# ARTIFICIAL INTELLIGENCE FOR SOFTWARE ENGINEERS

(Part 1: Supervised Machine Learning and Notebooks)

Christian Kaestner

Required Reading: Hulten, Geoff. "Building Intelligent Systems: A Guide to Machine Learning Engineering." (2018), Chapters 16–18, 20.

Suggested complementary reading: Géron, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow", 2nd Edition (2019), Ch 1.

# DISCLAIMER

If you are familiar with supervised machine learning and have used notebooks and scikit-learn before for a class or project, you will probably not learn much new today. Sorry

Next lecture deep learning and symbolic AI.

# LEARNING GOALS

- Understand how machine learning learns models from labeled data (basic mental model)
- Explain the steps of a typical machine learning pipeline and their responsibilities and challenges
- Understand the role of hyper-parameters
- Appropriately use vocabulary for machine learning concepts
- Apply steps of a machine-learning pipeline to build a simple model from static labeled data
- Evaluate a machine-learned classifier using cross-validation
- Explain the benefits and drawbacks of notebooks
- Demonstrate effective use of computational notebooks

# MACHINE LEARNING

> *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. -- Tom Mitchell, 1997*

# DEFINING MACHINE LEARNING (SIMPLIFIED)

learn a function (called model)

$$f(x_1, x_2, x_3, \ldots, x_n) \rightarrow y$$

by observing data

**Examples:**

- Detecting cancer in an image
- Transcribing an audio file
- Detecting spam
- Predicting recidivism
- Detect suspicious activity in a credit card

Typically used when writing that function manually is hard because the problem is hard or complex.

# RUNNING EXAMPLE: HOUSE PRICE ANALYSIS

Given data about a house and its neighborhood, what is the likely sales price for this house?

$$f(size, rooms, tax, neighborhood, \ldots) \rightarrow price$$

# SUPERVISED MACHINE LEARNING

Given a training dataset containing instances $(x_{1,i}, \ldots, x_{n,i}, y_i)$

which consists of features $x_1, \ldots, x_n$ and a corresponding outcome label $y$,

learn a function $f(x_1, \ldots, x_n) \rightarrow y$

that "fits" the given training set and "generalizes" to other data.

# TRAINING DATA FOR HOUSE PRICE ANALYSIS

Collect data from past sales

| Crime Rate | %Large Lots | %Industrial | Near River | # Rooms | ... | Price |
|---|---|---|---|---|---|---|
| 0.006 | 18 | 2.3 | 0 | 6 | ... | 240.000 |
| 0.027 | 0 | 7.0 | 0 | 6 | ... | 216.000 |
| 0.027 | 0 | 7.0 | 0 | 7 | ... | 347.000 |
| 0.032 | 0 | 2.1 | 0 | 6 | ... | 334.000 |
| 0.069 | 0 | 2.1 | 0 | 7 | ... | 362.000 |

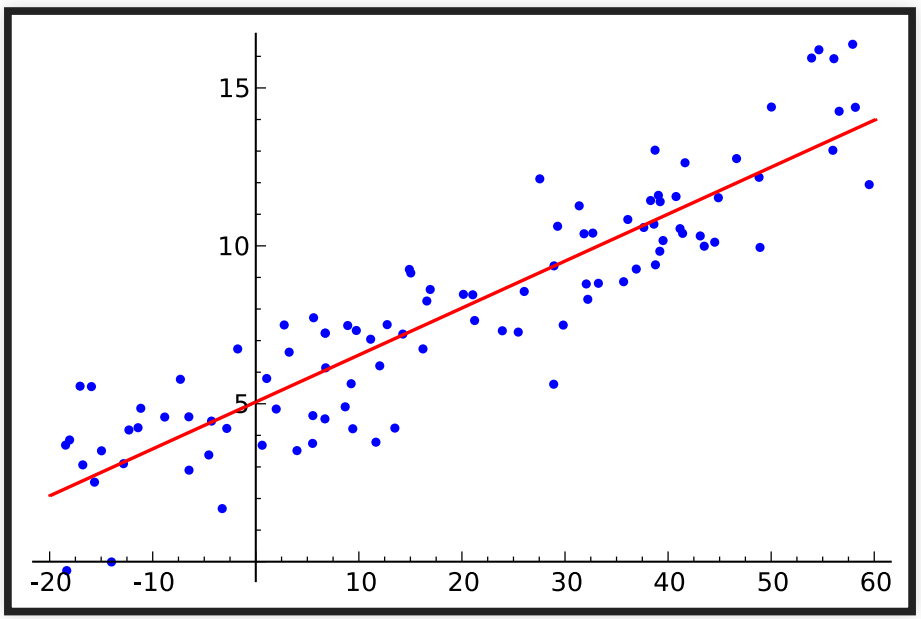
# COMMON DATASTRUCTURES: DATAFRAMES

- Tabular data, 2 dimensional
- Named columns
- Heterogeneous data: different types per column

```
>>> d = {'x1': [1, 2],
         'x2': ["overcast", "sunny"],
         'y': [True, False]}
>>> df = pd.DataFrame(data=d)
>>> df
   x1        x2      y
0   1  overcast   True
1   2     sunny  False
>>> df.dtypes
x1     int64
x2    object
y       bool
dtype: object
```

# LINEAR REGRESSION

$$f(x) = \alpha + \beta * x$$

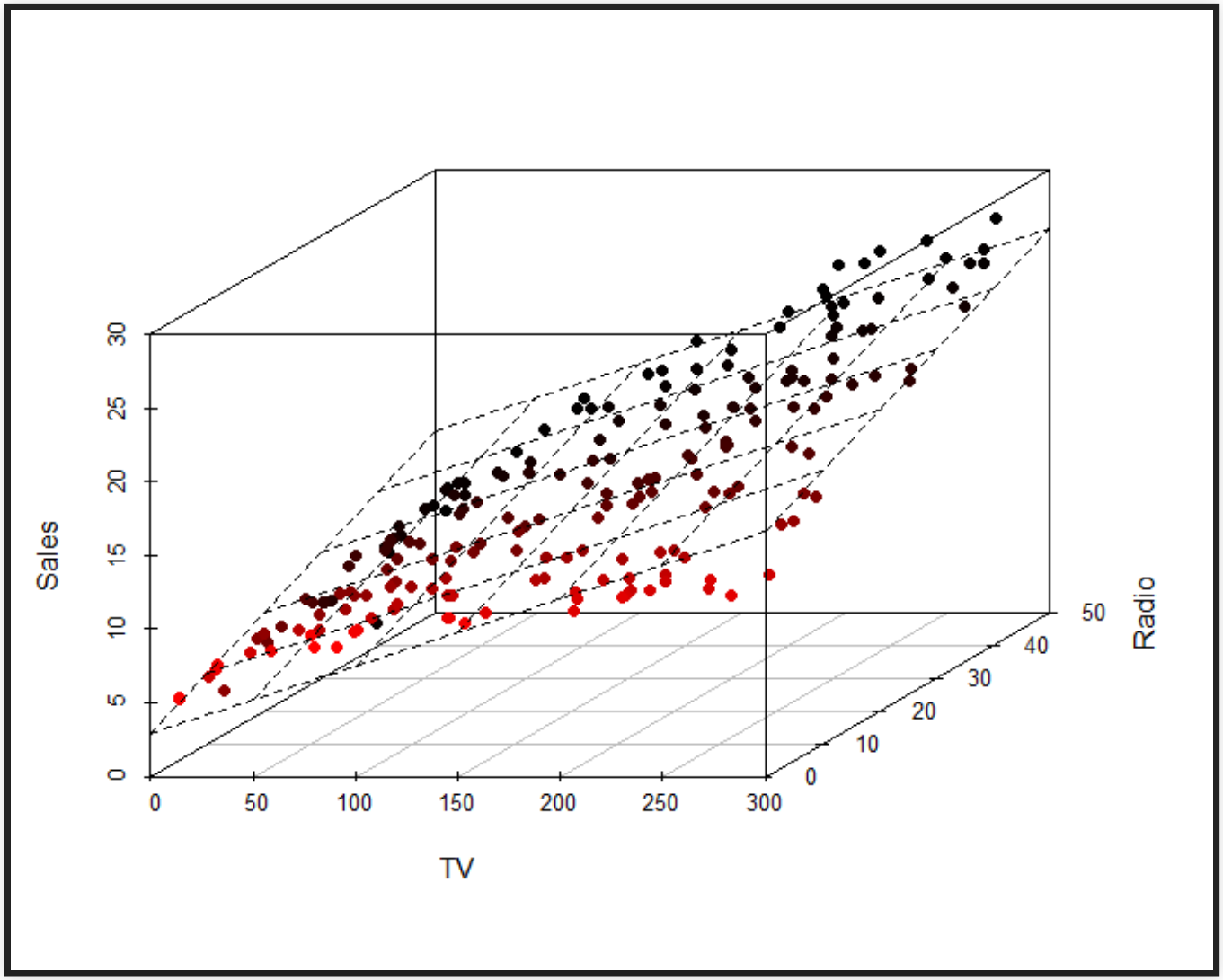


Speaker notes

One well known learning technique is linear regression. In the one-dimensional case, it simply fits a function $f(x) = \alpha + \beta * x$ to best explain all given data points (technically to minimize some error between f(x) and y across all given (x, y) pairs, e.g. sum of squared errors)

# LINEAR REGRESSION

$$f(x) = \alpha + \beta_1 * x_1 + \beta_2 * x_2 + \ldots + + \beta_n * x_n$$

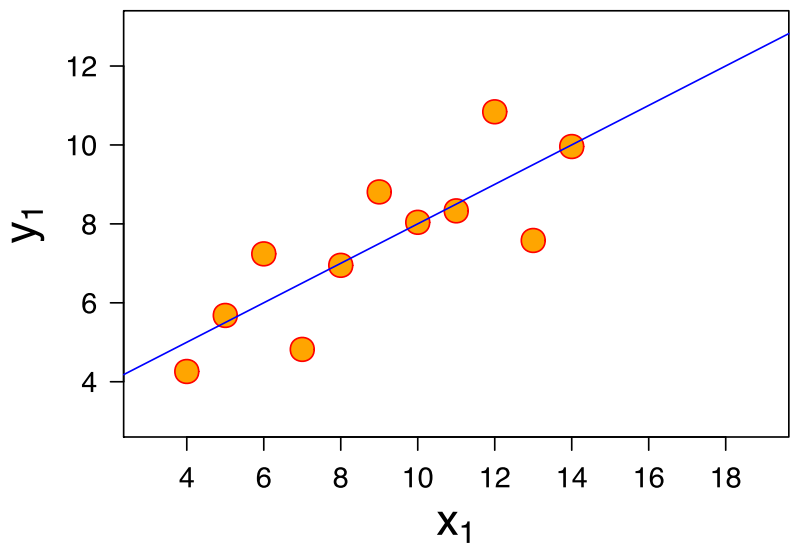


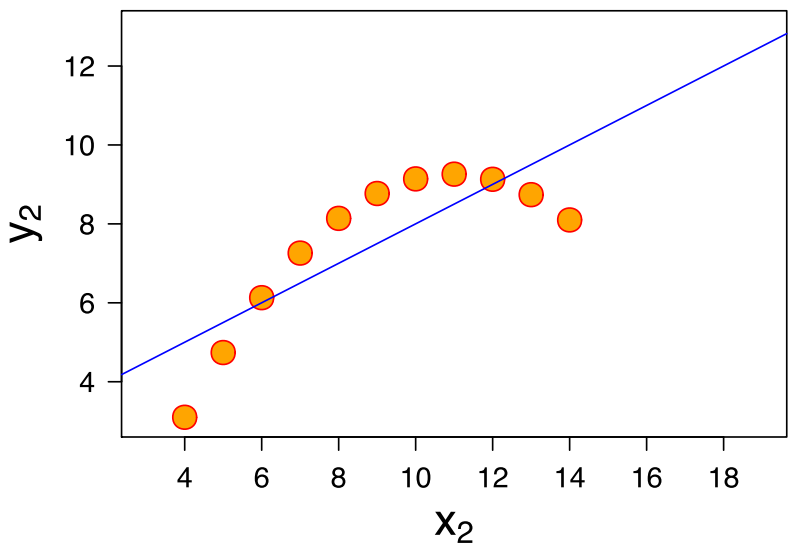

Graphic by ivanmp, CC BY-SA 3.0.

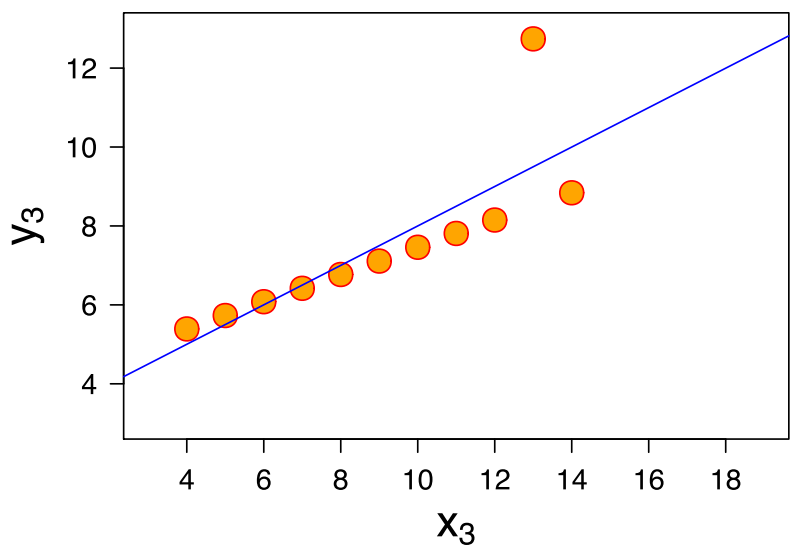
Speaker notes

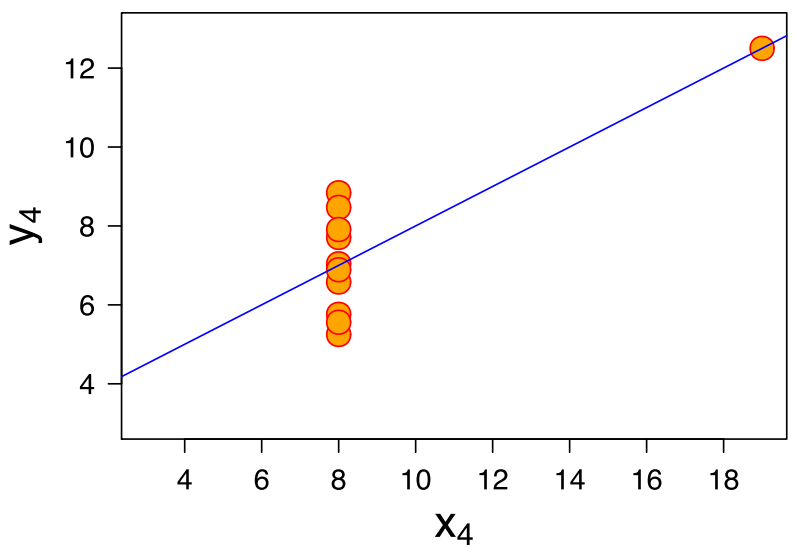
This generalizes to many dimensions (and also interactions), though this will be hard to visualize.

# DOES IT LEARN?

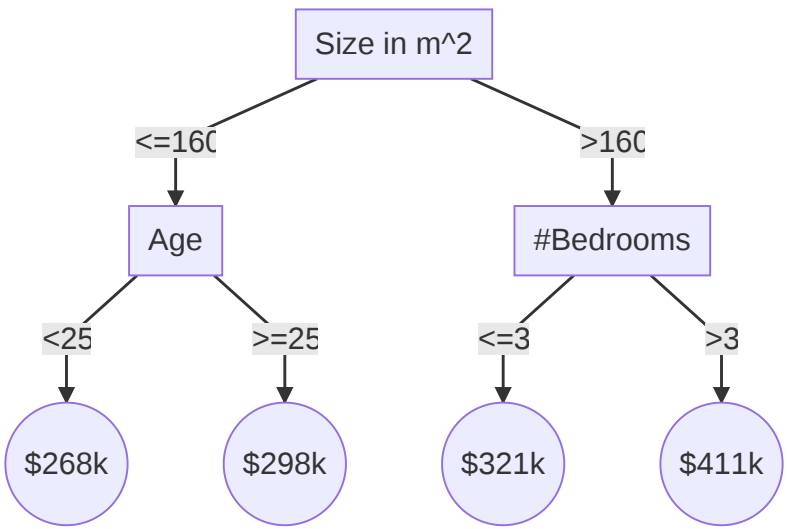

y1
x1
4
6
8
10
12
14
16
18
4
6
8
10
12


y2
x2
4
6
8
10
12
14
16
18
4
6
8
10
12


y3
x3
4
6
8
10
12
14
16
18
4
6
8
10
12


y4
x4
4
6
8
10
12
14
16
18
4
6
8
10
12

# LEARNING WITH DECISION TREES



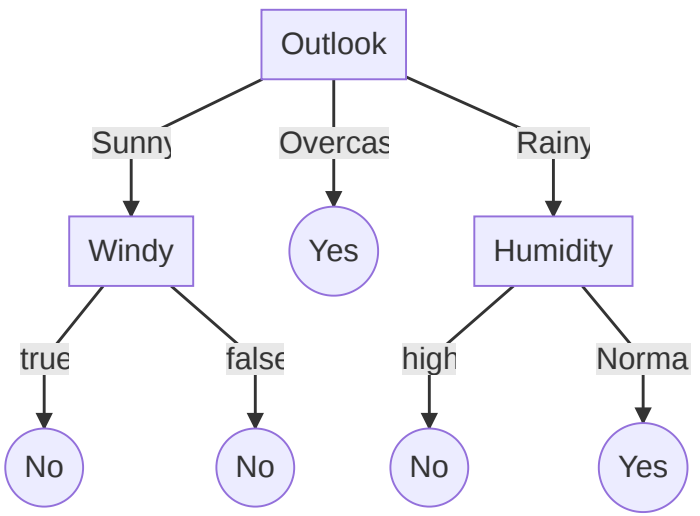
Speaker notes

We are using decision trees as an example of a simple and easy to understand learning algorithm. It is worth to understand at least one learning approach in some detail, to get an intutive sense of the functioning and limitations of machine learning. Also this example will illustrate the role of hyperparameters and how they relate to overfitting/underfitting.

# DECISION TREES

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| overcast | hot | high | false | yes |
| overcast | hot | high | false | no |
| overcast | hot | high | false | yes |
| overcast | cool | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| rainy | mild | normal | false | yes |
| rainy | mild | high | true | no |
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| sunny | mild | normal | true | yes |

f(Outlook, Temperature, Humidity, Windy) =



# BUILDING DECISION TREES

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| overcast | hot | high | false | yes |
| overcast | hot | high | false | no |
| overcast | hot | high | false | yes |
| overcast | cool | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| rainy | mild | normal | false | yes |
| rainy | mild | high | true | no |
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| sunny | mild | normal | true | yes |

**demo time**

# BUILDING DECISION TREES

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| overcast | hot | high | false | yes |
| overcast | hot | high | false | no |
| overcast | hot | high | false | yes |
| overcast | cool | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| rainy | mild | normal | false | yes |
| rainy | mild | high | true | no |
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| sunny | mild | normal | true | yes |

- Identify all possible decisions
- Select the decision that best splits the dataset into distinct outcomes (typically via entropy or similar measure)
- Repeatedly further split subsets, until stopping criteria reached

# OVERFITTING WITH DECISION TREES

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| overcast | hot | high | false | yes |
| overcast | hot | high | false | no |
| overcast | hot | high | false | yes |
| overcast | cool | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| rainy | mild | normal | false | yes |
| rainy | mild | high | true | no |
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| sunny | mild | normal | true | yes |

```
f(Outlook, Temperature, Humidity, Windy) =
  IF Humidity ∈ [high]
    IF Outlook ∈ [overcast,rainy]
      IF Outlook ∈ [overcast]
        IF Temperature ∈ [hot,cool]
          true (0.667)
          true (1.000)
        IF Windy ∈ [FALSE]
          true (1.000)
          false (1.000)
      false (1.000)
    IF Windy ∈ [FALSE]
      true (1.000)
      IF Temperature ∈ [hot,cool]
        IF Outlook ∈ [overcast]
          true (1.000)
          false (1.000)
        true (1.000)
```

The tree perfectly fits the data, except on overcast, hot and humid days without wind, where there is not enough data to distinguish 3 outcomes.

Not obvious that this tree will generalize well.

# UNDERFITTING WITH DECISION TREES

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| overcast | hot | high | false | yes |
| overcast | hot | high | false | no |
| overcast | hot | high | false | yes |
| overcast | cool | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| rainy | mild | normal | false | yes |
| rainy | mild | high | true | no |
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| sunny | mild | normal | true | yes |

```
f(Outlook, Temperature, Humidity, Windy) =
  IF Humidity ∈ [high]
    false (0.556)
    true (0.857)
```

If the model can only learn a single decision, it picks the best fit, but does not have enough freedom to make good predictions.

# OVERFITTING/UNDERFITTING

**Overfitting:** Model learned exactly for the input data, but does not generalize to unseen data (e.g., exact memorization)

**Underfitting:** Model makes very general observations but poorly fits to data (e.g., brightness in picture)

Typically adjust degrees of freedom during model learning to balance between overfitting and underfitting: can better learn the training data with more freedom (more complex models); but with too much freedom, will memorize details of the training data rather than generalizing

# ON TERMINOLOGY

- The decisions in a model are called *model parameter* of the model (constants in the resulting function, weights, coefficients), their values are usually learned from the data
- The parameters to the learning algorithm that are not the data are called *model hyperparameters*
- Degrees of freedom ~ number of model parameters

```
// max_depth and min_support are hyperparameters
def learn_decision_tree(data, max_depth, min_support): Model =
  ...

// A, B, C are model parameters of model f
def f(outlook, temperature, humidity, windy) =
   if A==outlook
      return B*temperature + C*windy > 10
```

# LEARNING FOR HOUSE PRICE ANALYSIS

| Crime Rate | %Large Lots | %Industrial | Near River | # Rooms | ... | Price |
|---|---|---|---|---|---|---|
| 0.006 | 18 | 2.3 | 0 | 6 | ... | 240.000 |
| 0.027 | 0 | 7.0 | 0 | 6 | ... | 216.000 |
| 0.027 | 0 | 7.0 | 0 | 7 | ... | 347.000 |
| 0.032 | 0 | 2.1 | 0 | 6 | ... | 334.000 |
| 0.069 | 0 | 2.1 | 0 | 7 | ... | 362.000 |

# IMPROVEMENTS

- Averaging across multiple trees (ensemble methods, including Boosting and Random forests) to avoid overfitting
    - building different trees on different subsets of the training data or basing decisions on different subsets of features
- Different decision selection criteria and heuristics, Gini impurity, information gain, statistical tests, etc
- Better handling of numeric data
- Extensions for graphs

# SUMMARY OF LEARNING WITH DECISION TREES

- Learning function fitting the data
- Generalizes to different decisions (categorical and numeric data) and different outcomes (classification and regression)
- Customizable **hyperparameter** (here: max tree height, min support, ...) to control learning process
- Many decisions influence qualities: accuracy/overfitting, learning cost, model size, ...
- Resulting models easy to understand and interpret (up to a size), mirroing human decision making procedures
- Scales fairly well to large datasets

# ON SPECIFICATIONS

No specification given for `f(outlook, temperature, humidity, windy)`

Learning `f` from data!

We do not expect perfect predictions; no possible model could always predict all training data correctly:

| **Outlook** | **Temperature** | **Humidity** | **Windy** | **Play** |
|---|---|---|---|---|
| overcast | hot | high | false | yes |
| overcast | hot | high | false | no |
| overcast | hot | high | false | yes |
| overcast | cool | normal | true | yes |

We are looking for models that *generalize well*

# EVALUATING MODELS (SUPERVISED LEARNING)

(more, much more on this later)

# BASIC APPROACH

Given labeled data, how well can the function predict the outcome labels?

Basic measure accuracy:

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

```
def accuracy(model, xs, ys):
  count = length(xs)
  countCorrect = 0
  for i in 1..count:
    predicted = model(xs[i])
    if predicted == ys[i]:
      countCorrect += 1
  return countCorrect / count
```

# SEPARATE TRAINING AND VALIDATION DATA

Always test for generalization on *unseen* validation data

Accuracy on training data (or similar measure) used during learning to find model parameters

```
train_xs, train_ys, valid_xs, valid_ys = split(all_xs, all_ys)
model = learn(train_xs, train_ys)

accuracy_train = accuracy(model, train_xs, train_ys)
accuracy_valid = accuracy(model, valid_xs, valid_ys)
```

accuracy_train >> accuracy_valid = sign of overfitting

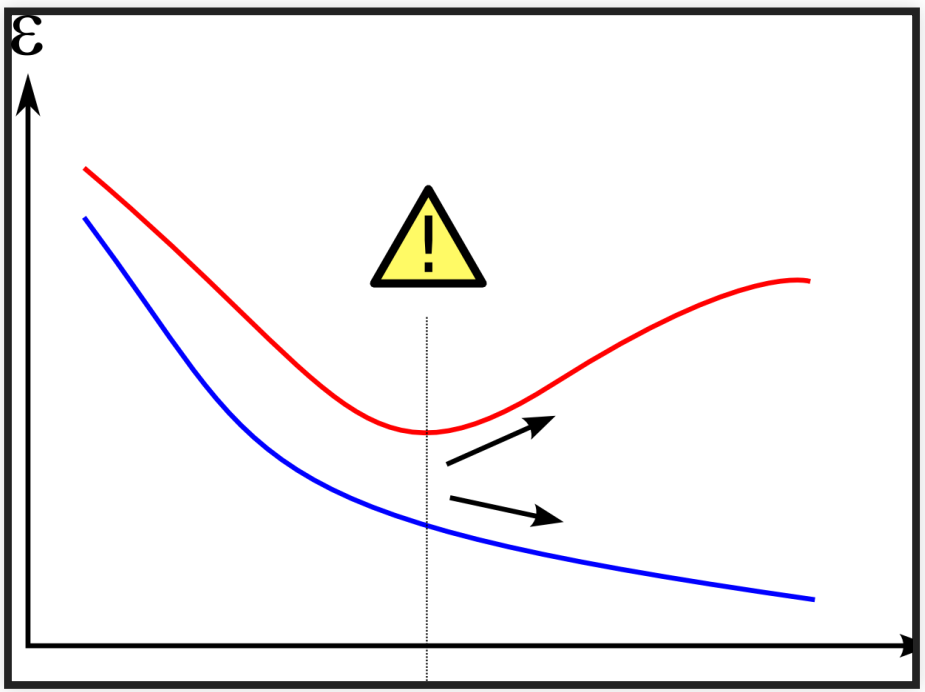
# RUNNING EXAMPLE: HOUSE PRICE ANALYSIS

Given data about a house and its neighborhood, what is the likely sales price for this house?

$$f(size, rooms, tax, neighborhood, \ldots) \rightarrow price$$

# DETECTING OVERFITTING

Change hyperparameter to detect training accuracy (blue)/validation accuracy (red) at different degrees of freedom




(CC SA 3.0 by Dake)

**demo time**

Speaker notes

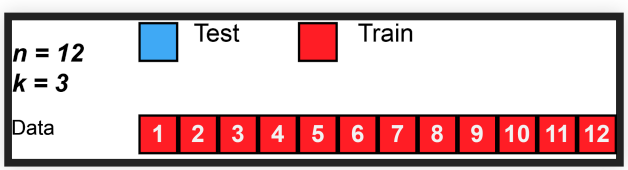
Overfitting is recognizable when performance of the evaluation set decreases.

Demo: Show how trees at different depth first improve accuracy on both sets and at some point reduce validation accuracy with small improvements in training accuracy

# CROSSVALIDATION

- Motivation
  - Evaluate accuracy on different training and validation splits
  - Evaluate with small amounts of validation data
- Method: Repeated partitioning of data into train and validation data, train and evaluate model on each partition, average results
- Many split strategies, including
  - leave-one-out: evaluate on each datapoint using all other data for training
  - k-fold: $k$ equal-sized partitions, evaluate on each training on others
  - repeated random sub-sampling (Monte Carlo)

**demo time**




(Graphic CC MBanuelos22 BY-SA 4.0)

# SEPARATE TRAINING, VALIDATION AND TEST DATA

Often a model is "tuned" manually or automatically on a validation set (hyperparameter optimization)

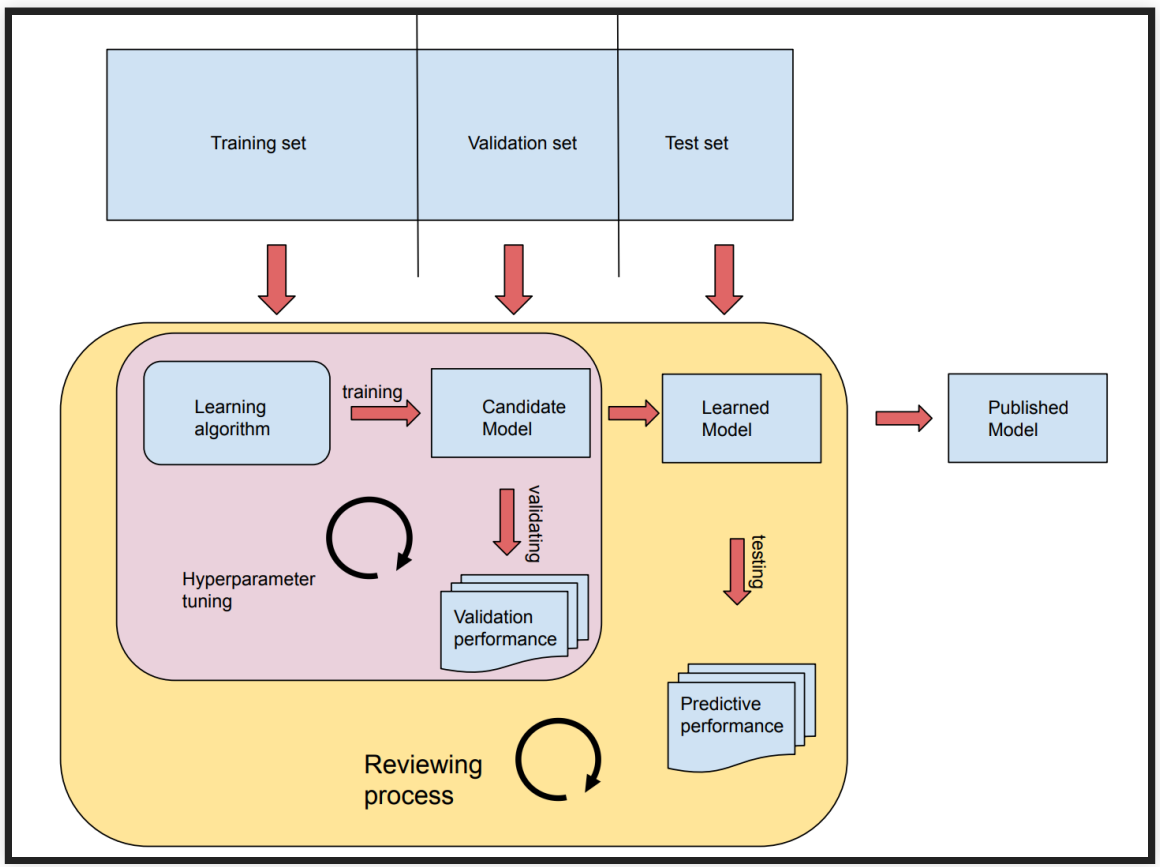
In this case, we can overfit on the validation set, separate test set is needed for final evaluation

```
train_xs, train_ys, valid_xs, valid_ys, test_xs, test_ys =
            split(all_xs, all_ys)

best_model = null
best_model_accuracy = 0
for (hyperparameters in candidate_hyperparameters)
  candidate_model = learn(train_xs, train_ys, hyperparameter)
  model_accuracy = accuracy(model, valid_xs, valid_ys)
  if (model_accuracy > best_model_accuracy)
    best_model = candidate_model
    best_model_accuracy = model_accuracy

accuracy_test = accuracy(model, test_xs, test_ys)
```

# ACADEMIC ESCALATION: OVERFITTING ON BENCHMARKS

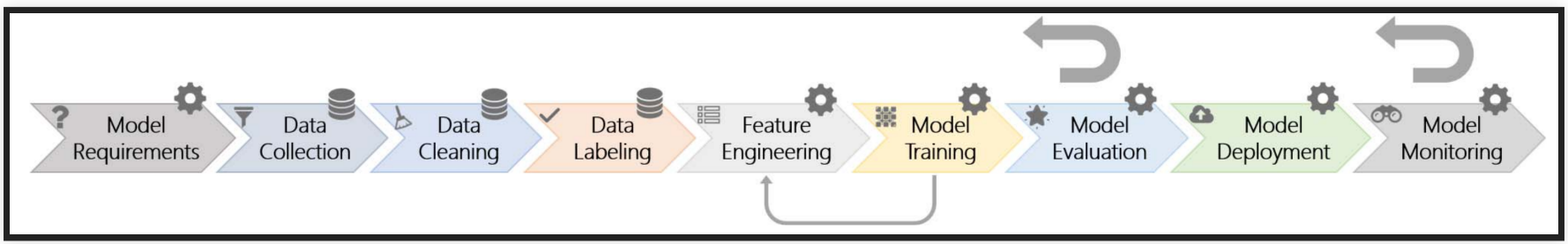



(Figure by Andrea Passerini)

Speaker notes

If many researchers publish best results on the same benchmark, collectively they perform "hyperparameter optimization" on the test set

# MACHINE LEARNING PIPELINE

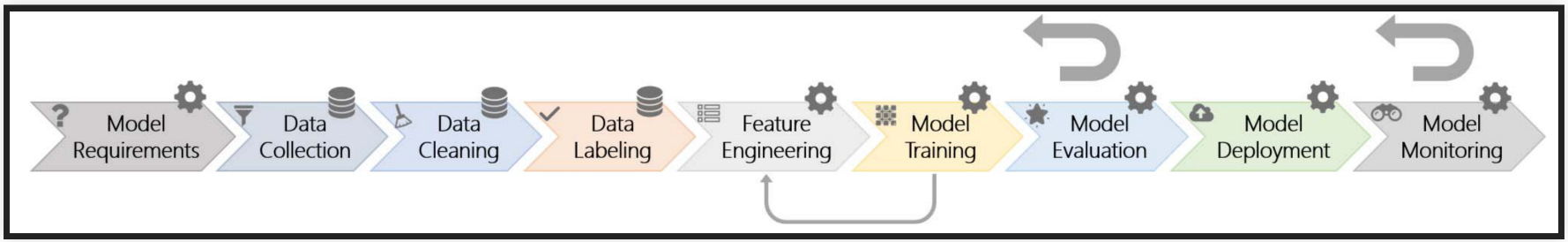



Graphic: Amershi, Saleema, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. "Software engineering for machine learning: A case study." In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291-300. IEEE, 2019.

Speaker notes

An average data scientist spends most of their time collecting and cleaning data.

# EXAMPLE




$$f(size, rooms, tax, neighborhood, \ldots) \rightarrow price$$

Speaker notes

Go through all the pipeline steps and discuss how they might look like in this example. What data would be collected, what cleaning might be needed, how might data be labled, what features are selected?

# PIPELINE STEPS

- Data collection: identify training data, often many sources
- Data cleaning: remove wrong data, outliers, merge data from multiple sources
- Data labeling: identify labels ($Y$) on training data, possibly croudsourced or (semi-)automated
- Feature engineering: convert raw data into a form suitable for learning, identifying features, encoding, normalizing
- Model training: build the model, tune hyperparameters
- Model evaluation: determine fitness for purpose

- Data science education focuses on feature engineering and model training
- Data science practitioners spend substantial time collecting and cleaning data
- Requirements, deployment, and monitoring rarely a focus in data science education

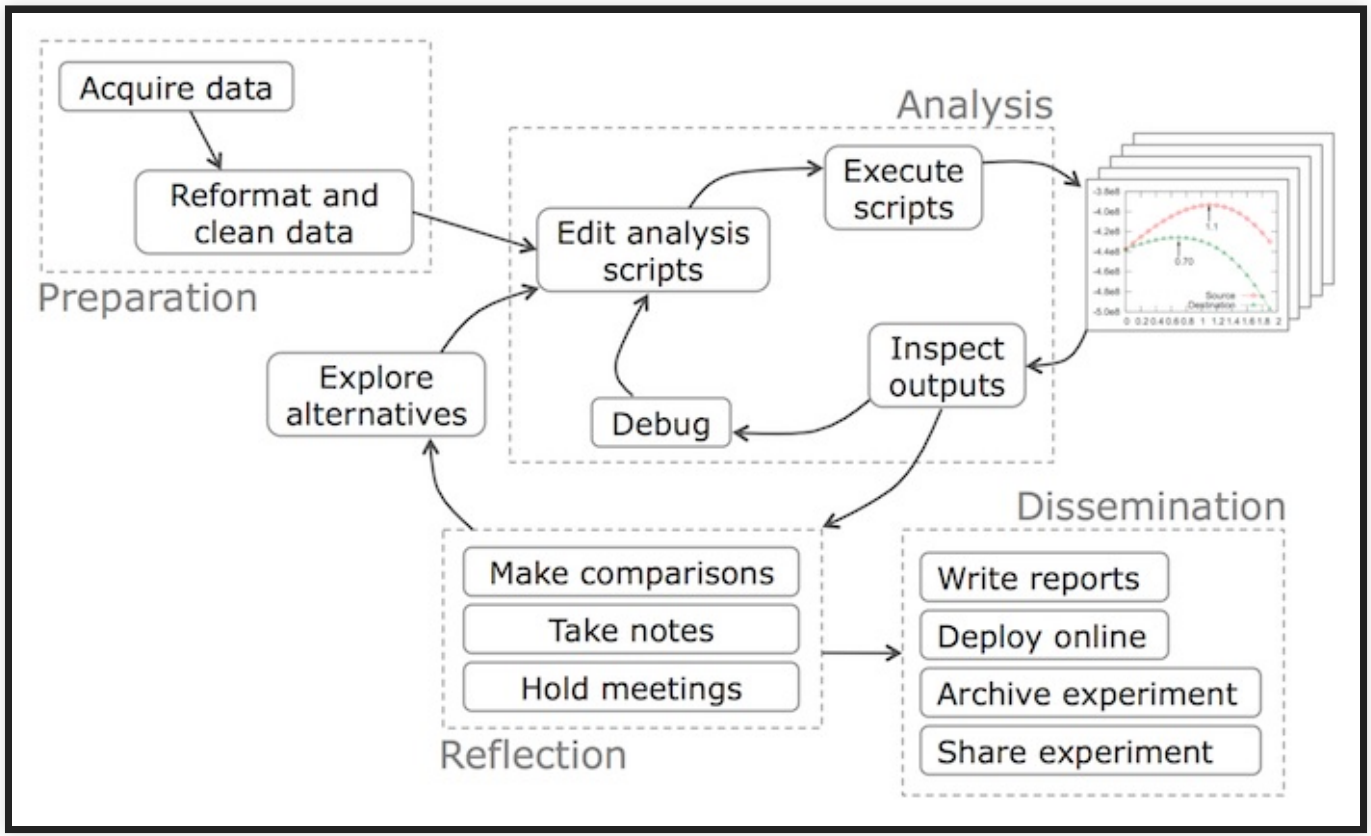
# COMMON CHALLENGES

- Insufficient Quantity of Training Data
- Nonrepresentative Training Data
- Poor Data Quality
- Irrelevant Features
- Overfitting, Underfitting
- Data + Model Match

Suggested complementary reading: Géron, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow", 2nd Edition (2019), Ch 1.

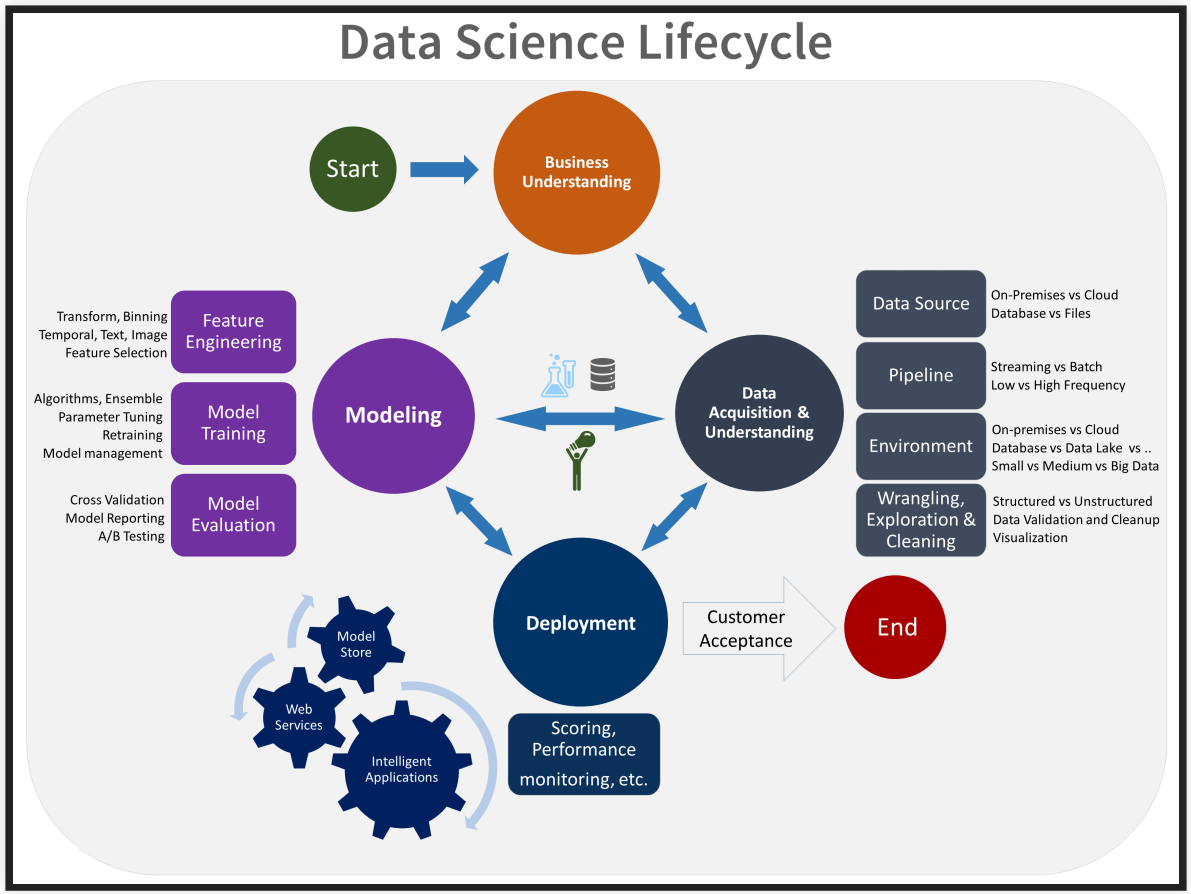
# ITERATION AND EXPLORATION WITH COMPUTATIONAL NOTEBOOKS

# DATA SCIENCE IS ITERATIVE AND EXPLORATORY



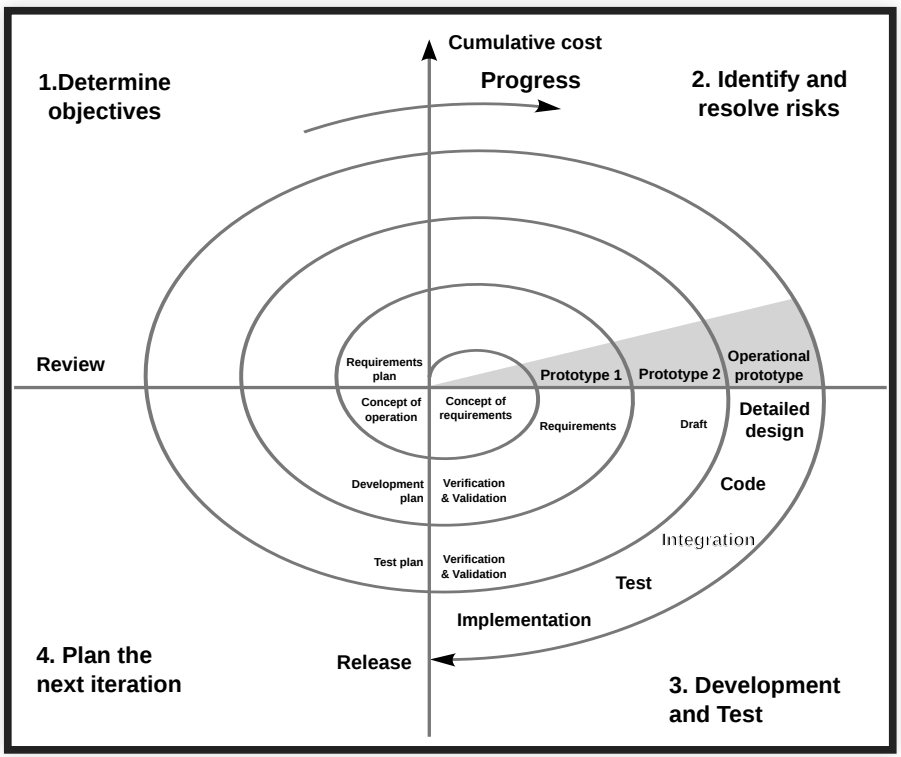

(Source: Guo. "Data Science Workflow: Overview and Challenges." Blog@CACM, Oct 2013)

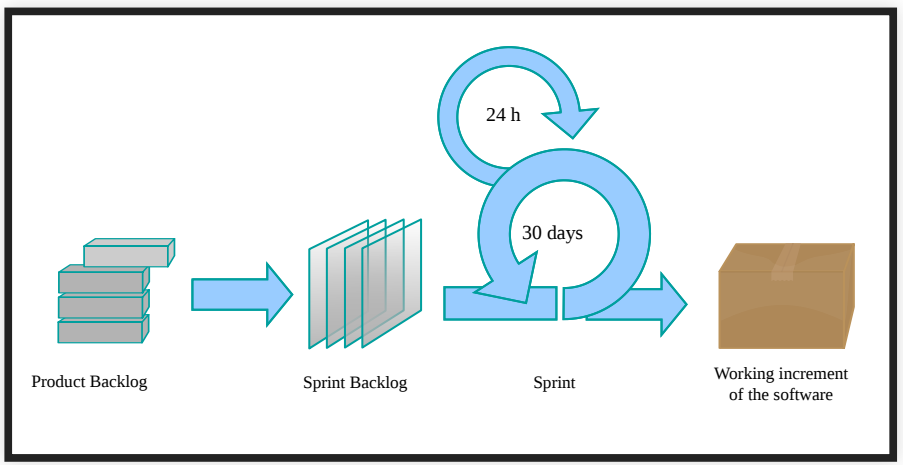
# DATA SCIENCE IS ITERATIVE AND EXPLORATORY




(Microsoft Azure Team, "What is the Team Data Science Process?" Microsoft Documentation, Jan 2020)

# SIMILAR TO SPIRAL PROCESS MODEL OR AGILE?

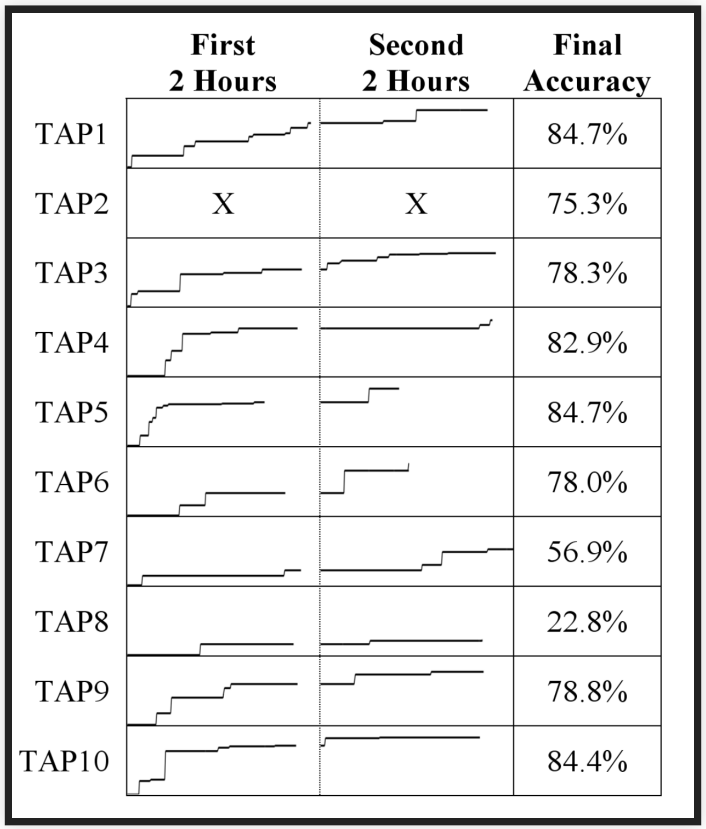


Speaker notes

There is similarity in that there is an iterative process, but the idea is different and the process model seems mostly orthogonal to iteration in data science. The spiral model prioritizes risk, especially when it is not clear whether a model is feasible. One can do similar things in model development, seeing whether it is feasible with data at hand at all and build an early prototype, but it is not clear that an initial okay model can be improved incrementally into a great one later. Agile can work with vague and changing requirements, but that again seems to be a rather orthogonal concern. Requirements on the product are not so much unclear or changing (the goal is often clear), but it's not clear whether and how a model can solve it.

# DATA SCIENCE IS ITERATIVE AND EXPLORATORY




Source: Patel, Kayur, James Fogarty, James A. Landay, and Beverly Harrison. "Investigating statistical machine learning as a tool for software development." In Proc. CHI, 2008.

Speaker notes

This figure shows the result from a controlled experiment in which participants had 2 sessions of 2h each to build a model. Whenever the participants evaluated a model in the process, the accuracy is recorded. These plots show the accuracy improvements over time, showing how data scientists make incremental improvements through frequent iteration.

# DATA SCIENCE IS ITERATIVE AND EXPLORATORY

- Science mindset: start with rough goal, no clear specification, unclear whether possible
- Heuristics and experience to guide the process
- Try and error, refine iteratively, hypothesis testing
- Go back to data collection and cleaning if needed, revise goals

# EXPLORATION AND ITERATION IN THE RUNNING EXAMPLE

Given data about a house and its neighborhood, what is the likely sales price for this house?

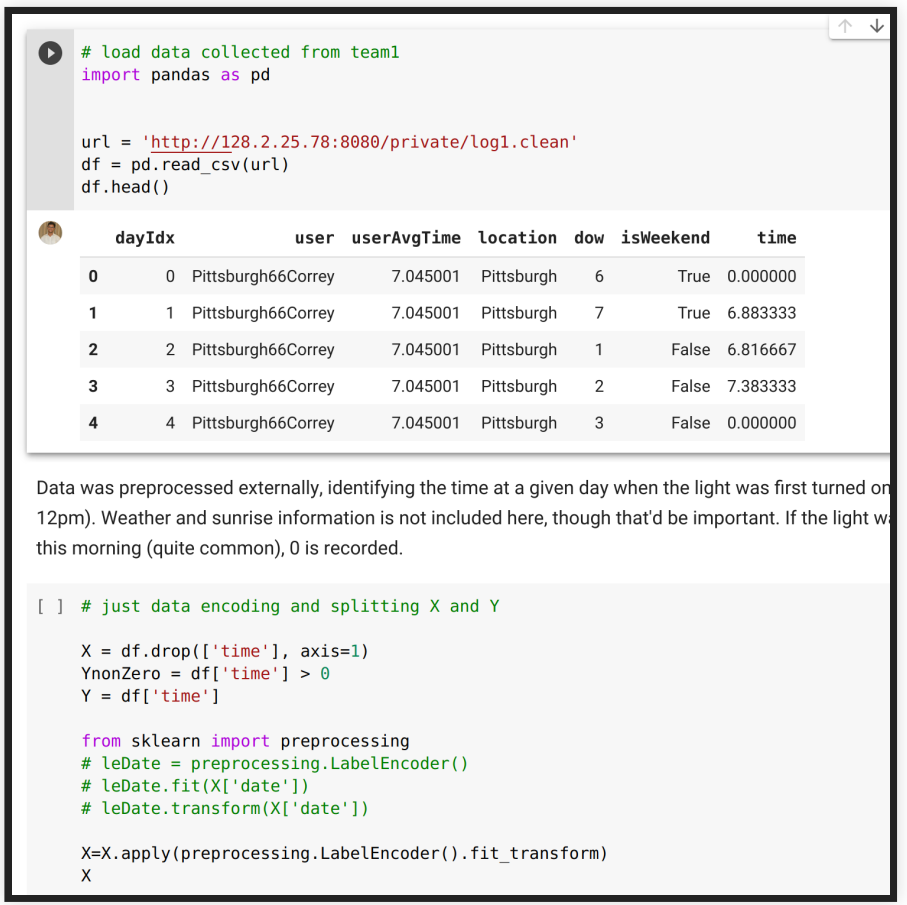
$$f(size, rooms, tax, neighborhood, \ldots) \rightarrow price$$

# SHARE EXPERIENCE?

# COMPUTATIONAL NOTEBOOKS

- Origins in "literal programming", interleaving text and code, treating programs as literature (Knuth'84)
- First notebook in Wolfram Mathematica 1.0 in 1988
- Document with text and code cells, showing execution results under cells
- Code of cells is executed, per cell, in a kernel
- Many notebook implementations and supported languages, Python + Jupyter currently most popular

**demo time**



```
# load data collected from team1
import pandas as pd

url = 'http://128.2.25.78:8080/private/log1.clean'
df = pd.read_csv(url)
df.head()
```

| | dayIdx | user | userAvgTime | location | dow | isWeekend | time |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Pittsburgh66Correy | 7.045001 | Pittsburgh | 6 | True | 0.000000 |
| 1 | 1 | Pittsburgh66Correy | 7.045001 | Pittsburgh | 7 | True | 6.883333 |
| 2 | 2 | Pittsburgh66Correy | 7.045001 | Pittsburgh | 1 | False | 6.816667 |
| 3 | 3 | Pittsburgh66Correy | 7.045001 | Pittsburgh | 2 | False | 7.383333 |
| 4 | 4 | Pittsburgh66Correy | 7.045001 | Pittsburgh | 3 | False | 0.000000 |

Data was preprocessed externally, identifying the time at a given day when the light was first turned on
12pm). Weather and sunrise information is not included here, though that'd be important. If the light w
this morning (quite common), 0 is recorded.

```
[ ] # just data encoding and splitting X and Y

    X = df.drop(['time'], axis=1)
    YnonZero = df['time'] > 0
    Y = df['time']

    from sklearn import preprocessing
    # leDate = preprocessing.LabelEncoder()
    # leDate.fit(X['date'])
    # leDate.transform(X['date'])

    X=X.apply(preprocessing.LabelEncoder().fit_transform)
    X
```

Speaker notes

- See also https://en.wikipedia.org/wiki/Literate_programming
- Demo with public notebook, e.g., https://colab.research.google.com/notebooks/mlcc/intro_to_pandas.ipynb

# NOTEBOOKS SUPPORT ITERATION AND EXPLORATION

- Quick feedback, similar to REPL
- Visual feedback including figures and tables
- Incremental computation: reexecuting individual cells
- Quick and easy: copy paste, no abstraction needed
- Easy to share: document includes text, code, and results

# BRIEF DISCUSSION: NOTEBOOK LIMITATIONS AND DRAWBACKS?

(later more)

# SUMMARY

- Key concepts in machine learning: dataframes, model, train/validation/test data
- A simple machine-learning algorithm: Decision trees
- Overfitting, underfitting, hyperparameter tuning
- Basic model accuracy measures and crossvalidation
- Steps of a machine learning pipeline
- Introduction to working with computational notebooks, typical iterative workflow, benefits and limitations of notebooks