Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Applied Machine Learning in Engineering

Lecture 01 summer term 2025

Prof. Merten Stender

Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

www.tu.berlin/cpsme
merten.stender@tu-berlin.de

Organizational Details

- **Lecture**:
  - **Wednesday 10am-12pm CT, 90min, room EB 107**
  - In-person, no video recording available
- **Exercise**:
  - **Tuesday 12pm-02pm CT, 90min, room EB 202**
  - Hands-on coding exercises
  - Please bring your own computer
- **ISIS (LINK)**:
  - Slides, lecture notes and exercise sheets will be published ahead in time
  - **Poll on active course participation**: your choice until April 30th (otherwise: unsubscription)
- **Exam**:
  - Written digital exam **at TU Berlin**. Dates: 21.07.2025 (08:00am); 09.10.2025 (10:00am)
  - Exam registration: ONLY through Moses (registration open!)

Recap: Lecture 00

- **Uncertain model parameters**
  - Structural damping of metals, …
  - Nonlinear behavior (elastomers stiffness, …)

- **Inherent modeling assumptions and limitations**
  - Simplified constitutive models, …
  - Idealized assumptions on homogeneity, …

- **Speed and energy-efficiency**
  - Homogenization of heterogeneous materials
  - Low-order yet fast surrogate models

Data-driven methods are particularly promising and powerful when a handcrafted algorithm is not existent or extremely difficult to formulate, parametrize, or execute.

Recap: Lecture 00

Structured data (tabular data)

- **Features** (attributes): quantities that describe measurements or characteristic properties of an individual data sample (record)

rows:
data samples

columns: features
CRIM, NOX, …

targets

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \cdots & \vdots \\ x_{N,1} & \cdots & x_{N,n} \end{bmatrix} \downarrow N \qquad \xrightarrow[\text{(to be approximated)}]{\text{Unknown relationship}} \qquad \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

$$\xrightarrow{\;n\;}$$

- High-dimensional data sets: n very large
- Big data: N very large (and n potentially, too)

implies

Recap: Lecture 00

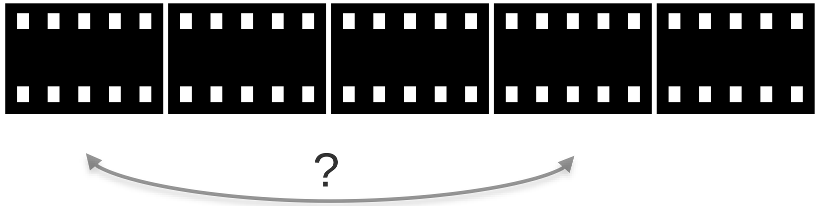
Unstructured data (also denoted as non-tabular data)

- Examples:
  - Text
  - Audio
  - Video
  - Images …

- Special about unstructured data:
  - Additional latent dimensions
  - **Order matters** (latent dim.)

[I | live | in | Munich | but | work | in | Berlin]

?



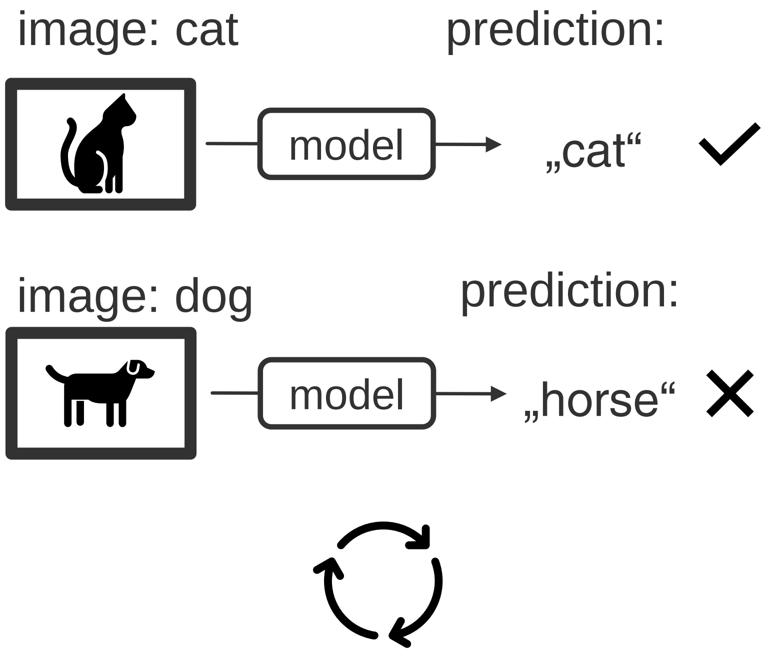
Order of features not interchangeable!

audio can be stored in an array but is not structured!

Latent dimension = time

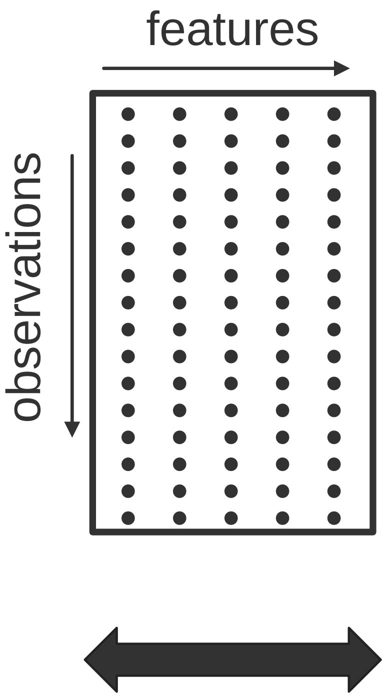
Recap: Lecture 00

supervised learning (predictive task)

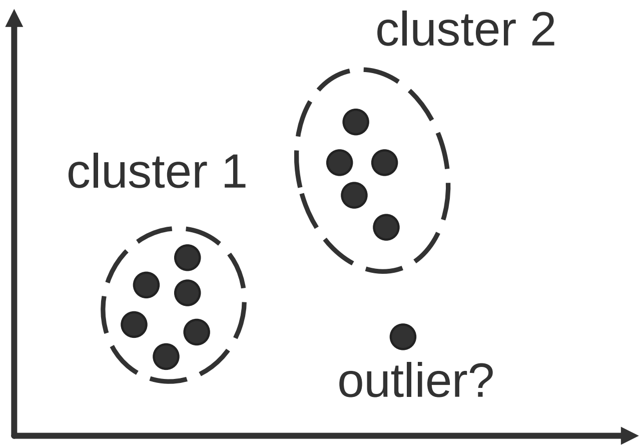


model training = reduce prediction error

data (tabular)



unsupervised learning (descriptive task)

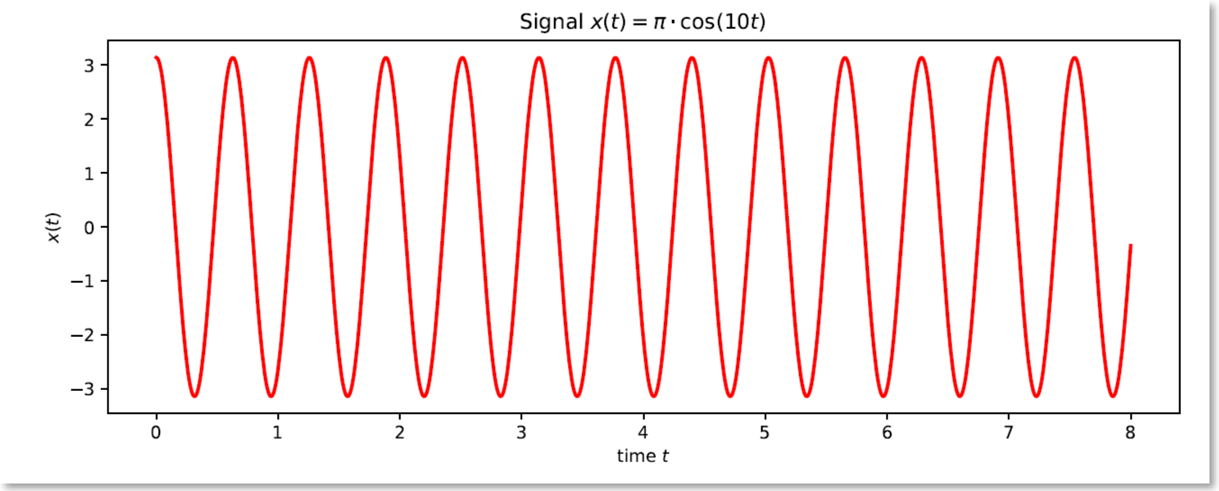


finding clusters, groups and anomalies

Recap: Exercise 00

- **Set up a Python 3.10+ environment for the semester**
- Build some basic Python **programming skills** (following an online tutorial).
- Create a first **figure** using matplotlib

```python
t = np.arange(start=0, stop=8 + 0.01, step=0.01)  # time vector
freq = 10
x = np.pi * np.cos(freq * t)  # x(t)

fig = plt.figure()  # figsize=(8,4), dpi=200)
plt.plot(t, x, color='red', linewidth=2)
plt.xlabel(r'time $t$')
plt.ylabel(r'$x(t)$')
plt.title(fr'Signal $x(t) = \pi \cdot \cos({freq}t)$')
plt.savefig('my_plot_of_cos_signal.png')
plt.show()
```



Recap: Exercise 00

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Implement the basic **Newton scheme**

Function y=f(x)

```python
def f(x: float) -> float:
    return x ** 3 - 3 * x - 10


def dfdx(x: float) -> float:
    return 3 * x ** 2 - 3
```

Ingredients (helper functions)

```python
def newton_iter(xn: float, f, dfdx) -> float:
    return xn - (f(xn) / dfdx(xn))


def converged_f(xn: float, f) -> bool:
    return f(xn) < 10 ** (-7)


def converged_x(xn_1: float, xn: float) -> float:
    return np.abs(xn_1 - xn) < 10 ** (-4)
```

Newton procedure

```python
# initial guess of the zero
xn = 5
n = 0

# using the first convergence criterion on f(xn)
while not converged_f(xn, f) and n < 100:
    print(f'iteration {n}: xn={xn}')
    xn_plus1 = newton_iter(xn, f, dfdx)
    xn = xn_plus1
    n += 1

print(f'zero of f(x) is at x={xn}')
```

Agenda

- Distance metrics for continuous attributes
- Linear regression fundamentals: Least squares method
- Evaluation of regression models

Python

- Loops, functions
- Type hints
- PEP8 style guide
- Test-driven development
- scikit-learn library

Learning outcomes

Learn to …

- Formulate a linear regression learning task
- Derive the closed-form solution to the least-squares linear regression
- Measure regression model prediction errors using the R^2 metric
- Understand the limitations of metrics and the importance of visual evaluation
- Understand the principles of Test-Driven Development (TDD)

Know about …

- The Minkowski distance
- The meaning and interpretation of the coefficient of determination (R^2)
- Separating what and how code is doing during the implementation (TDD)

Linear regression

Metrics for Continuous Attributes

- Continuous attributes allow for distance operations. Some distance metrics:

Minkowski distance $d(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} |\boldsymbol{x_i} - \boldsymbol{y_i}|^{\boldsymbol{r}}\right)^{\boldsymbol{1/r}}$

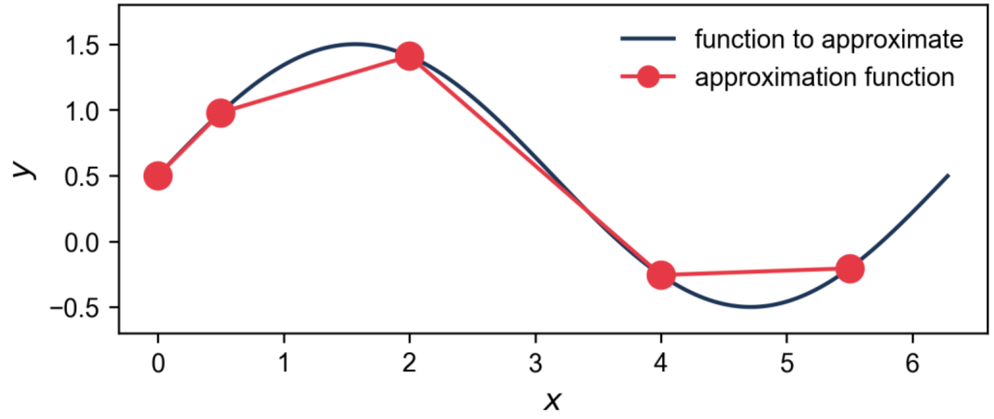
- Euclidean distance (L_2) $\|\mathbf{q}\|_2 = \left(\sum_{i=1}^{n} |q_i|^2\right)^{1/2}$ for distance vector $\mathbf{q} = \mathbf{x} - \mathbf{y}, \ \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$
- Manhatten distance (L_1) $\|\mathbf{q}\|_1 = \sum_{i=1}^{n} |q_i|$
- Supremum distance (L_∞) $\|\mathbf{q}\|_\infty = \lim_{\boldsymbol{r} \to \infty} \left(\sum_{i=1}^{n} |q_i|^r\right)^{\boldsymbol{1/r}}$

Regression – Definition

Definition: Regression is the task of learning a target function f that maps each attribute set $\boldsymbol{x}$ into a continuous-valued output y.

[Tan, Introduction to Data Mining]

- **Goal**: find a target function $f(\boldsymbol{x})$ with minimal error on training data
- **Error**: different definitions of error available



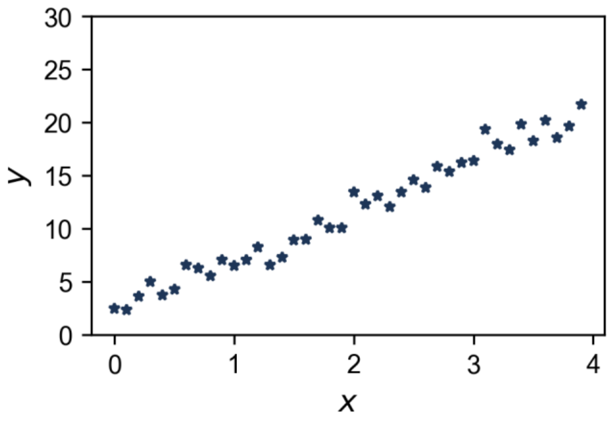
- **Methods**:
  - Linear regression
  - Decision Trees
  - Neural Networks
  - …

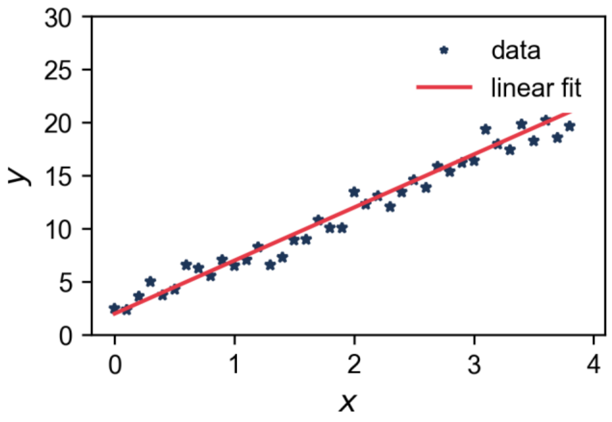
Linear Regression

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Find target function f that obtains a linear relationship between
  - (explanatory) variables $\boldsymbol{x}$
  - and target value y
- Target function, i.e. lin. regression model, is parameterized in $\boldsymbol{\theta} = [\theta_0, \dots]^\top$: $f(\boldsymbol{\theta})$
- Data set $D = \{(\boldsymbol{x}_i, y_i) \mid i = 1, \dots, N\}$, $x_i = \left[x_{i,1}, x_{i,2}, \dots, x_{i,n}\right]^\top \in \mathbb{R}^n$, N samples, n-dim. feature space

- Scalar example: $x \in \mathbb{R}^1$



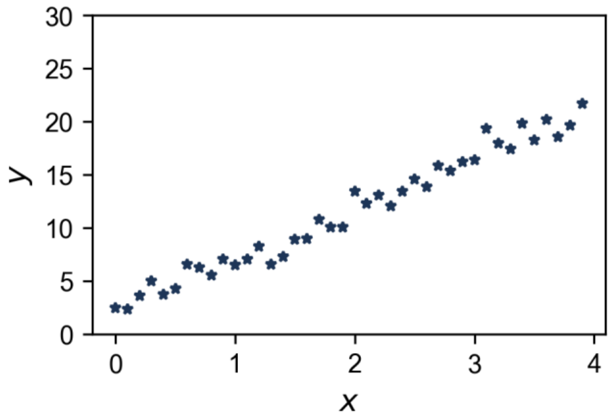


Linear Regression

- Scalar variable x and scalar target value y → find $f(x) = \theta_0 + \theta_1 x = \boldsymbol{x}_i^\top \boldsymbol{\theta}$ with $\boldsymbol{x} = [1, x_i]^\top$

- **Generating process** $y_i = \theta_0 + \theta_1 x_i + \epsilon_i = \boldsymbol{x}_i^\top \boldsymbol{\theta} + \epsilon_i$ for $i = 1, \dots, N$ samples
- Unobserved random variable ϵ causing deviations from a perfectly linear relationship

- **Prediction:** $\hat{y} = f(x)$ (target function evaluated at x)



- **Prediction error:** $\mathcal{L} = \sum \|y_i - \hat{y}_i\| = \sum \left\| y_i - \boldsymbol{x}_i^\top \boldsymbol{\theta} \right\|$

- **Error metrics:** $\|\cdot\|$: absolute error, squared error, …

- Squared error (sum of squares) → **Least squares linear regression**

Least Squares Linear Regression

- Solving a linear regression task for the minimum **sum of squared errors (SSE)**
- Sum of squared errors: $\mathcal{L}_{\text{SSE}} = \sum_i (y_i - \hat{y}_i)^2,\ i = 1, \dots N$
- Least squares method: $\min_{\theta} \left\| y_i - \boldsymbol{x}_i^{\top} \boldsymbol{\theta} \right\|$ for data samples $D = \{(\mathbf{x}_i, y_i) \mid i = 1, \dots, N\}$,
- Optimal model parameters $\boldsymbol{\theta}^*$
- Loss for scalar setting: $\mathcal{L}(\boldsymbol{\theta}, D) = \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \theta_0 - \theta_1 x_i)^2$
- Minimum of SSE: basic calculus. Vanishing gradient of $\mathcal{L}$ with respect to model parameters θ

$$\frac{\partial \mathcal{L}}{\partial \theta_0} = 0 \text{ and } \frac{\partial \mathcal{L}}{\partial \theta_1} = 0$$

Least Squares Linear Regression

- Loss for scalar setting: $\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \theta_0 - \theta_1 x_i)^2$

- Vanishing gradient of $\mathcal{L}$

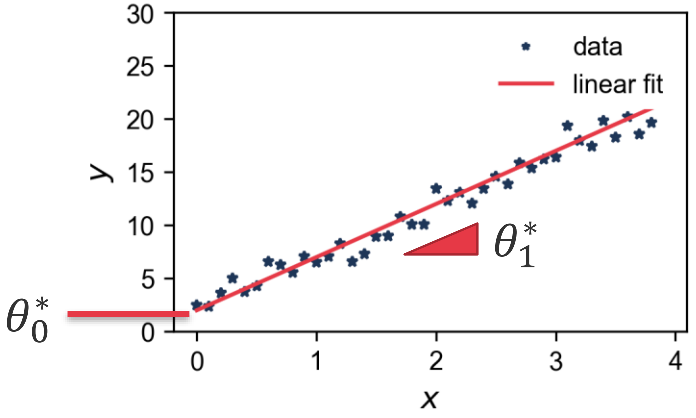
$$\frac{\partial \mathcal{L}}{\partial \theta_0} = -2 \sum_i (y_i - \theta_0 - \theta_1 x_i) = 0$$

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = -2 \sum_i (y_i - \theta_0 - \theta_1 x_i)\, x_i = 0$$

pen & paper exercise

- Normal equation

$$\underbrace{\begin{bmatrix} N & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} \sum_i y_i \\ \sum_i y_i x_i \end{bmatrix}}_{\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}}$$



→ Solve for $\boldsymbol{\theta}$ to find optimal parameters $\boldsymbol{\theta}^*$

Least Squares: Multi-Regression

- Feature space is multi-dimensional $\boldsymbol{x} \in \mathbb{R}^n, n > 2$, $\boldsymbol{x} = [1, x_1, \dots, x_n]^\top$, $D = \{(\boldsymbol{x}_i, y_i\}, i = 1, \dots N$

- Linear multi-regression: $\hat{y}_i = \boldsymbol{x}_i^\top \boldsymbol{\theta},$ $\boldsymbol{x}_i = \left[1, x_{i,1}, \dots, x_{i,n}\right]^\top$, $\boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^n$

  full data set: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta}$

- Sum of squares $\mathcal{L} = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2$

- Solution: $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathcal{L}(D, \boldsymbol{\theta})$

Task: Compute Normal Form

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Loss* $\mathcal{L} = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2$
- Model predictions $\hat{\mathbf{y}} = \mathbf{X\theta}$
- Minimum: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \mathbf{0}$

- Compute solution $\boldsymbol{\theta}^*$

*Inversed formulation of the L2 loss w.r.t. previous slides (without any effect on result)

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \left(\boldsymbol{\theta}^\top \mathbf{X}^\top - \mathbf{y}^\top\right) (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \left(\boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} - \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \mathbf{y}^\top \mathbf{y}\right) = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \left(\boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} - 2\boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y}\right) = 0$$

$$2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} - 2\mathbf{X}^\top \mathbf{y} = 0$$

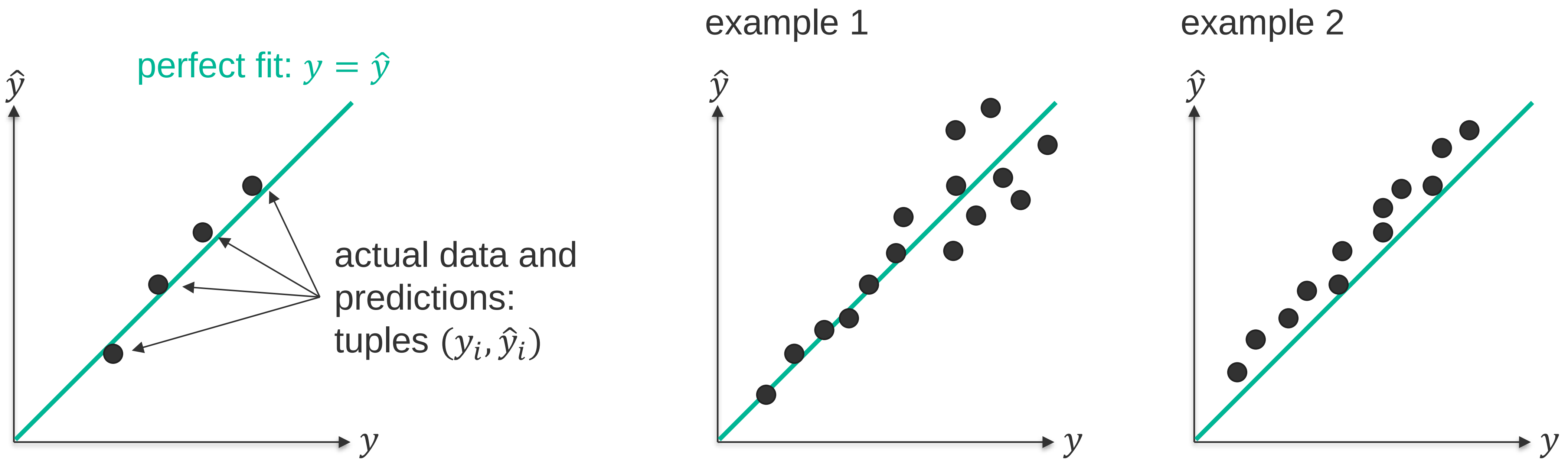
$$\boxed{\boldsymbol{\theta} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}}$$

Measuring Regression Performance

Visual Regression Model Evaluation

- Whenever building a regression model, it needs to be evaluated (on a <u>hold-out</u> data set)

- A visual evaluation tool is the scatter plot of expected (ground truth) vs. predicted values



Measuring Goodness of Fit: R^2

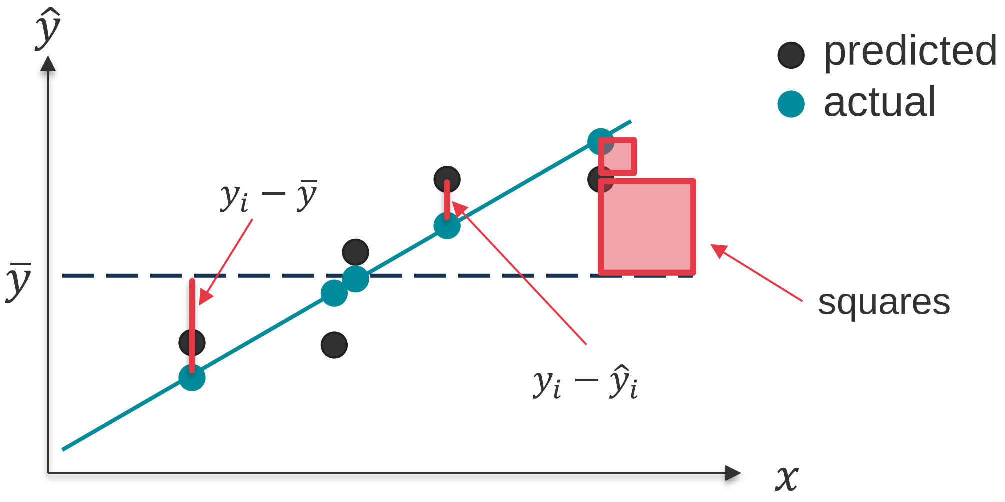
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Linear regression is a **supervised learning** approach
  - True target values are known → ground truth targets y_i
  - Predictions $\hat{y}$ can be compared against ground truth for measuring the goodness of fit

- **Coefficient of determination** $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$, $i = 1, \dots, N$, with sample mean $\bar{y} = \frac{1}{N}\sum_i y_i$

  - Residual sum of squares $\sum_i (y_i - \hat{y}_i)^2$
  - Total sum of squares $\sum_i (y_i - \bar{y})^2$

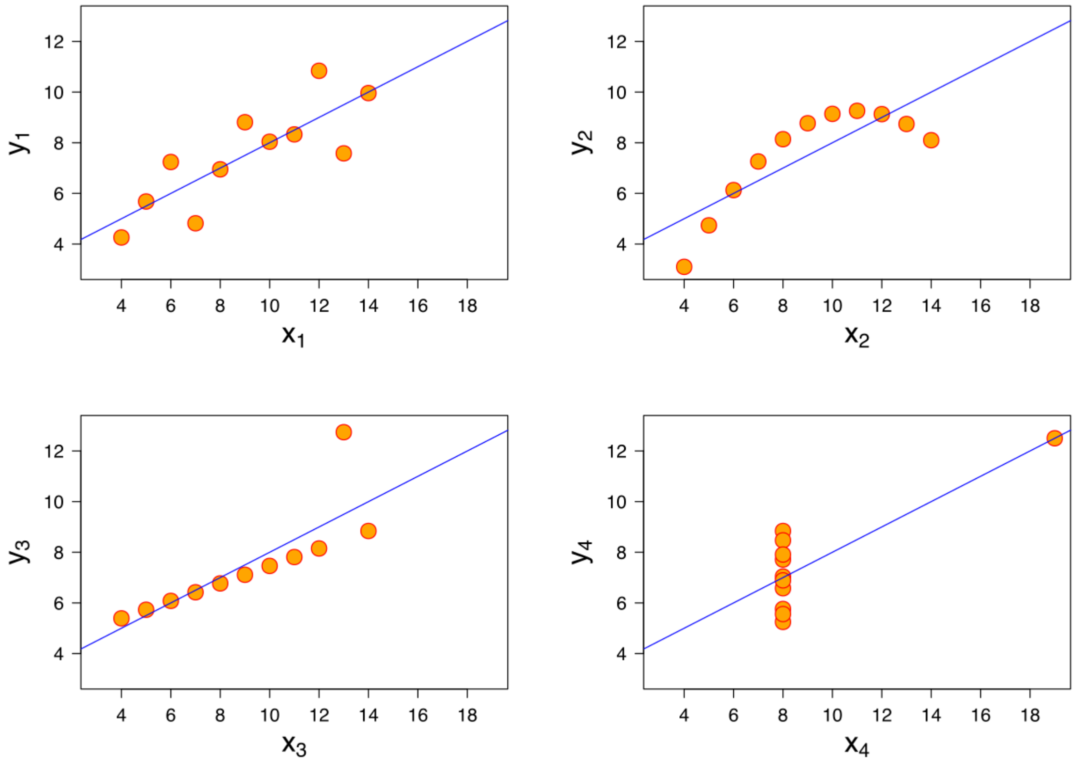
  - Perfect fit: $R^2 = 1.0$
  - Baseline model $f(x) = \bar{y}$ $R^2 = 0.0$
  - ‚Worse models' $R^2 < 0.0$



Anscombe's quartet

- Proposed by Francis Anscombe in 1973
  Graphs in Statistical Analysis". American Statistician. 27 (1): 17–21

- Four data sets with (approx.) **same statistics**
  - Mean (x, y)
  - Variance (x, y)
  - Correlation (x, y)
  - Linear regressor $f(x) = \theta_0 + \theta_1 x + \epsilon$
  - R^2 of linear regressor



Wikipedia

Take-away message:
- Always be cautious with metrics
- Always check model prediction results visually

Python

Python Basics

Loops, Functions, Type Hints

Loops

- **FOR loop**: A for loop lets you repeat a block of code for each item in a sequence (like a list, string, or range of numbers).

```
for item in sequence:
    # do sth. with item
```

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print("I like", fruit)
```

```
# range(n) gives numbers from 0 to n-1

for i in range(5):
    print(i)
```

- **WHILE loop**: A while loop keeps repeating as long as a condition is True

```
while condition:
    # do something
```

```
count = 0

while count < 3:
    print("Count is", count)
    count += 1
```

```
# This will never stop!

while True:
    print("Looping forever!")
```

Functions

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Function: A function is a named block of code that does something. You can call it whenever you need it
  - Define this task once, then reuse it whenever you want
  - Functions help avoid repetition and organize code

```
def function_name():
    # code to run
```

```
def say_hello():
    print("Hello!")

say_hello()
say_hello()

>>Hello!
>>Hello!
```

```
def greet(name):
    print(f"Hello, {name}!")

greet('Bob')
greet('Anna')

>>Hello, Bob!
>>Hello, Anna!
```

```
def square(x):
    return x * x

result = square(4)
print(result)

>>16
```

- Can, but does not require, one/multiple return objects
- Naming convention: **snake case** my_fancy_function()

Type Hints

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Recap Exercise 00: Plotting call wrapped into a function with type hints

```python
"""
Extra: wrap it into a function, accepting amplitude and frequency and returning
the plot
"""

def plot_cos_signal(amplitude: float = 1.0, frequency: float = 1.0) -> None:
    t = np.arange(start=0, stop=8 + 0.01, step=0.01)  # time vector
    x = amplitude * np.cos(frequency * t)  # x(t)

    plt.figure(figsize=(8, 4), dpi=200)
    plt.plot(t, x, color='red', linewidth=2)
    plt.xlabel(r'time $t$')
    plt.ylabel(r'$x(t)$')
    plt.title(fr'Signal $x(t) = {amplitude} \cdot \cos({frequency}t)$')
    plt.savefig('extra_plot_of_cos_signal.png')
    plt.show()


# use the function to plot a different cosine signal
plot_cos_signal(amplitude=0.5, frequency=3.14)
```

function definition

arguments

default values

type hints

call of the function

PEP8 Style Guide

PEP 8 – Style Guide for Python Code

Coding conventions for Python code. Standard style guide helps avoiding common errors

- Improves collaboration with other developers (everyone talking in same accent)
- Link: https://peps.python.org/pep-0008/ ; better to read presentation at https://pep8.org/
- PEP: Python Enhancement Proposals – style guide is constantly evolving over time!

Preliminary guides

Never use as a variable name:

- Capital or small-cap „o“ → can be mixed up with zero
- Capital „i“ → can be mixed up with 1
- Small-cap „L“ → can be mixed up with 1 or capital „i“

PEP 8 Naming Conventions (Basics)

- **Python scripts**
  - Module: lower-case words separated by underscore
  - Package: lower-case words, no separators

```
my_module.py
mypackage.py
```

- **Variables**
  - Small-cap characters or words, underscore-separated
  - CONSTANTS: all-capital letters

```
x = 5
my_variable = 'ten'
PI = 3.14
```

- **Functions**
  - No single characters
  - Small-cap words, underscore-separated

```
def my_function():
    pass
```

- **Classes**
  - Camel-case (capitalize each word, no separator)
  - Methods: syntax like functions

```
class Car():
class SportsCar():
```

PEP 8 Syntax Conventions

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Whitespaces around operators, and around „=“
- Line indentation and line breaks
- … and many more!
- Further reading: [Link]

```python
# Correct:
spam(ham[1], {eggs: 2})
```

```python
# Wrong:
spam( ham[ 1 ], { eggs: 2 } )
```

```python
# Correct:
if x == 4: print(x, y); x, y = y, x
```

```python
# Wrong:
if x == 4 : print(x , y) ; x , y = y , x
```

```python
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

Test-Driven Development

Testing Code

Why write code that tests other pieces of code?

Ensure correct functionality

- Does a piece of code know what we request it to do?
- Does code catch edge cases and undesired usage?

Continuous integration (version control, Git, …)

- Automatically checking functionality after code changes
- Pushing to code base only when passing tests

Documentation and collaboration

- Good tests can replace long documentation
- Splitting responsibilities: requirement definition and implementation

Test-Driven Development (TDD)

Paradigm in software development (among others)

Separate what the code needs to do from how it does it

- Focus on
  - Modular software: interfaces much more important than the implementation
  - Perspective of a user that knows only the interface of a piece of code (arguments, returns)
  - Thinking of behavior rather than of specific implementation details
  - Higher software quality, robustness, and maintenance readiness

- 3-stage procedure: **RED – GREEN – REFACTOR**
  1. Red: implement tests and make sure that all of them are failed
  2. Green: implement functionalities until all tests are passed
  3. Refactor: clean up and optimize code

Test-Driven Development (TDD)

- Five steps to follow:
  1. Write tests for each feature that you require
  2. Run the tests and make sure that all tests fail (red)
  3. Write the simplest code that passes the tests
  4. Make sure that all tests pass now (green)
  5. Refactor and improve the code from step step 3 and repeat from step 4 (refactor)

Example: Implement a function that adds two numeric numbers (floats, real-valued) using TDD

Interface definition:

```
def adding_function(a, b):
    return sum_a_b
```

Code snippet: → live demo

- `test_add_fun_unittest.py`
- `add_fun.py`

Test-Driven-Development: Result


```python
import unittest
from add_fun import adding_function as add

class TestAddingFunction(unittest.TestCase):

    def test_floats(self) -> None:
        # test the addition of two floats
        self.assertAlmostEqual(add(1.5, 1.5), 3)

    def test_ints(self) -> None:
        # test the addition of two ints
        self.assertEqual(add(1, 2), 3)

    def test_negatives(self) -> None:
        # test adding a negative and a positive number
        self.assertEqual(add(-1, 1), 0)

    def test_types(self) -> None:
        # test whether an exception is raised for non-
        numeric inputs
        # using the context manager
        with self.assertRaises(TypeError):
            add(5, 'five')

if __name__ == "__main__":
    # use the main to run this script directly from your
    editor
    unittest.main()
```

Checks (using `unittest` package)

- Correct result for
  - floats
  - ints
  - positives and negatives
- Check edge cases / unproper usage
  - ValueError raised for string inputs

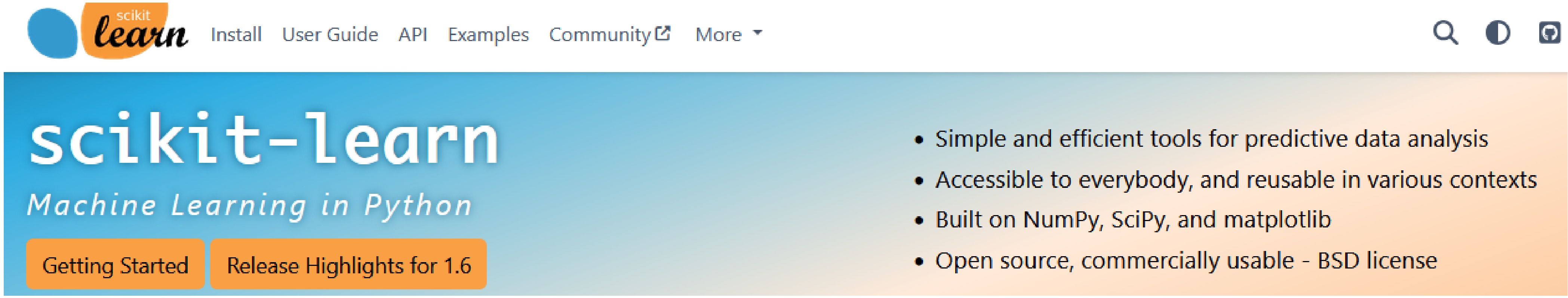
```python
def adding_function(a, b):
    # Add function

    # catch some errors if no numbers are handed over
    if (type(a) is not float) and (type(a) is not int):
        raise TypeError('input a is not of type int or float')

    if (type(b) is not float) and (type(b) is not int):
        raise TypeError('input b is not of type int or float')

    return a + b
```

scikit-learn library

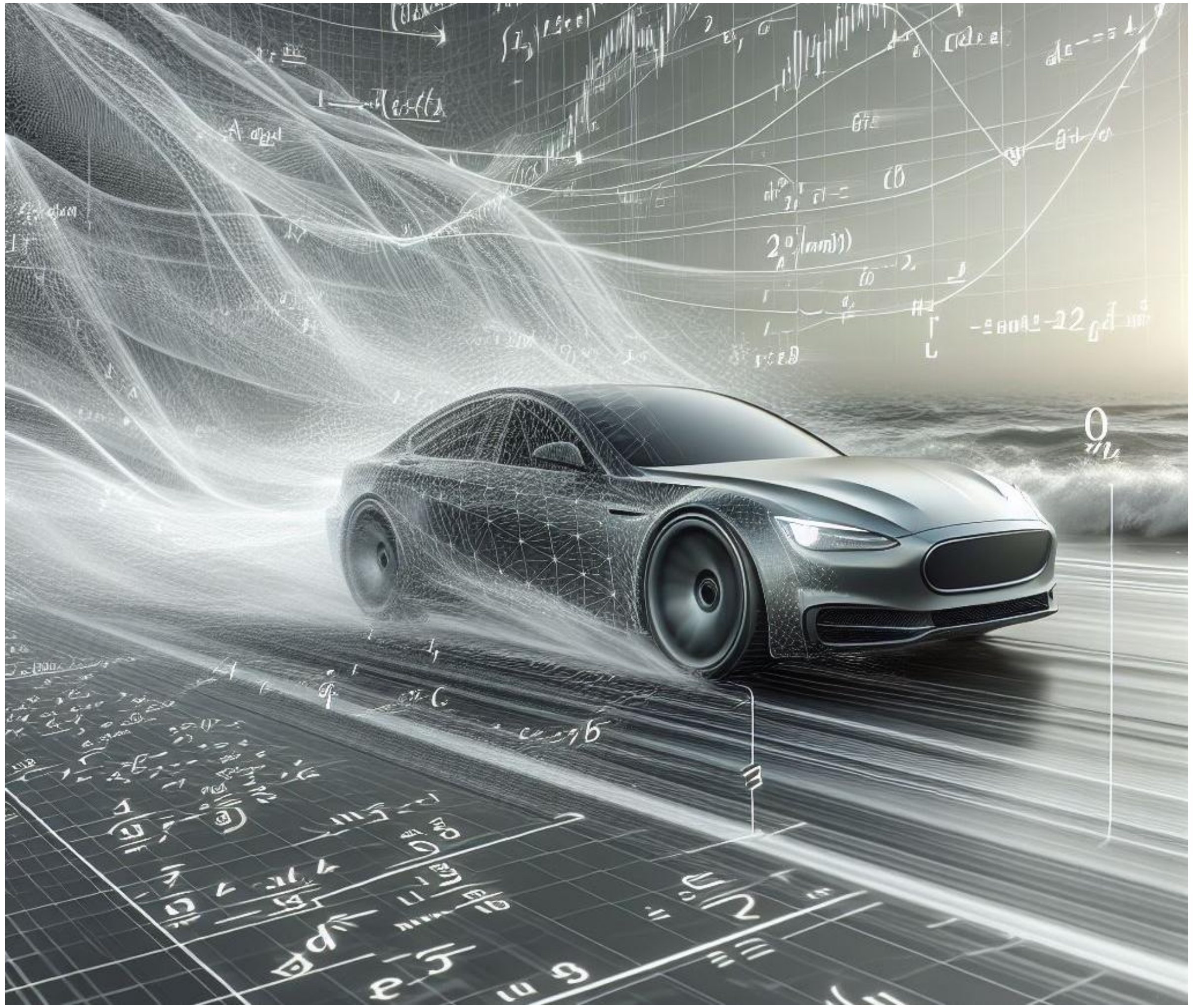
- **scikit-Learn** is the most relevant library for classic machine learning in Python
  - Data preprocessing
  - Wide range of ML models
  - Model evaluation
  - Well-documented with underlying mathematics and literature

`pip install scikit-learn`

- R2 score: `sklearn.metrics.r2_score(y_true, y_pred)`



Exercise 01



Least-Squares Regression

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

1. Implement your own version of a scalar linear regression function using `numpy` and the normal form derived in class.

2. Estimate the effective rolling resistance factor of a car from measurements of vehicle speed and engine power
   - Underlying physics: force $F_{\mathrm{wind}} = c_{\mathrm{W}} A \cdot \frac{\rho_{\mathrm{air}} \cdot v_{\mathrm{rel}}^2}{2}$, $F_{\mathrm{roll}} = c_{\mathrm{R}} M g \cos \alpha$

     power $P = v \cdot F$
   - Unknown random variables in the measurements (wind velocity, road inclination α)
   - Compute the R2 value of your fit and validate with scikit-learn

3. Using test-driven development, implement the R2 error metric

Questions?