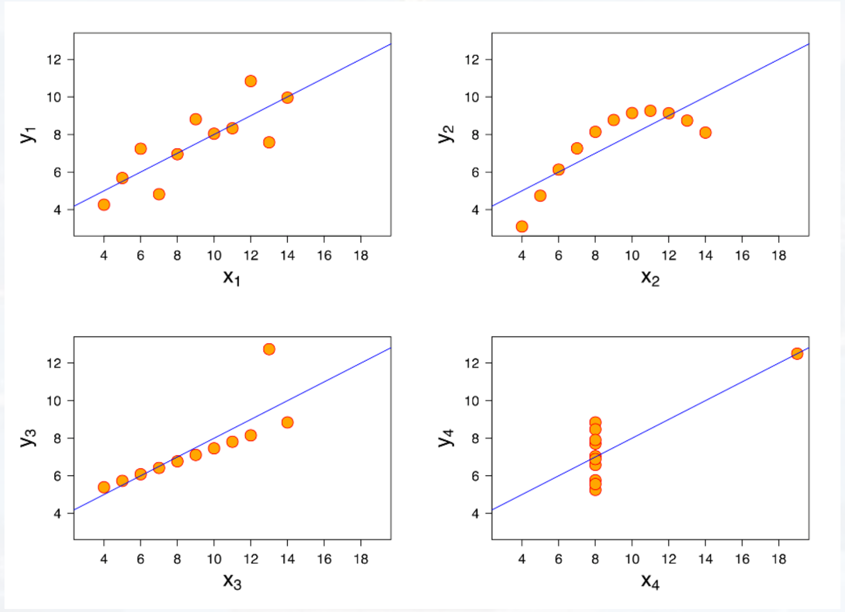
| | |
|---|---|
| **$\bar{y}$:** | **mean of the data point values** |
| **$y_i$:** | **measured value of data point** |
| **$\sigma_i$:** | **statistical error of $y_i$ (often aka $ey_i$)** |
| **$\hat{y}_i$:** | **prediction by the model *after the fit*** |
| **N :** | **number of data points** |
| **p:** | **number of fit parameter** |

## Data Fitting and Regression:

$$R^2 = 1 - \frac{\boxed{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}}{\boxed{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

variance data vs model
*(aka residual sum of squares)*

variance of the data
*(aka total sum of squares)*



all plots: same $R^2$

**conclusion:**

$R^2$ is **not** a measure of the fit quality (but $\chi^2$ is)

**Given a good fit**, $R^2$ tells how strong the dependent variable responds to the independent variable

**Also, Wiki is full of examples…**
**…and warnings (see "caveats" therein)**

## Data Fitting and Regression:

The Math
What is Linear?
Stats
**a Python Example**

see `Walk_Through_LinRegression.ipynb`

```python
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pylab
import scipy.stats as stats
import statsmodels.api as sm

from statsmodels.formula.api import ols
from sklearn.preprocessing import MinMaxScaler
```

reading .xlsx
.csv
.txt
…

standard plots

fancy plots: here a pair-plot

Q-Q plot

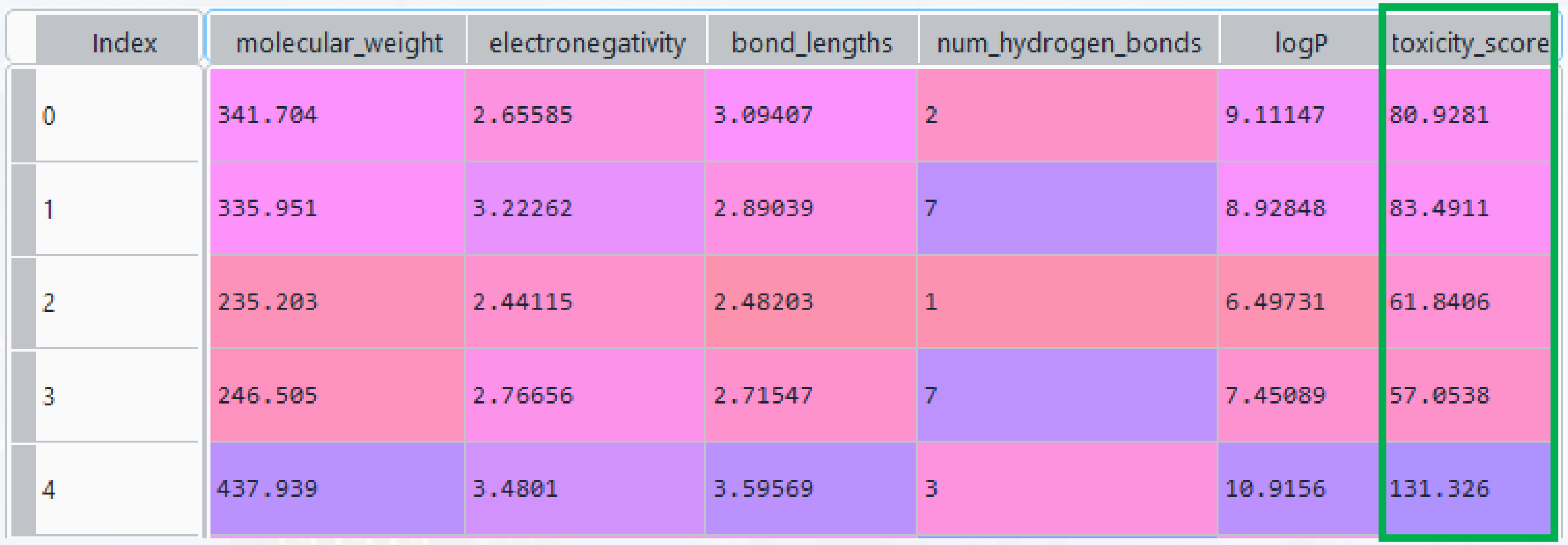
the actual super tool for superb data analysis

scaling and normalizing

## Data Fitting and Regression:

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
Train  = pd.read_csv("molecular_train_gbc.csv")
Test   = pd.read_csv("molecular_test_gbc.csv")
```



| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $y_k$ |
|---|---|---|---|---|---|---|
| Index | molecular_weight | electronegativity | bond_lengths | num_hydrogen_bonds | logP | toxicity_score |
| 0 | 341.704 | 2.65585 | 3.09407 | 2 | 9.11147 | 80.9281 |
| 1 | 335.951 | 3.22262 | 2.89039 | 7 | 8.92848 | 83.4911 |
| 2 | 235.203 | 2.44115 | 2.48203 | 1 | 6.49731 | 61.8406 |
| 3 | 246.505 | 2.76656 | 2.71547 | 7 | 7.45089 | 57.0538 |
| 4 | 437.939 | 3.4801 | 3.59569 | 3 | 10.9156 | 131.326 |

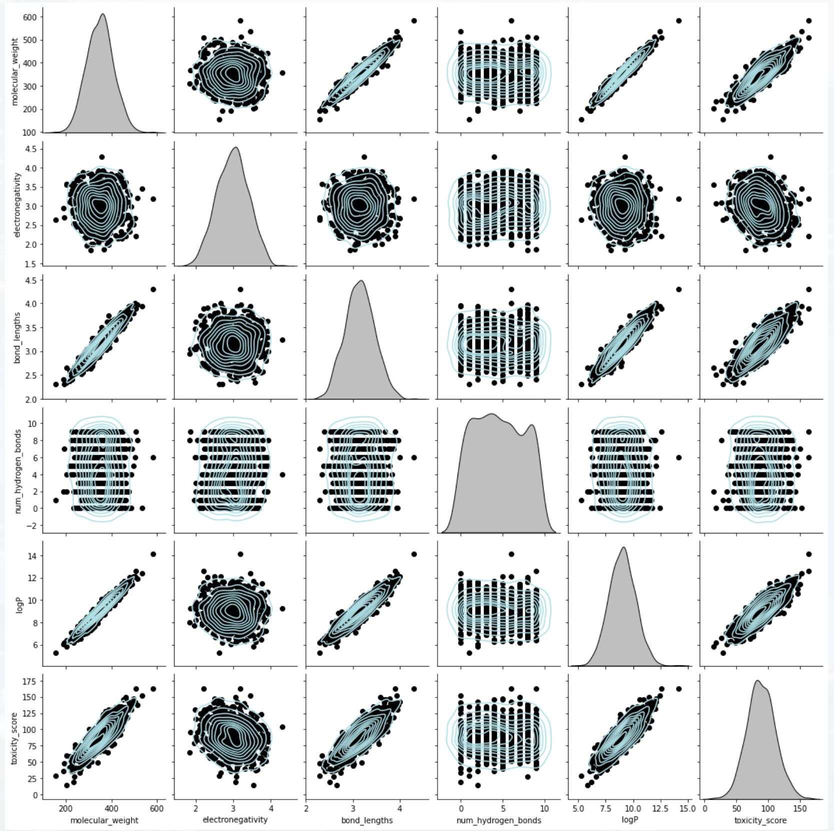
$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

| | |
|---|---|
| y: | toxicity_score |
| $x_n$ : | molecular_weight, electronegativity, bond_lengths, num_hydrogen_bonds, logP |

## Data Fitting and Regression: Linear Regression

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

```
Train  = pd.read_csv("molecular_train_gbc.csv")
Test   = pd.read_csv("molecular_test_gbc.csv")


out = sns.pairplot(Train, kind = "kde", \
                   plot_kws = {'color':[176/255, 224/255, 230/255]},\
                   diag_kws = {'color': 'black'})
out.map_offdiag(plt.scatter, color = 'black')
```



## Data Fitting and Regression:

```
Train  = pd.read_csv("molecular_train_gbc.csv")
Test   = pd.read_csv("molecular_test_gbc.csv")


out = sns.pairplot(Train, kind = "kde", \
                   plot_kws = {'color':[176/255, 224/255, 230/255]},\
                   diag_kws = {'color': 'black'})
out.map_offdiag(plt.scatter, color = 'black')
```

```
scaler  = MinMaxScaler(feature_range = (0, 1))
TrainS  = scaler.fit_transform(Train)
TestS   = scaler.transform(Test)
```

the scaler returns an `np.array`
→ convert back to `data frame`

```
TrainS = pd.DataFrame(TrainS, columns = Train.columns)
TestS  = pd.DataFrame(TestS, columns = Train.columns)
```

## Data Fitting and Regression:

```
TrainS = pd.DataFrame(TrainS, columns = Train.columns)
TestS  = pd.DataFrame(TestS,  columns = Train.columns)
```

```
equation = 'toxicity_score ~ ' + '+'.join(Train.columns[:-1])
print(equation)
```

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

```
toxicity_score ~      molecular_weight + electronegativity +
                      bond_lengths + num_hydrogen_bonds + logP
```

```
my_model = ols(equation, data = TrainS).fit()
my_model.summary()
```

**OLS** (ordinary least squares)

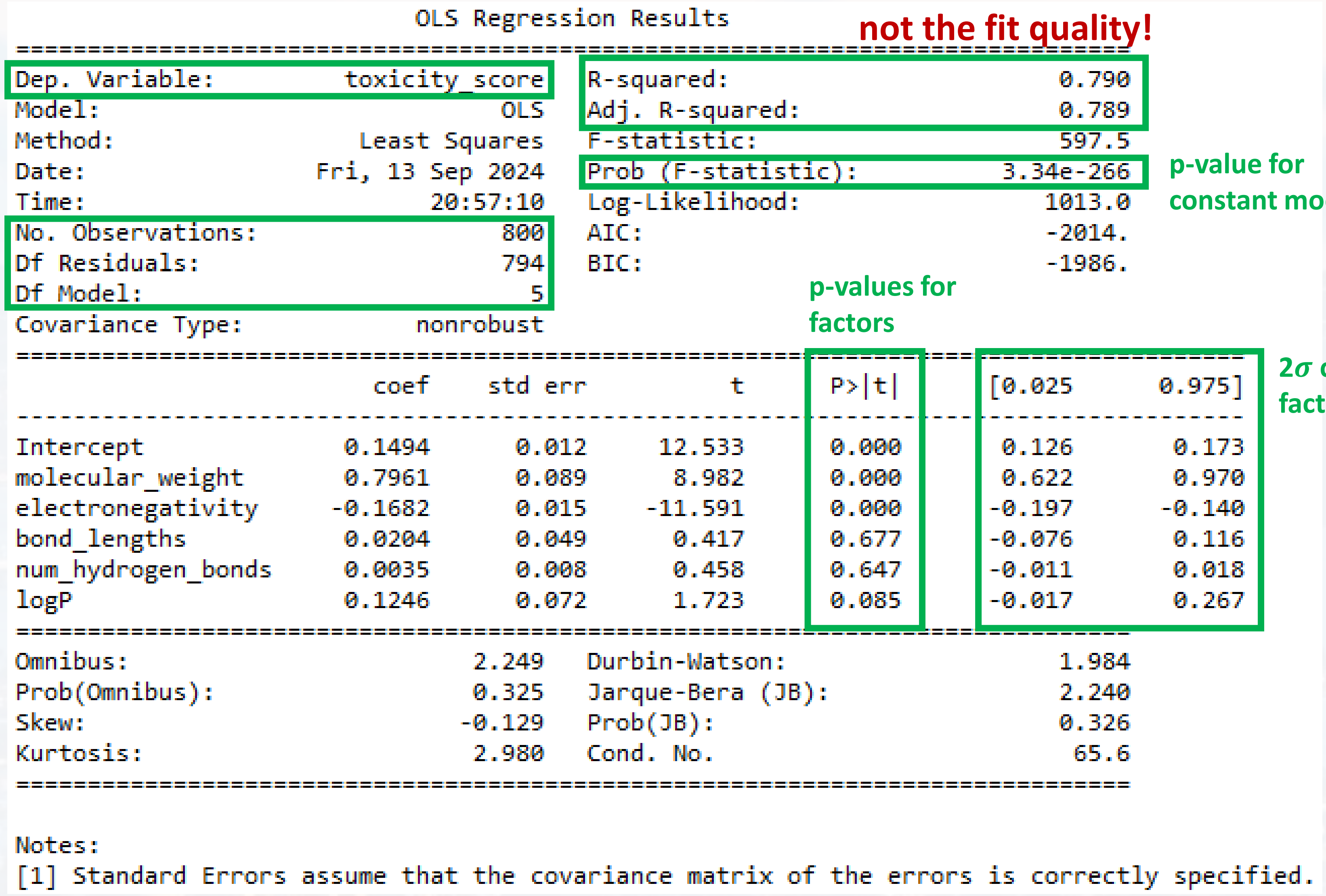
## Data Fitting and Regression:

Linear Regression

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

```
my_model.summary()
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:         toxicity_score   R-squared:                       0.790
Model:                            OLS   Adj. R-squared:                  0.789
Method:                 Least Squares   F-statistic:                     597.5
Date:                Fri, 13 Sep 2024   Prob (F-statistic):          3.34e-266
Time:                        20:57:10   Log-Likelihood:                 1013.0
No. Observations:                 800   AIC:                            -2014.
Df Residuals:                     794   BIC:                            -1986.
Df Model:                           5
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
Intercept             0.1494      0.012     12.533      0.000       0.126       0.173
molecular_weight      0.7961      0.089      8.982      0.000       0.622       0.970
electronegativity    -0.1682      0.015    -11.591      0.000      -0.197      -0.140
bond_lengths          0.0204      0.049      0.417      0.677      -0.076       0.116
num_hydrogen_bonds    0.0035      0.008      0.458      0.647      -0.011       0.018
logP                  0.1246      0.072      1.723      0.085      -0.017       0.267
==============================================================================
Omnibus:                        2.249   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.325   Jarque-Bera (JB):                2.240
Skew:                          -0.129   Prob(JB):                        0.326
Kurtosis:                       2.980   Cond. No.                         65.6
==============================================================================

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

not the fit quality!

p-value for constant model

p-values for factors

$2\sigma$ conf range of factors

number of data points is much larger than the number of regressors
$\rightarrow$ degree of freedom approx. no of obs

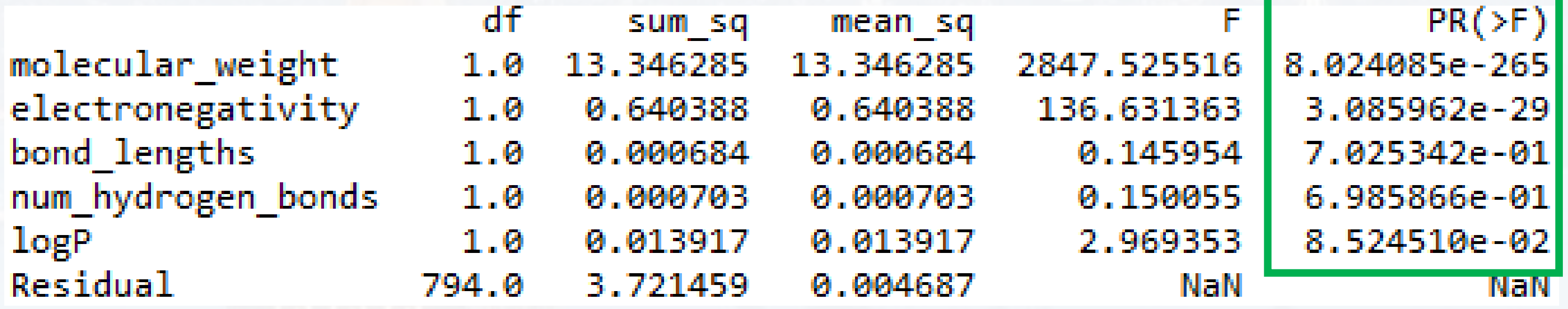
## Data Fitting and Regression:

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

more accurate: determining **the p-values for the factors using ANOVA** for the corresponding residuals

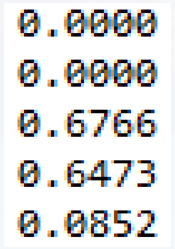
```
table    = sm.stats.anova_lm(my_model, typ = 1)
print(table)
```

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



```
                         df     sum_sq    mean_sq            F         PR(>F)
molecular_weight        1.0  13.346285  13.346285  2847.525516  8.024085e-265
electronegativity       1.0   0.640388   0.640388   136.631363   3.085962e-29
bond_lengths            1.0   0.000684   0.000684     0.145954   7.025342e-01
num_hydrogen_bonds      1.0   0.000703   0.000703     0.150055   6.985866e-01
logP                    1.0   0.013917   0.013917     2.969353   8.524510e-02
Residual              794.0   3.721459   0.004687          NaN            NaN
```

vs from t-test

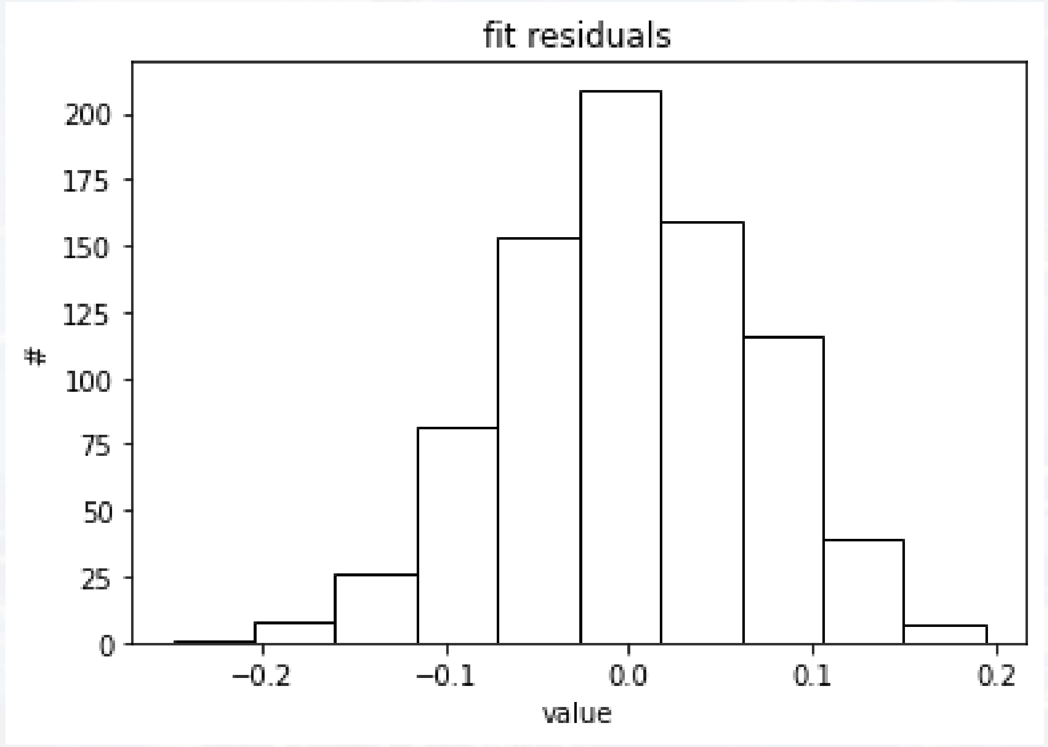


```
0.0000
0.0000
0.6766
0.6473
0.0852
```

## Data Fitting and Regression:

```
residuals = my_model.resid

plt.hist(residuals, color = 'w', edgecolor = 'black')
plt.title('fit residuals')
plt.ylabel('#')
plt.xlabel('value')
plt.show()
```

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



**residuals approx. normally distributed around μ = 0**

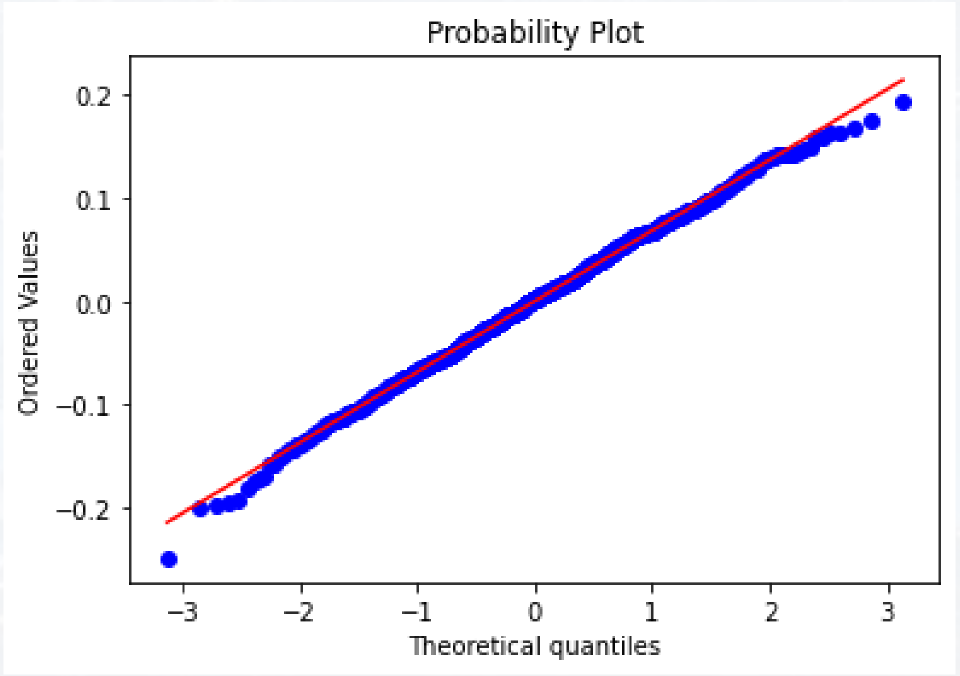
## Data Fitting and Regression:

```
residuals = my_model.resid

plt.hist(residuals, color = 'w', edgecolor = 'black')
plt.title('fit residuals')
plt.ylabel('#')
plt.xlabel('value')
plt.show()

stats.probplot(residuals, dist = "norm", plot = pylab)
pylab.show()
```

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

**residuals approx. normally distributed around μ = 0**

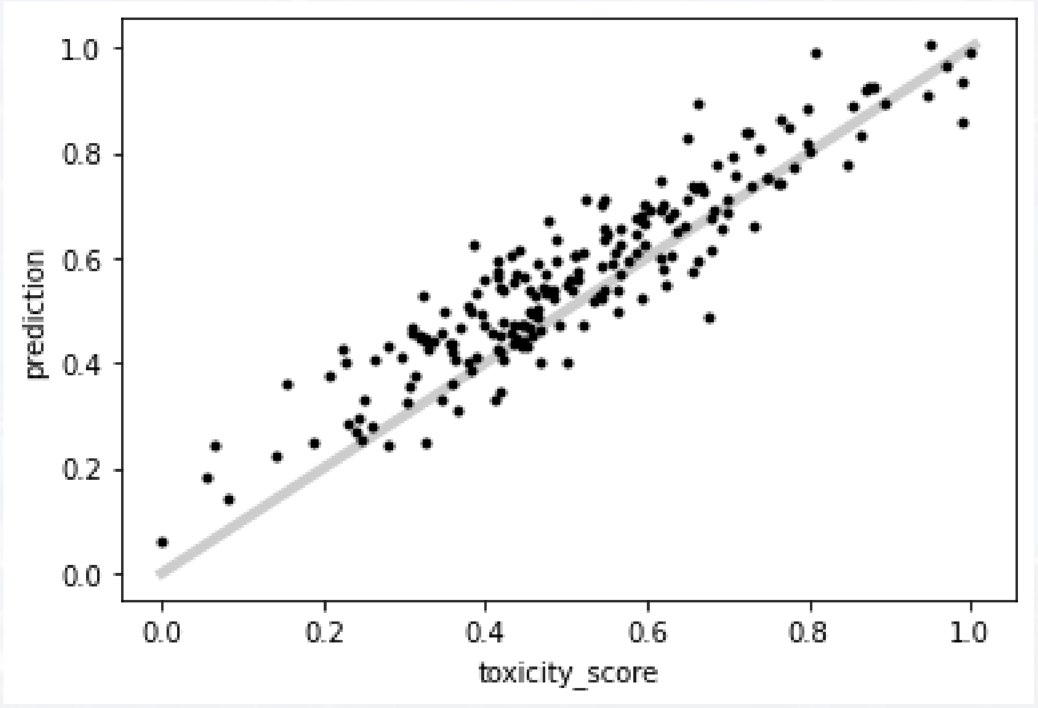


## Data Fitting and Regression:

```
Ypred  = my_model.predict(TestS)

higher = np.max([Ypred, TestS.toxicity_score])
lower  = np.min([Ypred, TestS.toxicity_score])

plt.plot([lower, higher], [lower, higher], c = [0, 0, 0, 0.2],\
         linewidth = 4)
plt.scatter(TestS.toxicity_score, Ypred, marker = '.', c = 'k')
plt.ylabel('prediction')
plt.xlabel('toxicity score')
plt.show()
```

1) loading data
2) plotting data
3) scaling data
4) fitting model
5) evaluating model

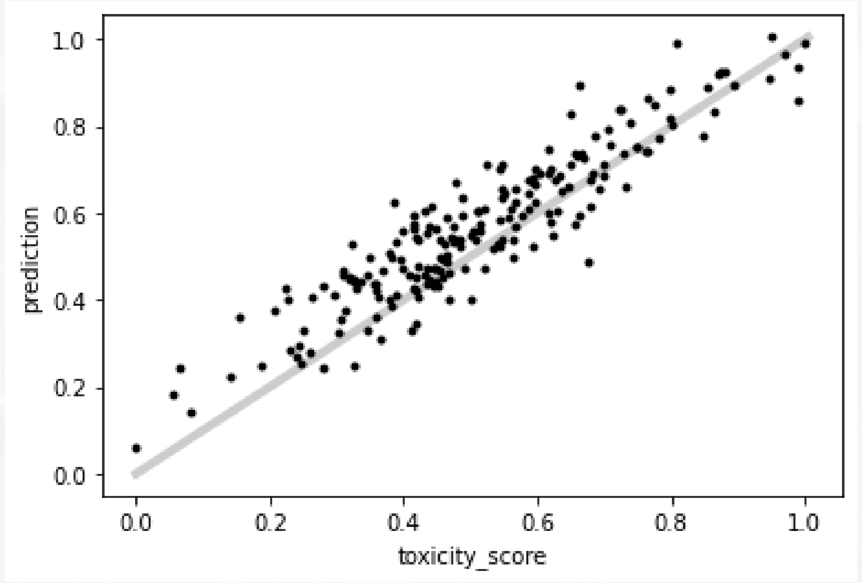
$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



## Data Fitting and Regression:

```
Ypred  = my_model.predict(TestS)

higher = np.max([Ypred, TestS.toxicity_score])
lower  = np.min([Ypred, TestS.toxicity_score])

plt.plot([lower, higher], [lower, higher], c = [0, 0, 0, 0.2], linewidth = 4)
plt.scatter(TestS.toxicity_score, Ypred, marker = '.', c = 'k')
plt.ylabel('prediction')
plt.xlabel('toxicity score')
plt.show()
```

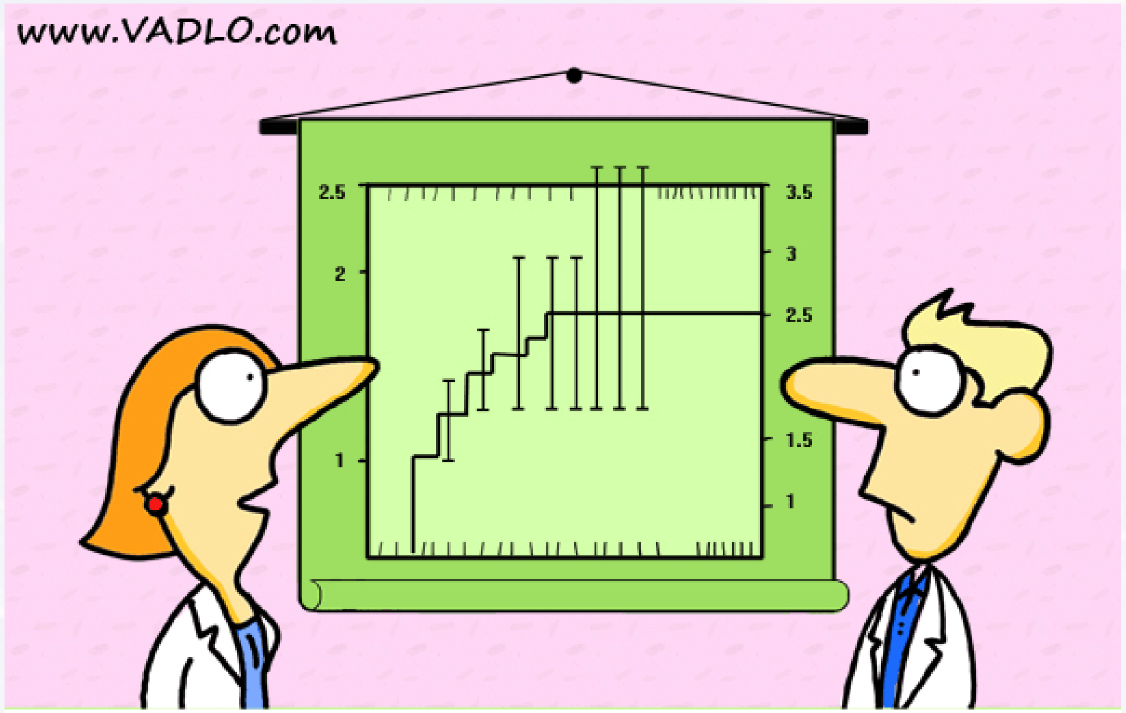
$$y_k = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$



```
mean_dev = np.sum( abs(TestS.toxicity_score - Ypred) )/len(Ypred)
print(mean_dev)
```

```
5%
```

## Outline

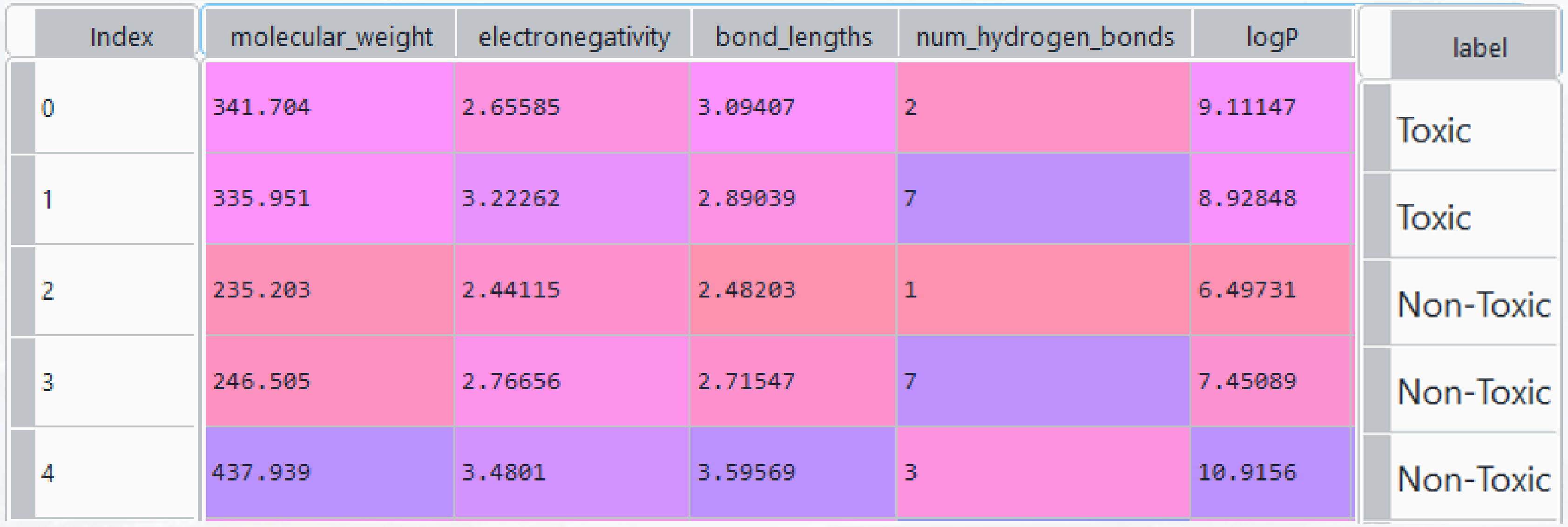


"Did you really have to show the error bars?"

## Data Fitting and Regression:

| | |
|---|---|
| linear model: | regressors are continuous or categorical, response is continuous |
| logistic model: | response is **categorical** |

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |



| Index | molecular_weight | electronegativity | bond_lengths | num_hydrogen_bonds | logP | label |
|---|---|---|---|---|---|---|
| 0 | 341.704 | 2.65585 | 3.09407 | 2 | 9.11147 | Toxic |
| 1 | 335.951 | 3.22262 | 2.89039 | 7 | 8.92848 | Toxic |
| 2 | 235.203 | 2.44115 | 2.48203 | 1 | 6.49731 | Non-Toxic |
| 3 | 246.505 | 2.76656 | 2.71547 | 7 | 7.45089 | Non-Toxic |
| 4 | 437.939 | 3.4801 | 3.59569 | 3 | 10.9156 | Non-Toxic |

## Data Fitting and Regression:

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |

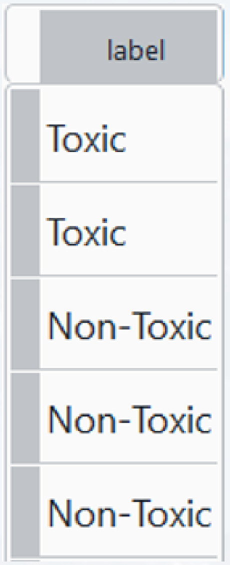
<u>linear model:</u> regressors are continuous or categorical, response is continuous

<u>logistic model:</u> response is **categorical**

dichotomic model:

**probability** to be in state A) → ***p***

**probability** to be in state B) → ***1 - p***



| label |
|---|
| Toxic |
| Toxic |
| Non-Toxic |
| Non-Toxic |
| Non-Toxic |

<u>ansatz:</u>

$$\boxed{log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon}$$

**log odds ratio: linear model**

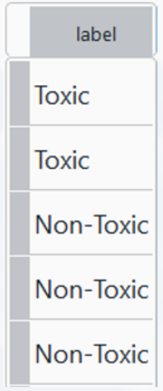
## Data Fitting and Regression:

dichotomic model:

**probability** to be in state A) → ***p***

**probability** to be in state B) → ***1 - p***

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |



| label |
|---|
| Toxic |
| Toxic |
| Non-Toxic |
| Non-Toxic |
| Non-Toxic |

ansatz:

$$\boxed{log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon}$$

**log odds ratio: linear model**

→ probability for being in a certain state

$$\boxed{p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}}}$$

often:

$$\boxed{logit(p) = log\left(\frac{p}{1-p}\right)}$$

examples:

- probability that a gene has been mutated
- probability of being diseased (cancer, alzheimer etc) as function of age, environmental influence etc ...
- Verhulst equation: $N(t) = N_0 \frac{e^{rt}}{C+e^{rt}}$
- activation functions in ANNs

# Data Fitting and Regression:

$$log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon$$

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |

**Note: one can derive the logit function from max. entropy too!**

$$\boxed{p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}} = \frac{1}{1+e^{-\beta_0-\beta_1 x_1-\cdots}}}$$

onset of Alzheimer's disease (AD) is a function of ***age*** and years spent in ***education***
(and other risk factors we ignore here for the sake of simplicity)

education: $\mathrm{d} = x_1$ [yrs]
age: $a = x_2$ [yrs]

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

+ data set + fit → $\beta_0 = +0.1$
$\beta_1 = -1.5$
$\beta_2 = +0.12$

- *positive* value → *increasing p*
- *negative* value → *decreasing p*
- intercept: "background" prevalence, not related to environmental/internal conditions

## Data Fitting and Regression:

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |

education: $d = x_1$ [yrs]

age: $a = x_2$ [yrs]

$\beta_0 = +0.1$

$\beta_1 = -1.5$

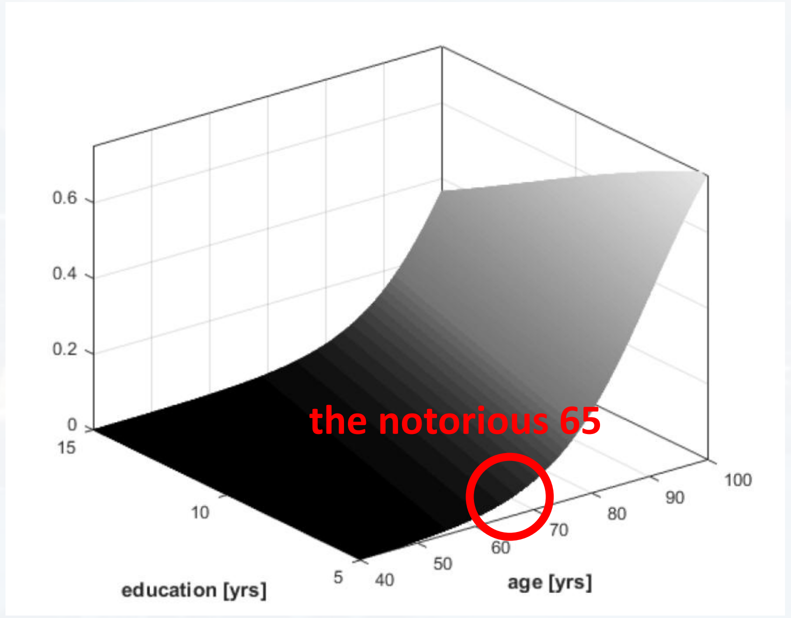
$\beta_2 = +0.12$

example: ***65yrs*** old person, ***8yrs*** spent in education

→ $p_{AD} = 1.6\%$

***65yrs*** old person, ***13yrs*** spent in education

→ $p_{AD} = 0.001\%$



<u>How does education compensate aging?</u>

$$p_{AD}(d+\bar{d}, a+\bar{a}) = p_{AD}(d,a)$$

→ $\bar{a} = 12.5\,\bar{d}$

**hence, one more year prolonged education compensates 12.5 years of aging**

**(warning: don't confuse correlation with causation here!)**

## Data Fitting and Regression:

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |

model: $p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$

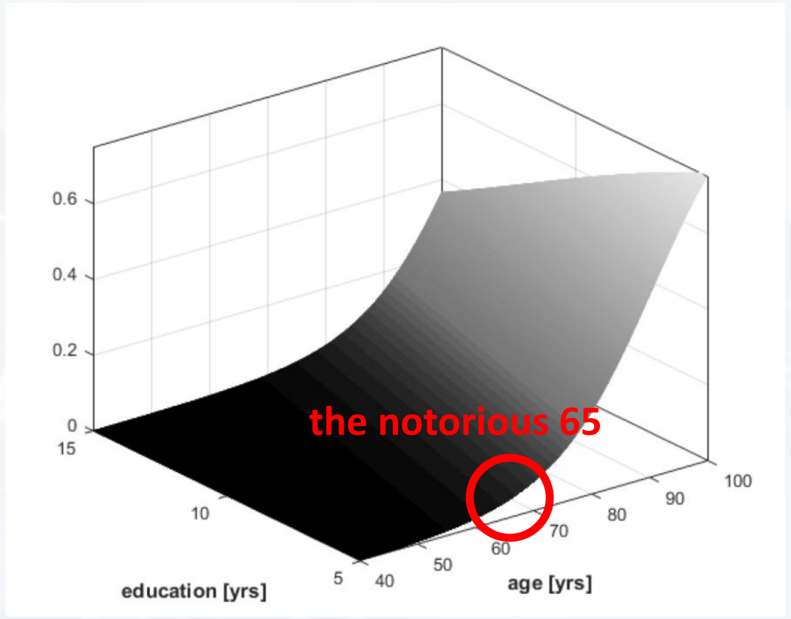
education: $\mathrm{d} = x_1$ [yrs]

age: $a = x_2$ [yrs]

$\beta_0 = +0.1$

$\beta_1 = -1.5$

$\beta_2 = +0.12$



How does the risk of onset changes *per year*?

relative change:

$$\frac{p_{AD}(a+1) - p_{AD}(a)}{p_{AD}(a)} \approx e^{\beta_2} - 1 \approx 12.7\%$$

**$p_{AD} \ll 1$ (hence, for small $\Delta a$ and "young" ages, i. e. below $\approx$ 80yrs )**

**the risk of getting AD increases by 12.7% every year**

**(warning: does not mean that it increases by 127% in ten yrs – we made an approximation!)**

## Data Fitting and Regression:

| | |
|---|---|
| y: | response |
| x: | regressors **(assumed to be independent)** |
| β: | factors |
| $\beta_0$: | intercept |
| ε: | error **(stochasticity of the data, assumed to be normally dist.)** |

model: $$p_{AD} = \frac{1}{1+e^{-\beta_0-\beta_1 d-\beta_2 a}}$$

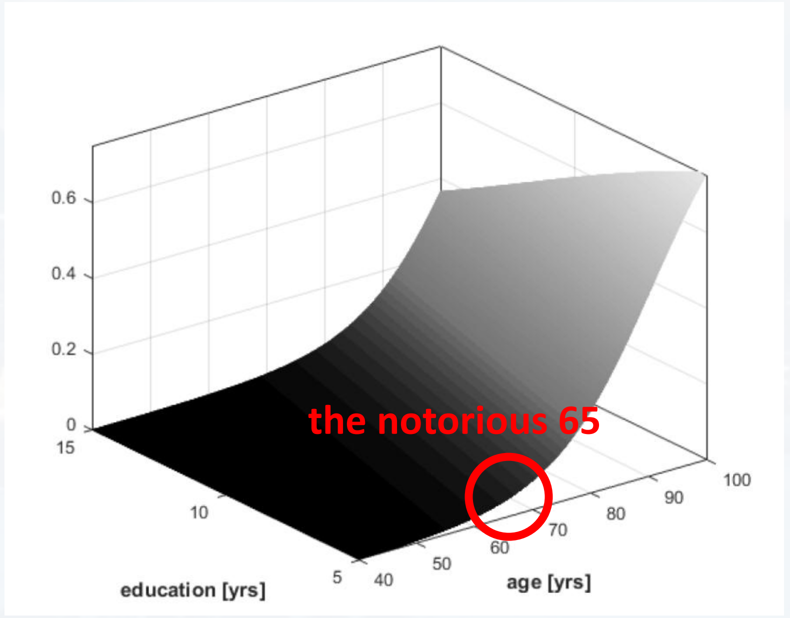
education: $d = x_1$ [yrs]

age: $a = x_2$ [yrs]

$\beta_0 = +0.1$

$\beta_1 = -1.5$

$\beta_2 = +0.12$



How does the risk of onset changes *per year*?

more precise: relative change of the odds ratio

$$\frac{\frac{\partial}{\partial x_i}\left(\frac{p_{AD}}{1-p_{AD}}\right)}{\frac{p_{AD}}{1-p_{AD}}} = \beta_i$$

$x_i$ is the desired regressor, for example, age again ($x_2$)

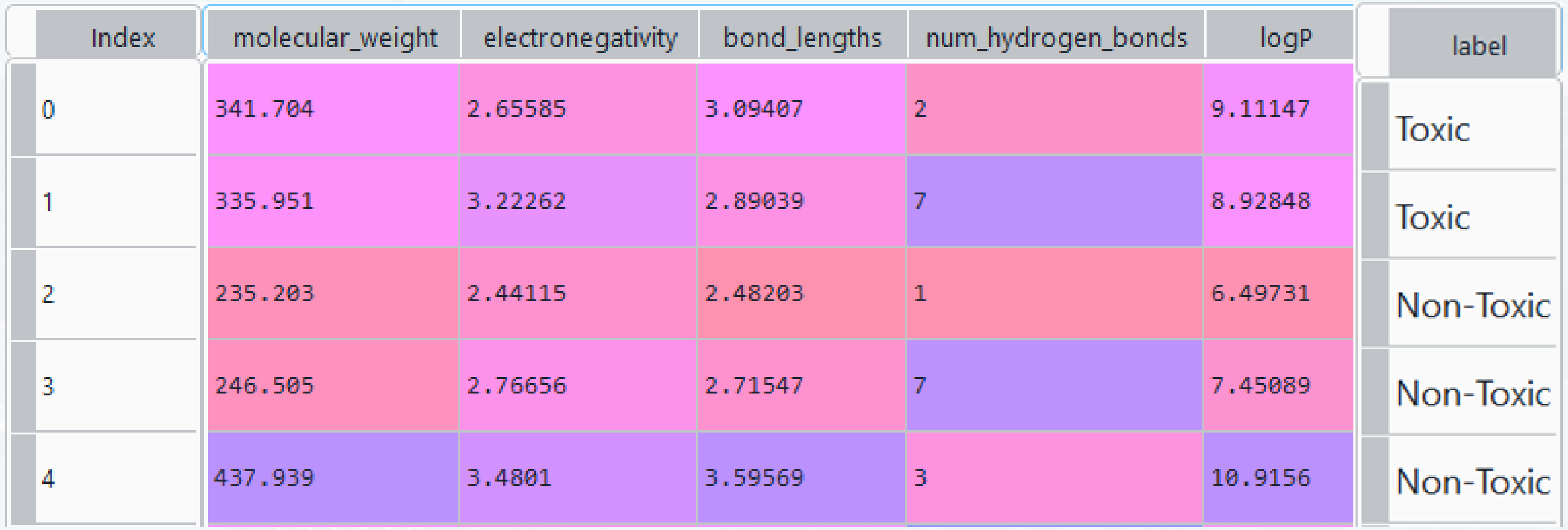
**the factors $\boldsymbol{\beta_i}$ indicate how strong (and in which direction) *p* changes wrt a regressor $\boldsymbol{x_i}$**

## Data Fitting and Regression:

Logistic Regression

let us return to the molecule data set:

```
Train  = pd.read_csv("molecular_train_gbc_cat.csv")
Test   = pd.read_csv("molecular_test_gbc_cat.csv")
```



| Index | molecular_weight | electronegativity | bond_lengths | num_hydrogen_bonds | logP | label |
|---|---|---|---|---|---|---|
| 0 | 341.704 | 2.65585 | 3.09407 | 2 | 9.11147 | Toxic |
| 1 | 335.951 | 3.22262 | 2.89039 | 7 | 8.92848 | Toxic |
| 2 | 235.203 | 2.44115 | 2.48203 | 1 | 6.49731 | Non-Toxic |
| 3 | 246.505 | 2.76656 | 2.71547 | 7 | 7.45089 | Non-Toxic |
| 4 | 437.939 | 3.4801 | 3.59569 | 3 | 10.9156 | Non-Toxic |

```
import statsmodels.api as sm
```

## Data Fitting and Regression: Logistic Regression

it is the same data set → plotting and scaling is as before

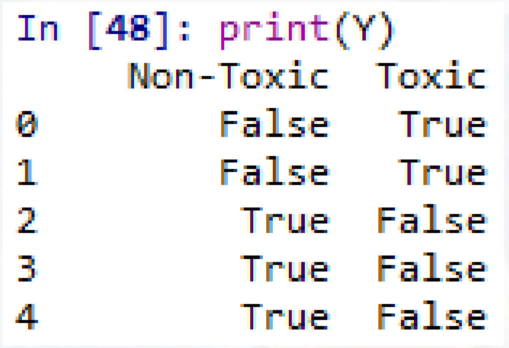
```
X = sm.add_constant(TrainS)
```

adding the intercept

```
Y = pd.get_dummies(Train['label'])
```

Python needs True/False as categorical

$$p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}}$$



```
In [48]: print(Y)
     Non-Toxic  Toxic
0        False   True
1        False   True
2         True  False
3         True  False
4         True  False
```

we have two states: toxic / non-toxic

```
my_model = sm.GLM(Y, X, family = sm.families.Binomial()).fit()

my_model.summary()
```

GLM: general linear model

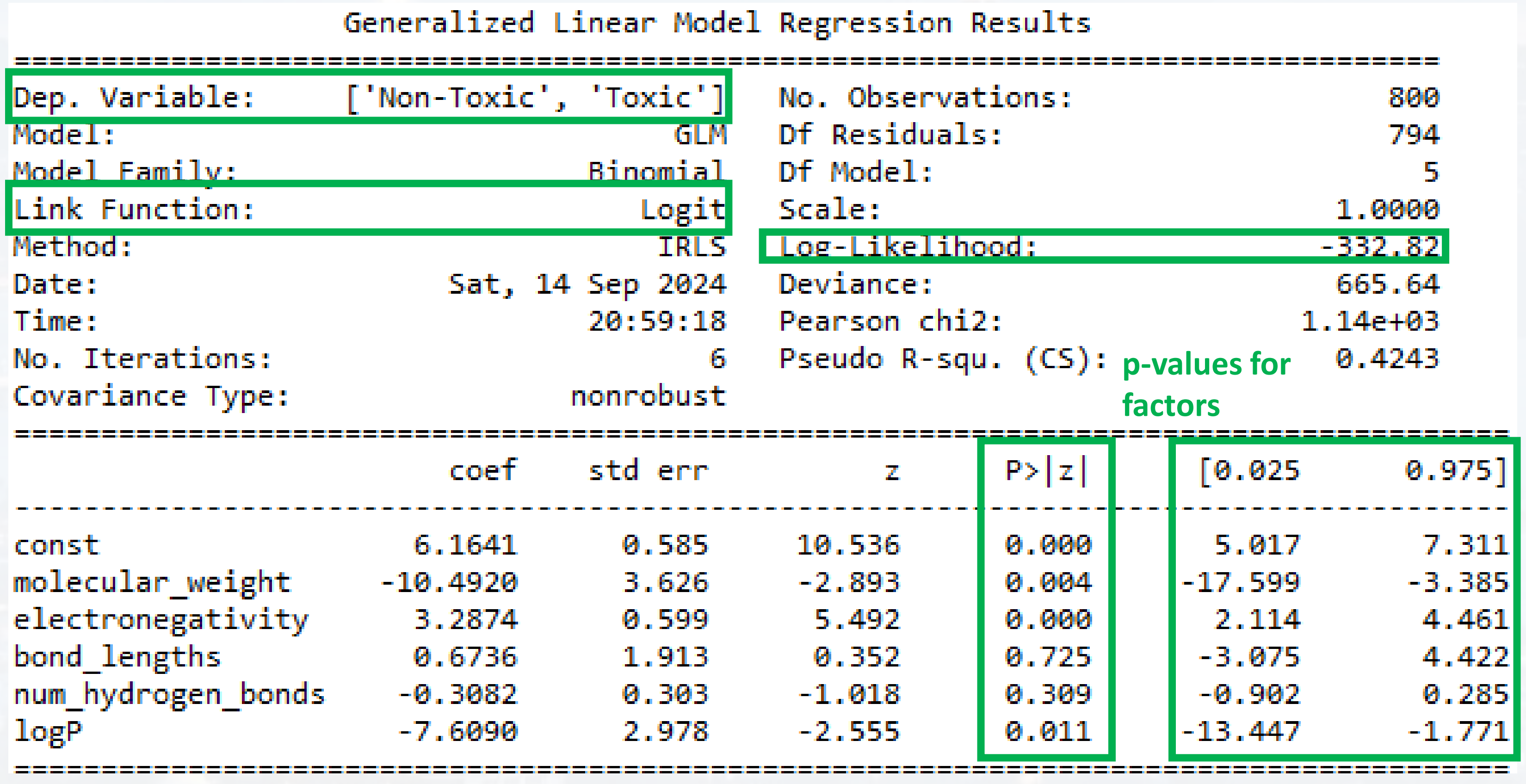
## Data Fitting and Regression:

Logistic Regression

it is the same data set → plotting and scaling is as before

```
X = sm.add_constant(TrainS)
Y = pd.get_dummies(Train['label'])

my_model = sm.GLM(Y, X, family = sm.families.Binomial()).fit()
my_model.summary()
```

$$p = \frac{e^{\beta_0+\beta_1 x_1+\cdots}}{1+e^{\beta_0+\beta_1 x_1+\cdots}}$$



```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:      ['Non-Toxic', 'Toxic']   No. Observations:              800
Model:                                 GLM   Df Residuals:                  794
Model Family:                     Binomial   Df Model:                        5
Link Function:                       Logit   Scale:                      1.0000
Method:                               IRLS   Log-Likelihood:            -332.82
Date:                     Sat, 14 Sep 2024   Deviance:                   665.64
Time:                             20:59:18   Pearson chi2:             1.14e+03
No. Iterations:                          6   Pseudo R-squ. (CS):         0.4243
Covariance Type:                 nonrobust
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 6.1641      0.585     10.536      0.000       5.017       7.311
molecular_weight    -10.4920      3.626     -2.893      0.004     -17.599      -3.385
electronegativity     3.2874      0.599      5.492      0.000       2.114       4.461
bond_lengths          0.6736      1.913      0.352      0.725      -3.075       4.422
num_hydrogen_bonds   -0.3082      0.303     -1.018      0.309      -0.902       0.285
logP                 -7.6090      2.978     -2.555      0.011     -13.447      -1.771
=====================================================================================
```

p-value for constant model

p-values for factors

$2\sigma$ conf range of factors

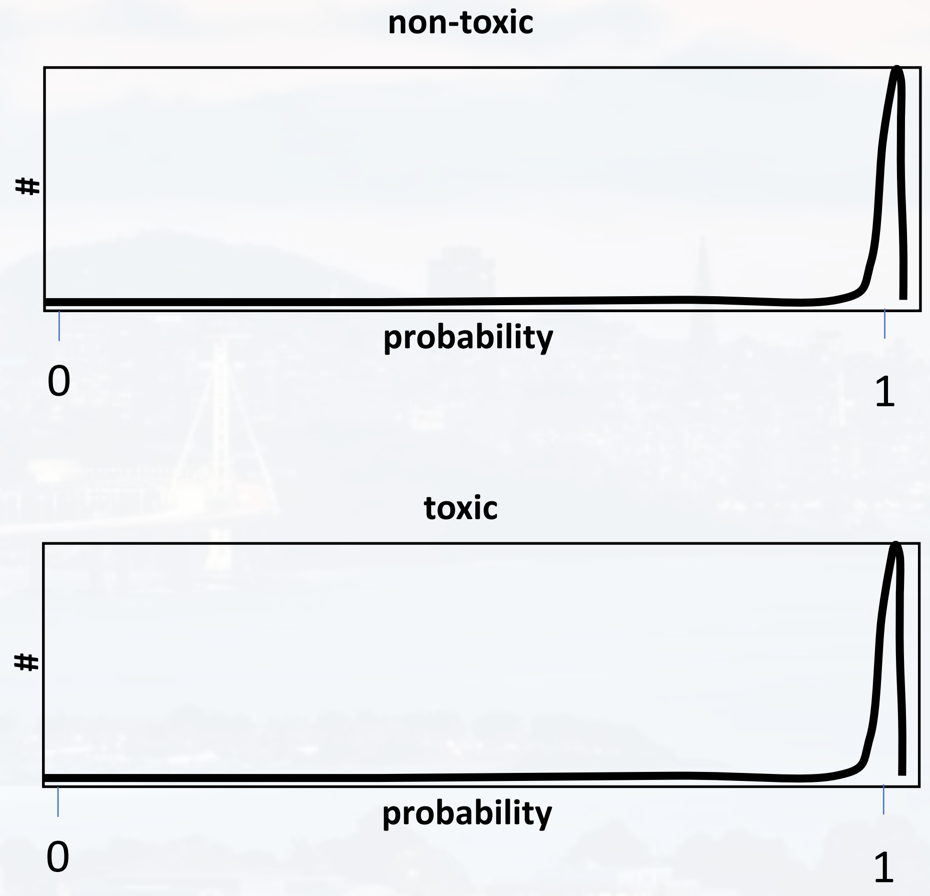
## Data Fitting and Regression:

| | |
|---|---|
| accuracy: | How ***often*** did the model make the correct prediction. |
| cross-entropy: | How ***certain*** was the model when making the prediction. |

<u>ideal world:</u>

| true label | non-toxic | toxic |
|---|---|---|
| **non-toxic** | **100%** | **0%** |
| **toxic** | **0%** | **100%** |

predicted label

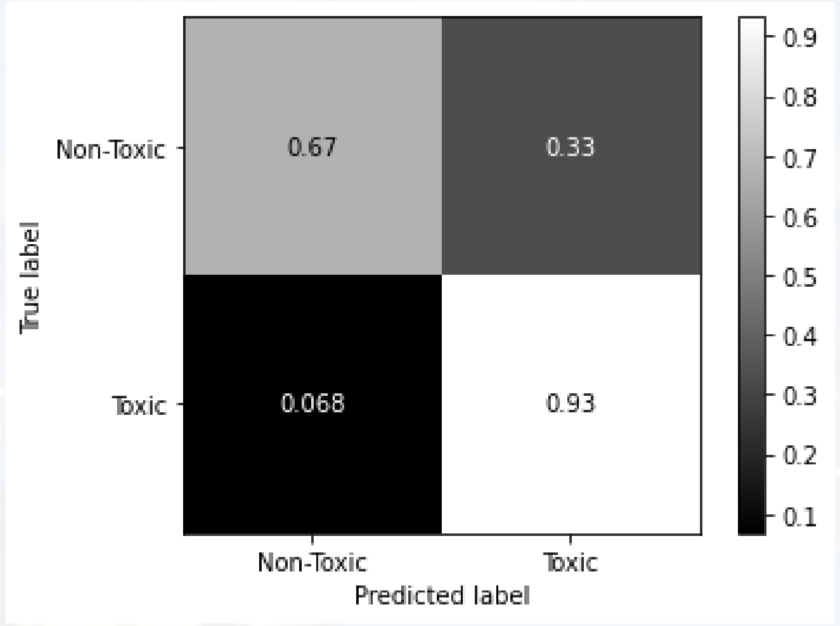


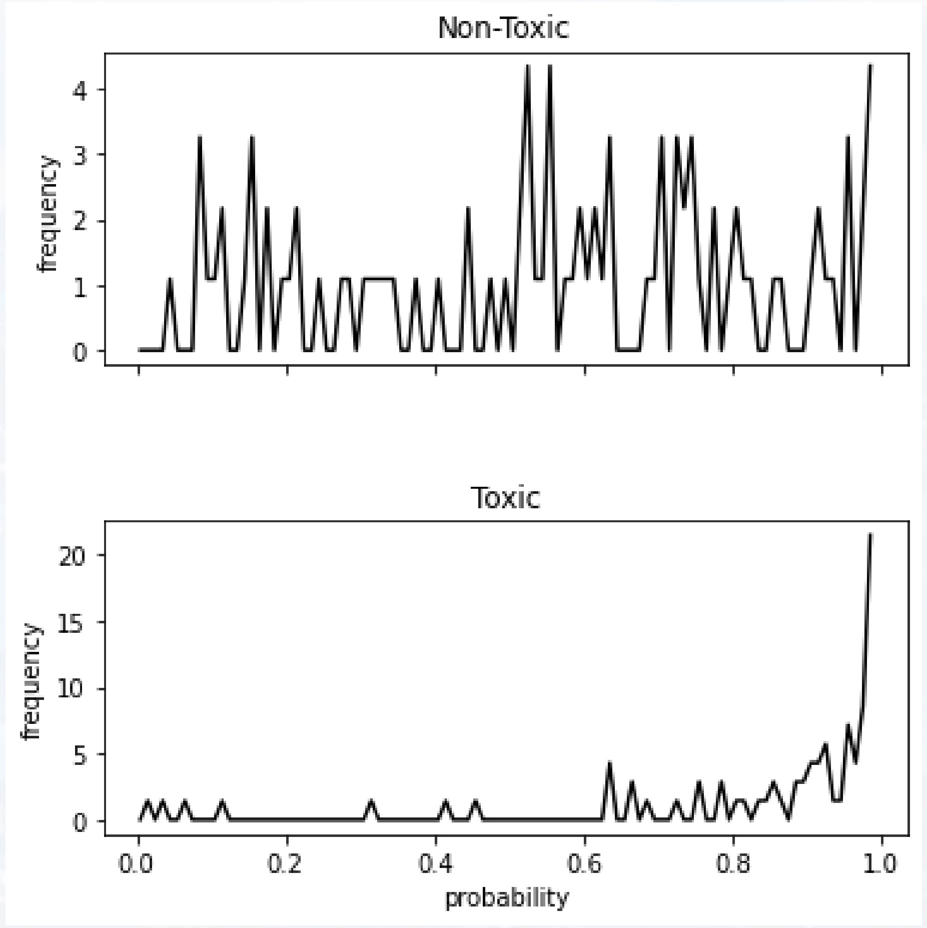
more details see **Chem 277**

## Data Fitting and Regression:

| | |
|---|---|
| accuracy: | How ***often*** did the model make the correct prediction. |
| cross-entropy: | How ***certain*** was the model when making the prediction. |

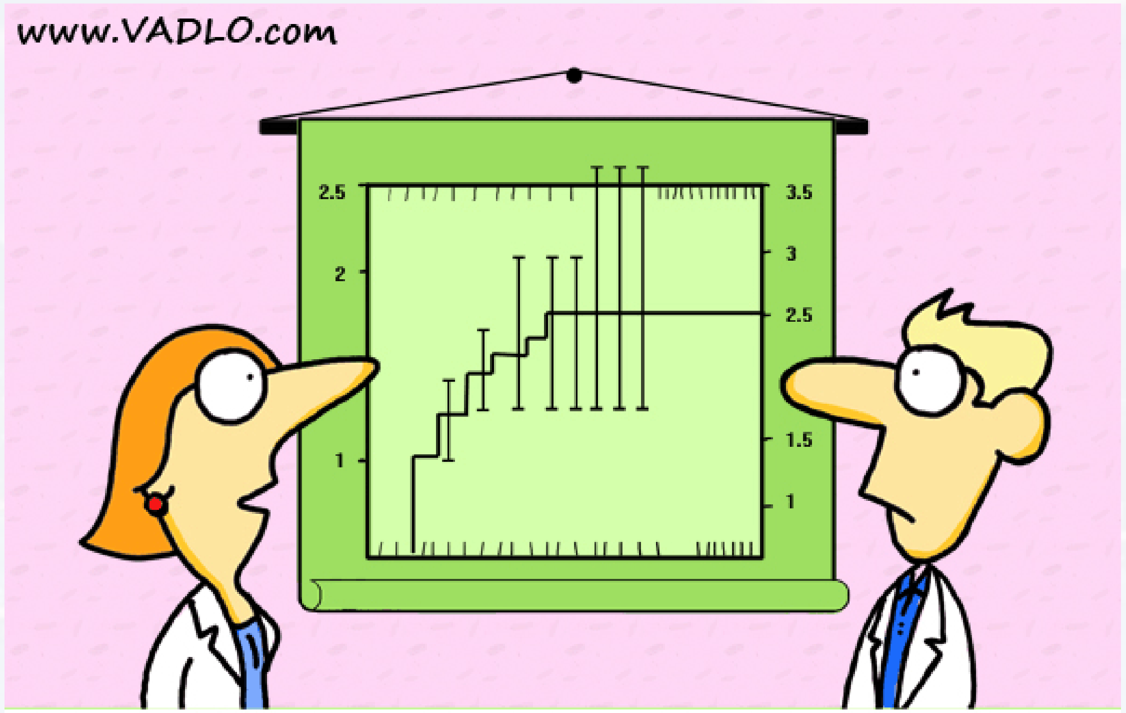
<u>real world:</u>





more details see **Chem 277**

## Outline

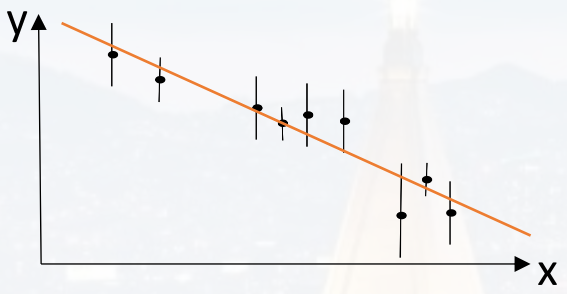


"Did you really have to show the error bars?"

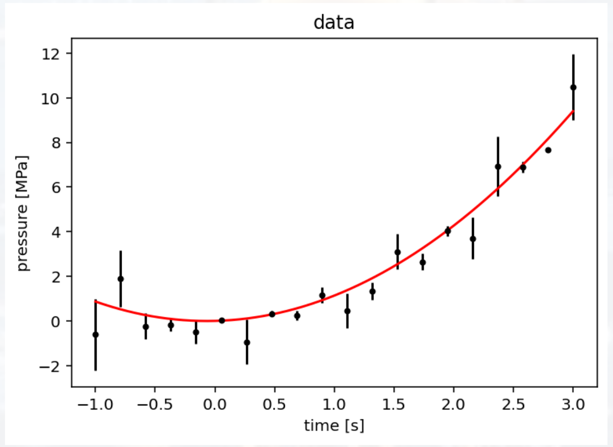
## Data Fitting and Regression:

**problem:** set of data points $y_i = f(x_i)$, ideally each data point $y_i$ has an error $\sigma_i$
we know $f(x_i)$

**goal:** find parameters of $f(x_i)$
once we know the parameters → prediction



model: $f(x_i) = m\,x_i + n$
parameter: $m, n$

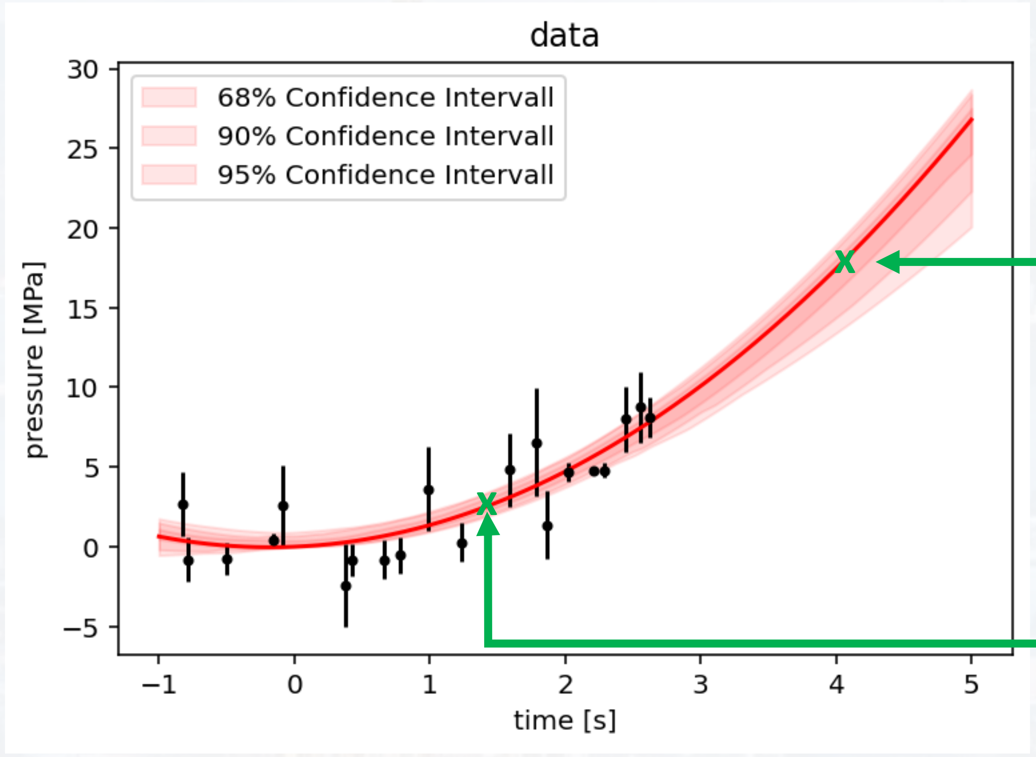


model: $f(x_i) = a\,x_i^2 + b\,x_i + c$
parameter: a, b, c

## Data Fitting and Regression:

**problem:** set of data points $y_i = f(x_i)$, ideally each data point $y_i$ has an error $\sigma_i$
we know $f(x_i)$

**goal:** find parameters of $f(x_i)$
once we know the parameters → **prediction**



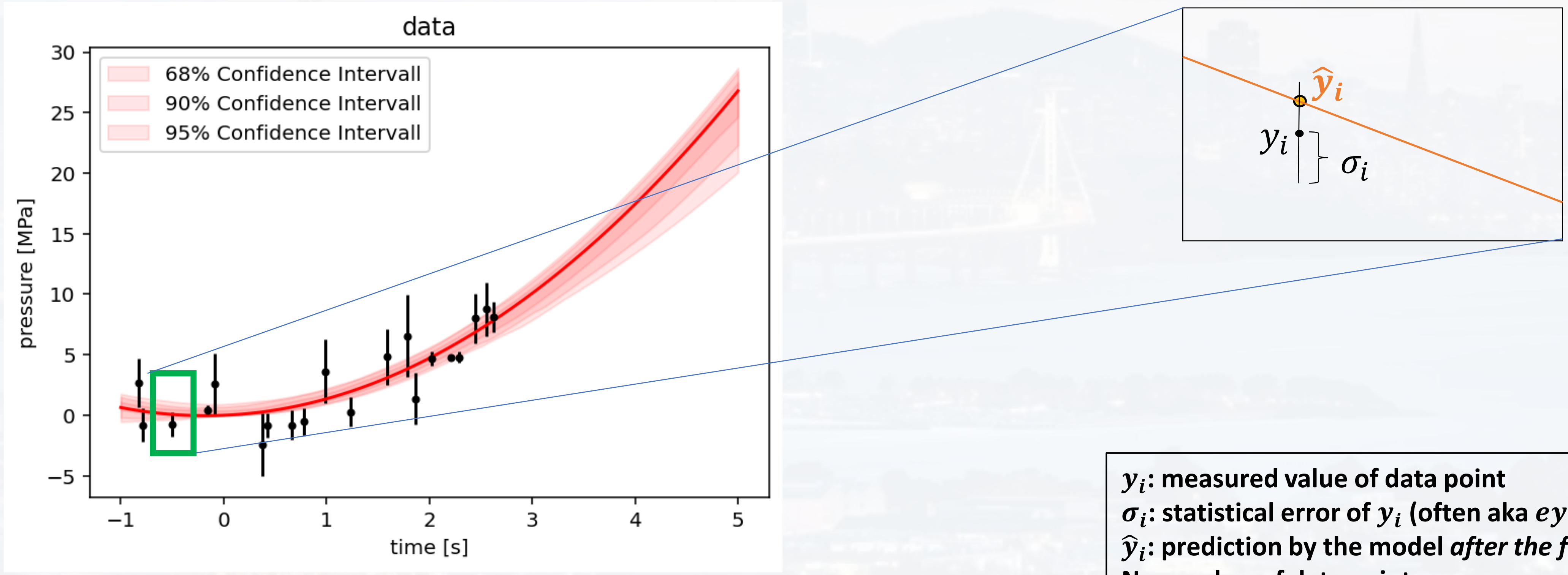
$y_i$ for data point ***outside*** known interval

$y_i$ for data point ***within*** known interval

## Data Fitting and Regression:

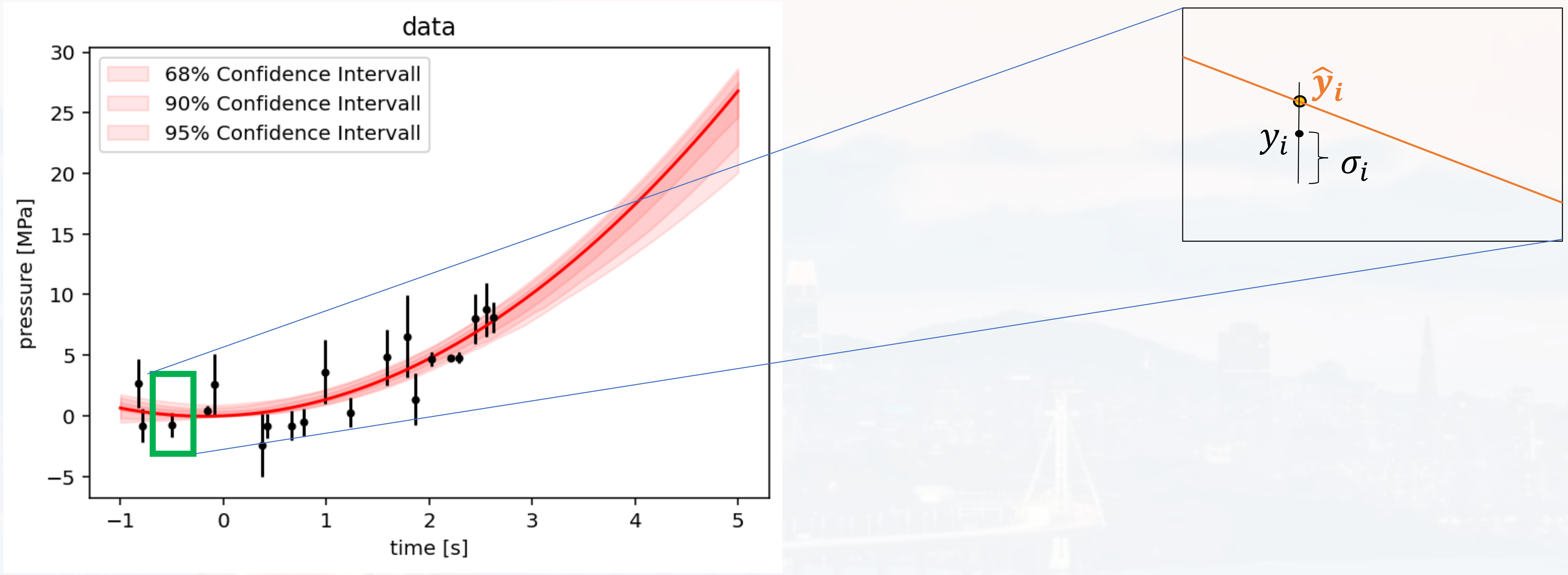
**problem:** set of data points $y_i = f(x_i)$, ideally each data point $y_i$ has an error $\sigma_i$
we know $f(x_i)$

**goal:** find parameters of $f(x_i)$
once we know the parameters → prediction



$y_i$: measured value of data point
$\sigma_i$: statistical error of $y_i$ (often aka $ey_i$)
$\hat{y}_i$: prediction by the model *after the fit*
N : number of data points
p: number of fit parameter

## Data Fitting and Regression:



finding best parameters by minimizing $$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2$$ $df = N - p - 1$

see module 8

or $$MSE = \frac{1}{df}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$ if no errors given

**$y_i$: measured value of data point**
**$\sigma_i$: statistical error of $y_i$ (often aka $ey_i$)**
**$\hat{y}_i$: prediction by the model *after the fit***
**N : number of data points**
**p: number of fit parameter**

## Data Fitting and Regression:

finding best parameters by minimizing

$$\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2$$

$df = N - p - 1$

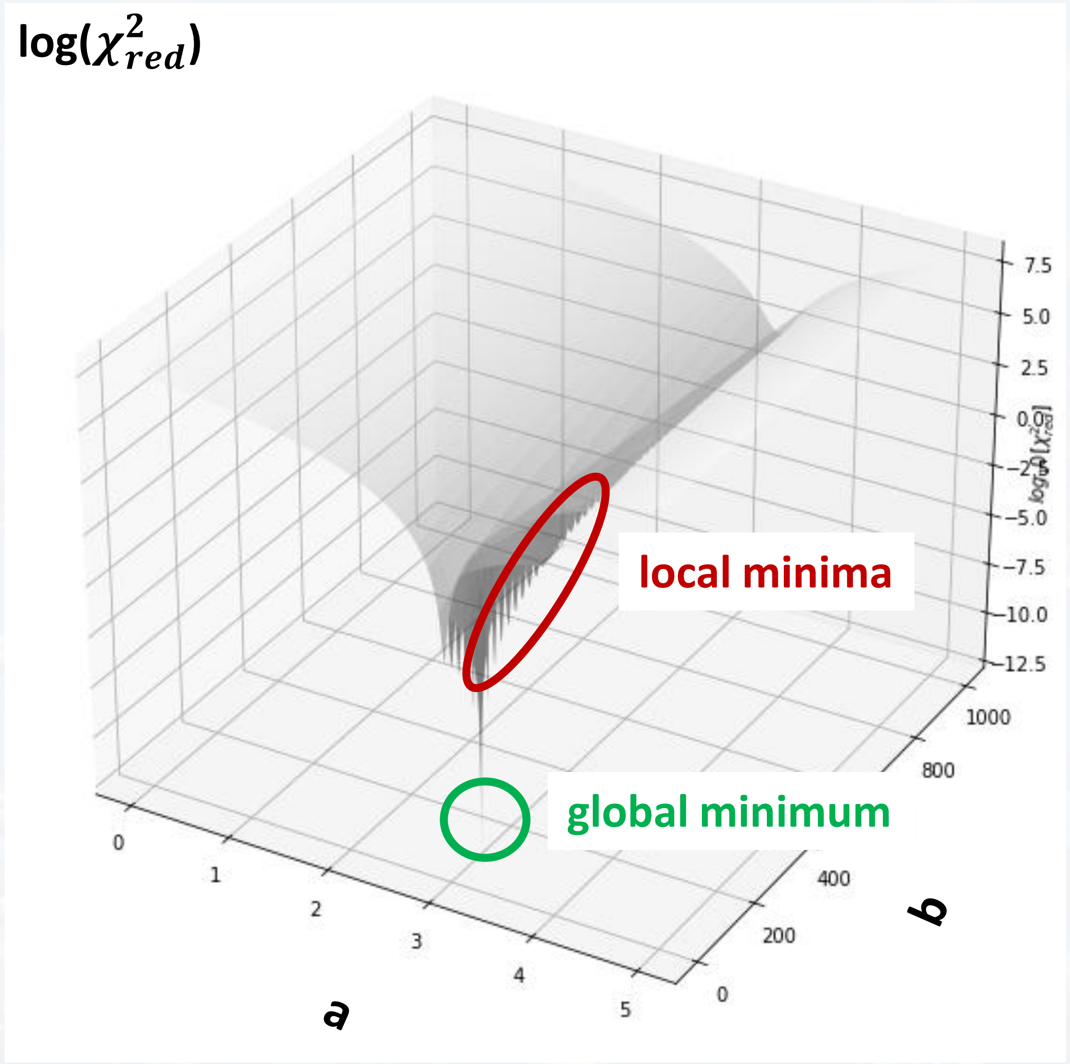
see module 8

or

$$MSE = \frac{1}{df}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

if no errors given

$\log(\chi^2_{red})$

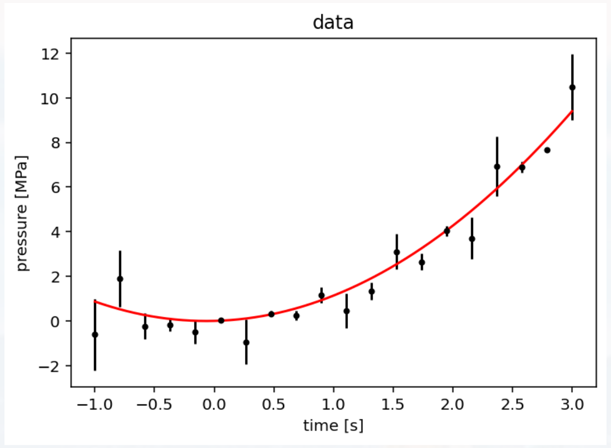


finding the global minimum of a higher dimensional non-analytical function

→ see also module 3

## Data Fitting and Regression:

from module 8



$$\boxed{\chi^2_{red} = \frac{1}{df}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{\sigma_i}\right)^2}$$

$$df = N - p - 1$$

**N : number of data points**
**p: number of fit parameter (model)**

given a fitted model:
$\chi^2_{red}$ is a measure of the fit quality

for large **(> 50…100)** N **(number of data points)**:

≈ **2/3** of the data points should be consistent with the model within their **1σ** error bars

≈ **95%** of the data points should be consistent with the model within their **2σ** error bars

≈ **99.7%** of the data points should be consistent with the model within their **3σ** error bars

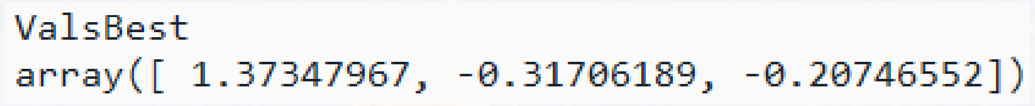
$\chi^2_{red} \approx$

1.0 excellent fit
1.0…1.5 acceptable fit
1.5…1.7 bad fit
>2.0 not acceptable

<<1.0 suspicious, errors are overestimated!

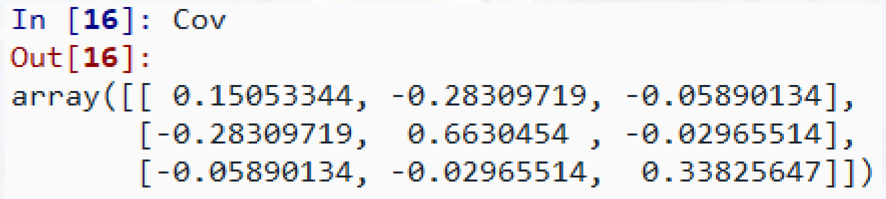
## Data Fitting and Regression: Curve Fitting

```python
from scipy.optimize import curve_fit


def fun_to_fit(x, a, b, c):
        return a*x**2 + b*x + c


ValsBest, Cov = curve_fit(fun_to_fit, x, y)
```



```
ValsBest
array([ 1.37347967, -0.31706189, -0.20746552])
```

a b c



```
In [16]: Cov
Out[16]:
array([[ 0.15053344, -0.28309719, -0.05890134],
       [-0.28309719,  0.6630454 , -0.02965514],
       [-0.05890134, -0.02965514,  0.33825647]])
```

$$\begin{pmatrix} \sigma_a^2 & cov(a,b) & cov(a,c) \\ cov(b,a) & \sigma_b^2 & cov(b,c) \\ cov(c,a) & cov(c,b) & \sigma_c^2 \end{pmatrix}$$

## Data Fitting and Regression:

```
from scipy.optimize import curve_fit
```

scipy has the optimization tool cuve_fit

```
def fun_to_fit(x, a, b, c):
        return a*x**2 + b*x + c
```

defining the model as a python function

```
ValsBest, Cov = curve_fit(fun_to_fit, x, y)
```

run the curve fit

```
ValsBest
array([ 1.37347967, -0.31706189, -0.20746552])
        a                 b                 c
```

best values by minimizing MSE or related

```
In [16]: Cov
Out[16]:
array([[ 0.15053344, -0.28309719, -0.05890134],
       [-0.28309719,  0.6630454 , -0.02965514],
       [-0.05890134, -0.02965514,  0.33825647]])
```

$$\begin{pmatrix} \sigma_a^2 & cov(a,b) & cov(a,c) \\ cov(b,a) & \sigma_b^2 & cov(b,c) \\ cov(c,a) & cov(c,b) & \sigma_c^2 \end{pmatrix}$$

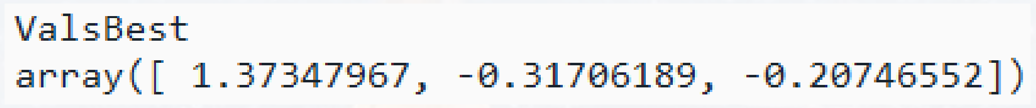
covariance matrix for the model parameters

- **non diagonal values should be ≈ 0** (mutually independent)
- **diagonal: squared errors** (see module 8)

## Data Fitting and Regression:

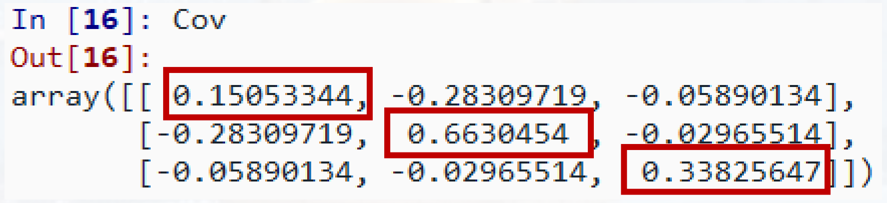
```python
from scipy.optimize import curve_fit


def fun_to_fit(x, a, b, c):
        return a*x**2 + b*x + c


ValsBest, Cov = curve_fit(fun_to_fit, x, y)
```

$$\boxed{\begin{aligned} a &= \phantom{-}1.37 \pm 0.39 \\ b &= -0.32 \pm 0.81 \\ c &= -0.21 \pm 0.58 \end{aligned}}$$



```
ValsBest
array([ 1.37347967, -0.31706189, -0.20746552])
```

a b c



```
In [16]: Cov
Out[16]:
array([[ 0.15053344, -0.28309719, -0.05890134],
       [-0.28309719,  0.6630454 , -0.02965514],
       [-0.05890134, -0.02965514,  0.33825647]])
```

$$\begin{pmatrix} \sigma_a^2 & cov(a,b) & cov(a,c) \\ cov(b,a) & \sigma_b^2 & cov(b,c) \\ cov(c,a) & cov(c,b) & \sigma_c^2 \end{pmatrix}$$

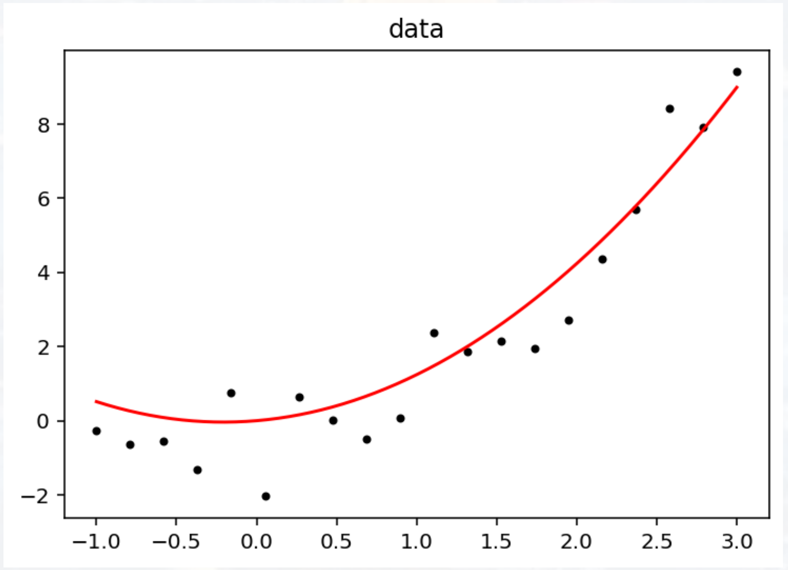
```python
error_1sigma = np.sqrt(np.diagonal(Cov))
```

## Data Fitting and Regression:

```
from scipy.optimize import curve_fit


def fun_to_fit(x, a, b, c):
        return a*x**2 + b*x + c


ValsBest, Cov = curve_fit(fun_to_fit, x, y)
```

$$\boxed{\begin{aligned} a &= \phantom{-}1.37 \pm 0.39 \\ b &= -0.32 \pm 0.81 \\ c &= -0.21 \pm 0.58 \end{aligned}}$$

$$y = 1.37\,x^2 - 0.32\,x - 0.21$$

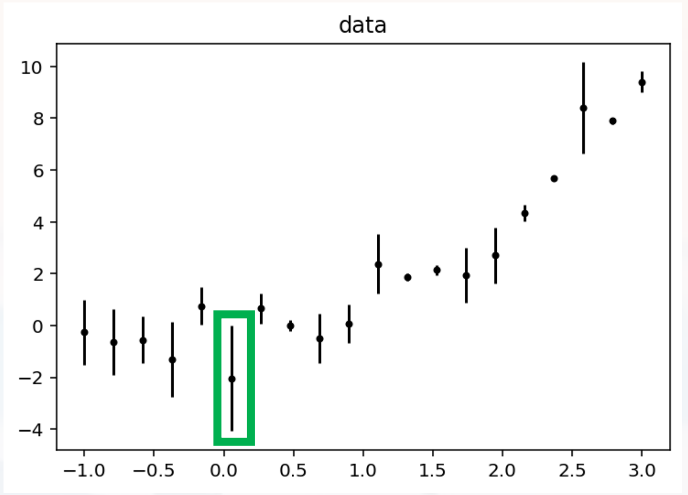


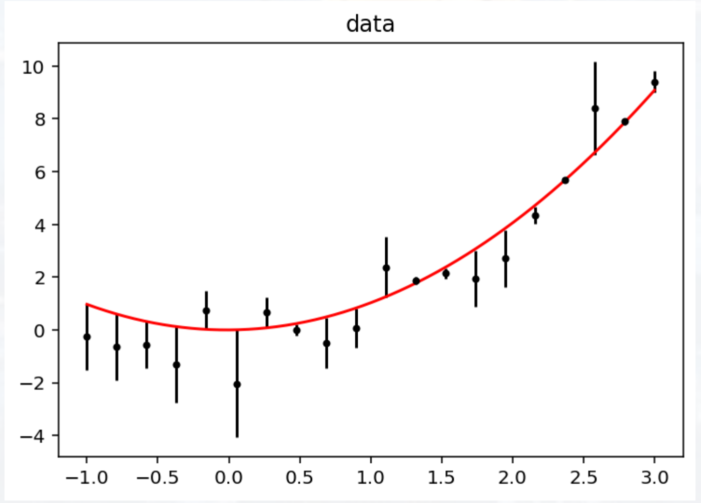
## Data Fitting and Regression:

```
from scipy.optimize import curve_fit


def fun_to_fit(x, a, b, c):
        return a*x**2 + b*x + c


ValsBest, Cov = curve_fit(fun_to_fit, x, y)
```

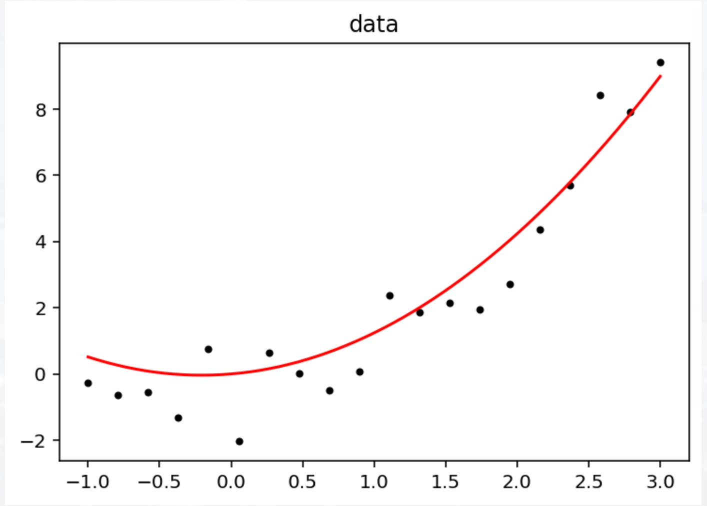
if error bars know → **error weighted fit**!

```
ValsBest, Cov = curve_fit(fun_to_fit, x, y, sigma = err, absolute_sigma = True)
```





vs

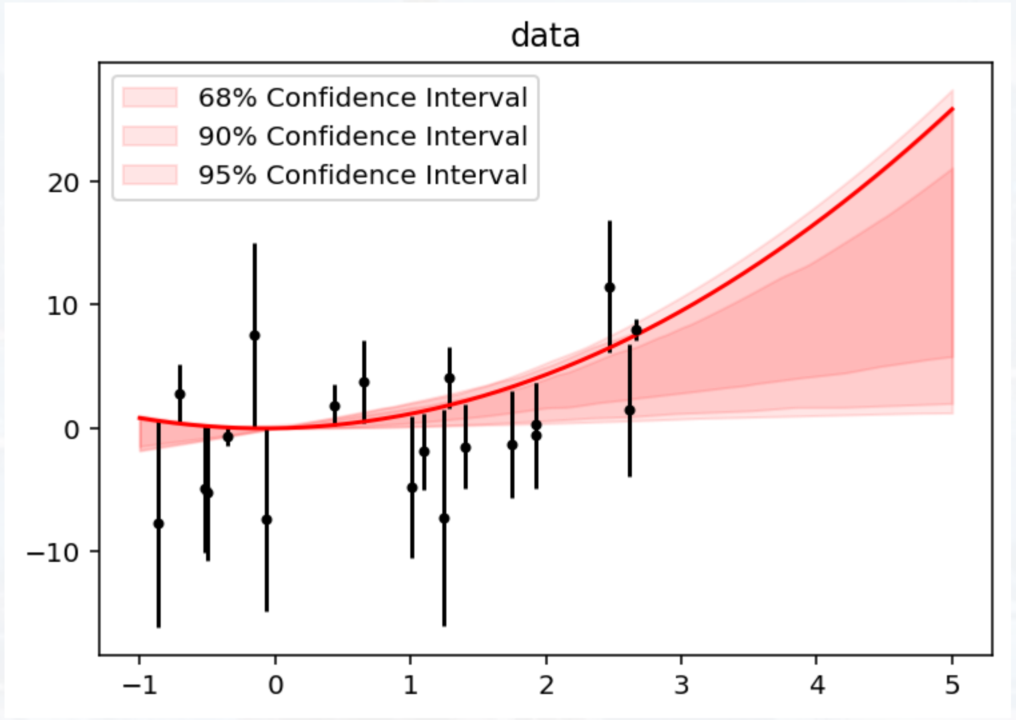


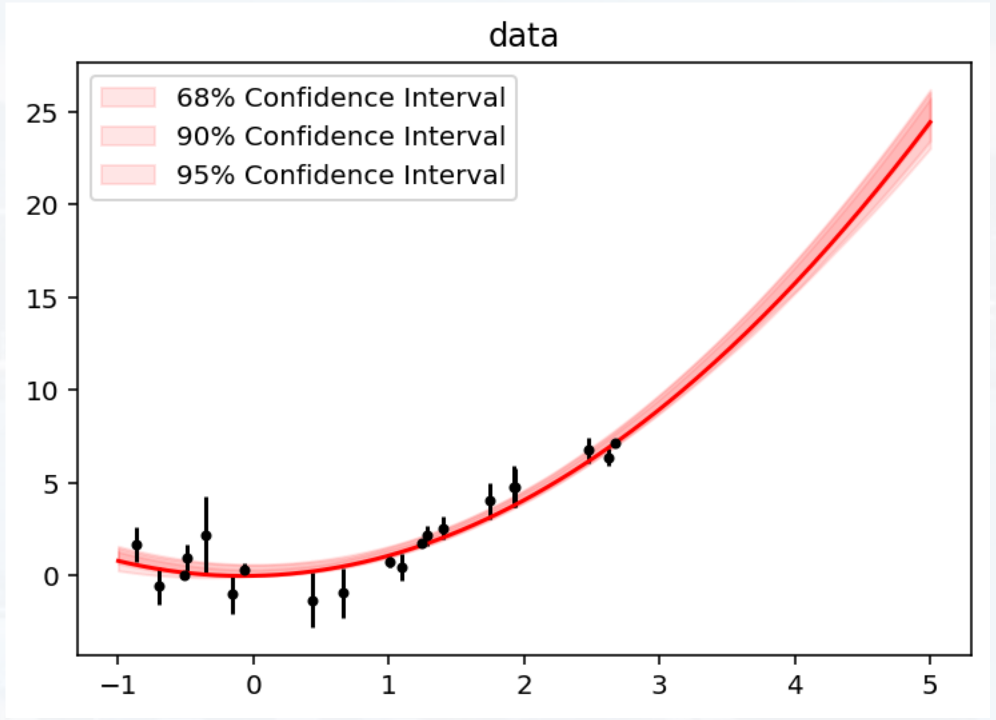
## Data Fitting and Regression:

data points have errors and/or fit parameter have uncertainties too

→ implies uncertainties of the model
→ confidence band/interval

$$\boxed{\begin{aligned} a &= \phantom{-}1.37 \pm 0.39 \\ b &= -0.32 \pm 0.81 \\ c &= -0.21 \pm 0.58 \end{aligned}}$$





## Data Fitting and Regression:

data points have errors and/or fit parameter have uncertainties too

$$\boxed{\begin{aligned} a &= \phantom{-}1.37 \pm 0.39 \\ b &= -0.32 \pm 0.81 \\ c &= -0.21 \pm 0.58 \end{aligned}}$$

→ implies uncertainties of the model
→ confidence band/interval
→ idea: **bootstrapping!**

## Data Fitting and Regression:

data points have errors and/or fit parameter have uncertainties too

$$\begin{aligned} a &= \phantom{-}1.37 \pm 0.39 \\ b &= -0.32 \pm 0.81 \\ c &= -0.21 \pm 0.58 \end{aligned}$$

→ implies uncertainties of the model
→ confidence band/interval
→ idea: **bootstrapping!**

if **no errors of $y_i$ known**

→ assuming that fitted parameters follow a **normal distribution**, i.e. $a = 1.37 \pm 0.39$ where $\mu_a = 1.37$ and $\sigma_a = 0.39$ and so on…

→ **varying** the parameters **within their errors** using `np.random.normal(`$\mu_a$`, `$\sigma_a$`, N)` **N** times

→ for each **N, generating a curve fit**

→ from set of N curve fits → **calculating percentiles** for confidence band/ interval

## Data Fitting and Regression:

data points have errors and/or fit parameter have uncertainties too

$$\boxed{\begin{aligned} a &= \phantom{-}1.37 \pm 0.39 \\ b &= -0.32 \pm 0.81 \\ c &= -0.21 \pm 0.58 \end{aligned}}$$

→ implies uncertainties of the model
→ confidence band/interval
→ idea: **bootstrapping!**

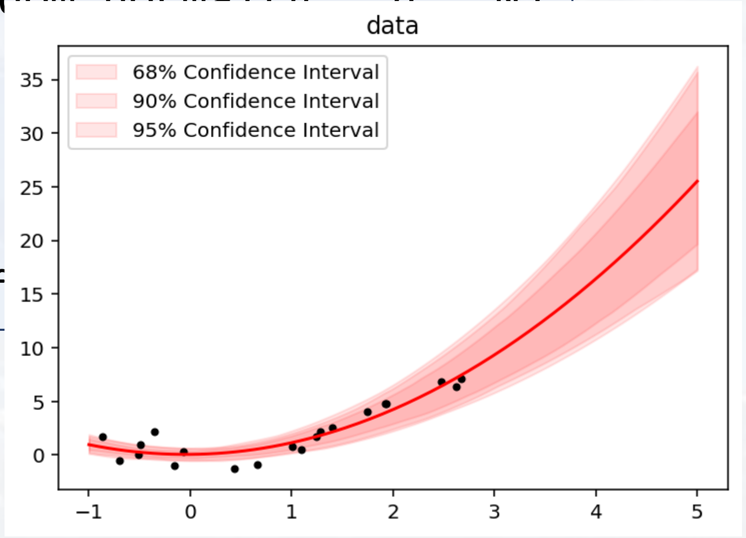
if **no errors of $y_i$ known**

→ assuming that fitted parameters follow a **normal distribution**, i.e. $a = 1.37 \pm 0.39$ where $\mu_a = 1.37$ and $\sigma_a = 0.39$ and so on…

→ **varying** the parameters **within their errors** using `np.random.normal(`$\mu$ $\sigma$ N) **N** times

→ for each **N, generating a curve fit**

→ from set of N curve fits → **calculating percentiles** for conf



## Data Fitting and Regression:

data points have errors and/or fit parameter have uncertainties too

$$a = \quad 1.37 \pm 0.39$$
$$b = -0.32 \pm 0.81$$
$$c = -0.21 \pm 0.58$$

→ implies uncertainties of the model
→ confidence band/interval
→ idea: **bootstrapping!**

if **errors of $y_i$ known**

→ assuming that errors of $y_i$ follow a **normal distribution**, i.e. $\boldsymbol{y_i(boot) = y_i \pm \sigma_i}$

→ **varying** all $y_i$ **within their errors** using `np.random.normal(`$y_i$`,` $\sigma_i$`, N)` **N** times

→ for each **N, generating a curve fit**

→ from set of N curve fits → **calculating percentiles** for confidence band/ interval

## Data Fitting and Regression:

data points have errors and/or fit parameter have uncertainties too

$$a = 1.37 \pm 0.39$$
$$b = -0.32 \pm 0.81$$
$$c = -0.21 \pm 0.58$$

→ implies uncertainties of the model
→ confidence band/interval
→ idea: **bootstrapping!**

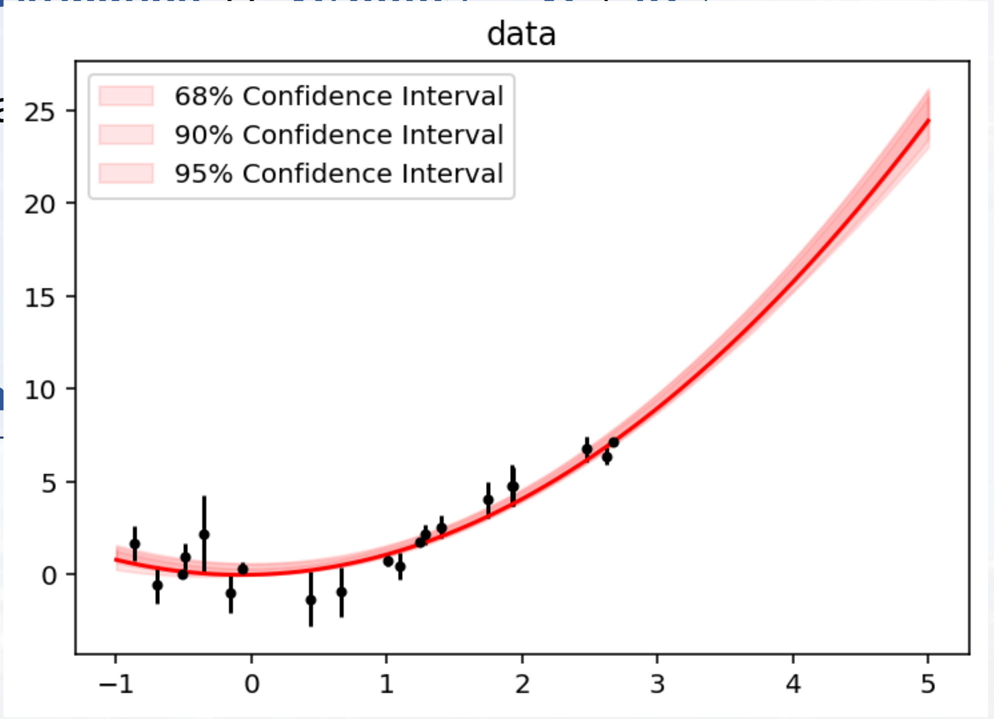
if **errors of $y_i$ known**

→ assuming that errors of $y_i$ follow a **normal distribution**, i.e $y_i(boot) = y_i + \sigma_i$

→ **varying** all $y_i$ **within their errors** using np.ra
**N** times

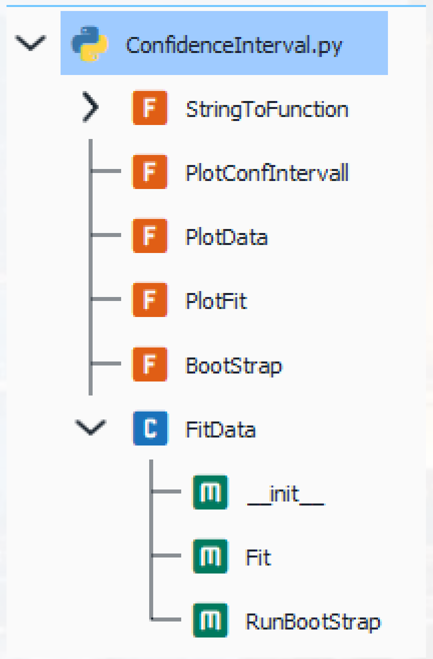
→ for each **N, generating a curve fit**

→ from set of N curve fits → **calculating percen**

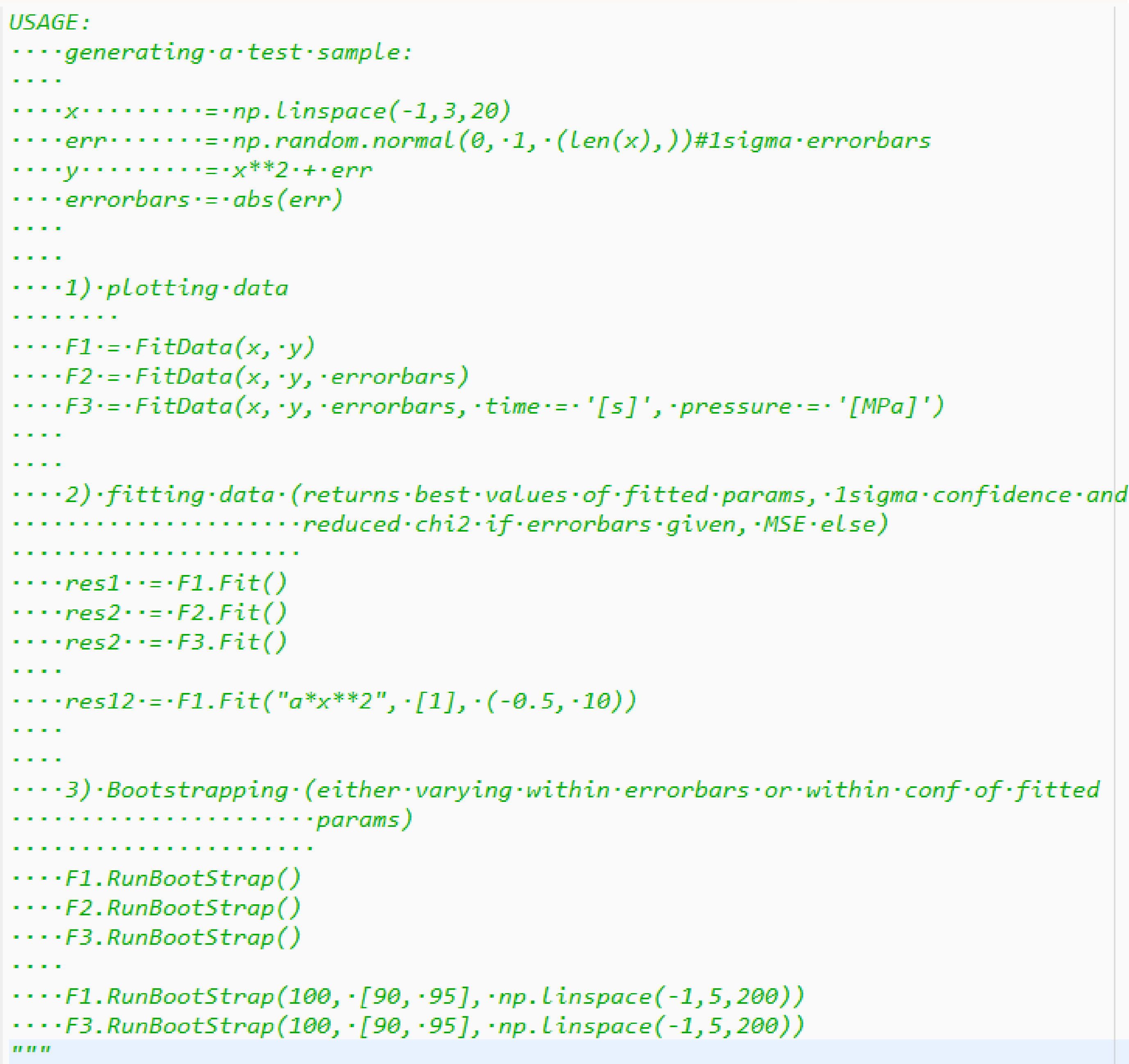


## bootstrapping

explore the .py file
`ConfidenceInterval.py`

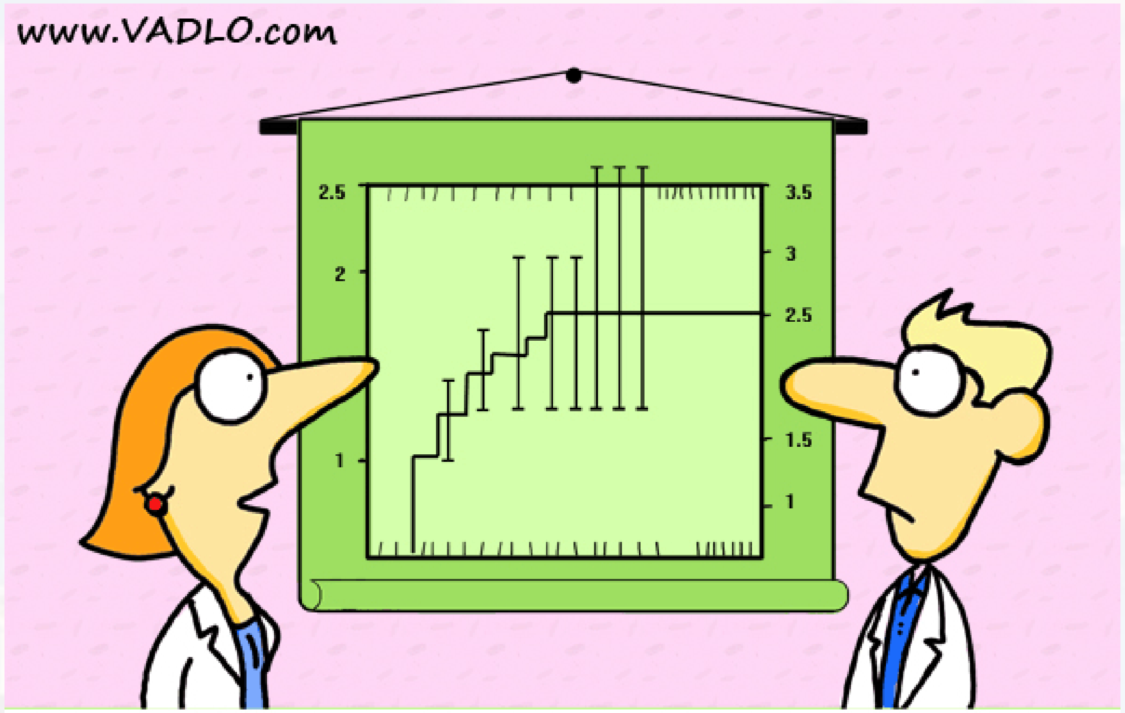


```
ConfidenceInterval.py
    StringToFunction
    PlotConfIntervall
    PlotData
    PlotFit
    BootStrap
    FitData
        __init__
        Fit
        RunBootStrap
```



```
USAGE:
    generating a test sample:

    x         = np.linspace(-1,3,20)
    err       = np.random.normal(0, 1, (len(x),))#1sigma errorbars
    y         = x**2 + err
    errorbars = abs(err)


    1) plotting data

    F1 = FitData(x, y)
    F2 = FitData(x, y, errorbars)
    F3 = FitData(x, y, errorbars, time = '[s]', pressure = '[MPa]')


    2) fitting data (returns best values of fitted params, 1sigma confidence and
                     reduced chi2 if errorbars given, MSE else)

    res1  = F1.Fit()
    res2  = F2.Fit()
    res2  = F3.Fit()

    res12 = F1.Fit("a*x**2", [1], (-0.5, 10))


    3) Bootstrapping (either varying within errorbars or within conf of fitted
                      params)

    F1.RunBootStrap()
    F2.RunBootStrap()
    F3.RunBootStrap()

    F1.RunBootStrap(100, [90, 95], np.linspace(-1,5,200))
    F3.RunBootStrap(100, [90, 95], np.linspace(-1,5,200))
"""
```

Thank you very much for your attention!



"Did you really have to show the error bars?"