Data Science Fundamentals

Lecture 1. Introduction

Fernando Lovera
Mario Verstraeten

Introduction

Agenda

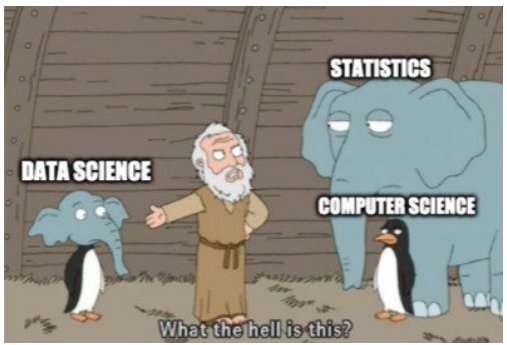
- The ascendance of data
- What is Data Science?
- Motivating hypothetical datascientist (Finding key connectors, datascientis you may know, salaries and experiences)
- Practicalities (Course organization, material, evaluation, outline)
- What is AI, Machine Learning & Deep Learning
- Recap Data Cleaning & Visualisation project

The ascendance of Data

- We live in a world that's full of data and people chasing data. But WHY... because it's valuable, it can tell you the future or at least a possible future. Data can come from different places, for example:
  - Websites that track clicks of users.
  - Smartphones building records of your locations.
  - Smart cars collect driving habits, smart homes collect living habits and smart marketers collect purchasing habits.
- Buried in this data are answers to countless questions that no one's ever thought to ask.

What is Datascience

There is a joke that says a data scientis is someone who knows more statistics than a computer scientist and more computer science than a statistician.



Data Science is a scientific field which combines techniques from computer science and statistics to provide meaningful insights from large amounts of complex data, and the means to put these insights into practice.

What is Data Science?

No matter how you define Data Science, there will be people coming from many different fields.

One thing is clear though, a data scientist is someone who extracts insights from data, regardless of the status of the data.

For instance, Facebook asks you to list your hometown and your current location, this makes it easier for your friends to find and connect with you. But it also analyses the locations to identify global migration patterns.

What is Data Science?

Another example happened with a large retailer: Target. Target tracks your purchases and interactions, both online and in store. And it uses the data to predictively model which of its customers are pregnant, to better market baby-related purchases to them.

In 2012, the Obama campaign employed dozens of data scientists who data-mined and experimented their way to identifying voters who needed extra attention, choosing optimal donor-specific fundraising appeals and programs. In 2016 the Trump campaign tested a staggering variety of online ads and analyzed the data to find what worked and what didn't.

Some Data science tasks:

It's your first day on the job as a data scientist and the VP of Networking is full of questions about your users. Until now he's had no one to ask, so he's very excited to have you onboard.

He wants you to identify who the "key connectors" are among data scientists. To this end, he gives you a dump of the entire datascience network. This dump of data looks like the following:

```
users = [
        {"id": 0, "name": "Hero"},{"id": 1, "name": "Dunn"},
        {"id": 2, "name": "Sue"},{"id": 3, "name": "Chi"},
        {"id": 4, "name": "Thor"},{"id": 5, "name": "Clive"},
        {"id": 6, "name": "Hicks"},{"id": 7, "name": "Devin"},
        {"id": 0, "name": "Kate"},{"id": 0, "name": "Klein"},
]
```

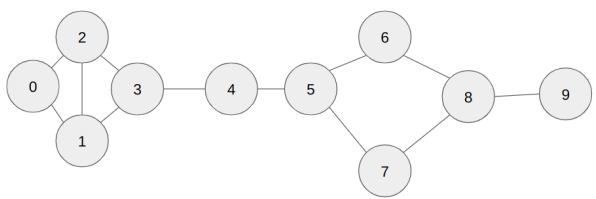
Some Data science tasks:

He also gives you the "friendship" data, represented as a list of pairs of IDs:

```
friendship_pairs = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
                                    (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

The tuple (0, 1) indicates that the data scientist with id 0 (Hero) and the data scientist with id 1 (Dunn) are friends.

This can seen as a graph as the follow data science graph.



Finding key connectors

Having friends represented as a list of pairs is not the easiest way to work with them. To find all the friendships for user 1, you have to iterate over every pair looking for pairs containing 1. If you had a lot of pairs, this would take a long time.

So, let's create a dictionary where the keys are user ids and the values are list of friends ids. We'll still have to look at every pair to create the dictionary, but we only have to do that once, and we'll get cheap lookups after that:

```
1 # initialize the dict with an empty list for each user id:
2 friendship = {user["id"]: [] for user in users}
3
4 #loop over the friendship pairs to populate it:
5 for i, j in friendship_pairs:
6   friendships[i].append(j)      # add j as a friend of user i
7   friendships[j].append(i)      # add i as friend of user j
```

Finding key connectors

Now that we have the friendships in a dictionary, we can easily ask questions of our graph, like: "What's the average number of connections?"

First, we find the total number of connections, by summing up the lengths of all the friends lists:

```
def number_of_friends(user):
  #"How many friends does user have?"
  user_id = user["id"]
  friends_ids = friendships[user_id]

  return len(friend_ids)

total_connections = sum(number_of_friends(user) for user in users) # 24
```

FInally, we divide by the number of users:

```
num_users = len(users)
avg_connections = total_connections / num_users # 24 / 10 == 2.4
```

Finding key connectors

It's also easy to find the most connected people—they're the people who have the largest number of friends. Since there aren't very many users, we can sort them from "most friends" to "least friends"

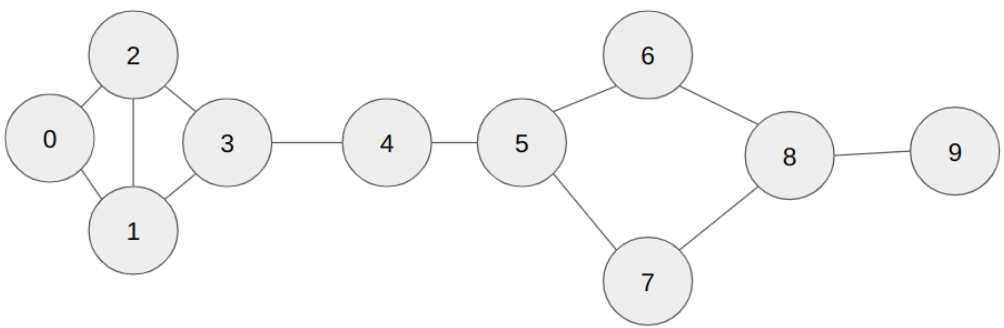
```
# create a list (user_id, number_of_friends)
num_friends_by_id = [(user["id"], number_of_friends(user)) for user in users]

sorted(num_friends_by_id, # get it sorted
 key=lambda (user_id, num_friends): num_friends, # by num_friends
 reverse=True) # largest to smallest

# each pair is (user_id, num_friends)
# [(1, 3), (2, 3), (3, 3), (5, 3), (8, 3),
# (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]
```

Finding key connectors

This has the virtue of being pretty easy to calculate, but it doesn't always give the results you'd want or expect. For example, in the network Thor (id 4) only has two connections while Dunn (id 1) has three. Yet looking at the network it intuitively seems like Thor should be more central.



Finding key connectors- Data

Scientisis you may know

The VP now wants to encourage more connections among your members, and she asks you to design a “Data Scientists You May Know” suggester. Your first try friends of friends: for each of a user’s friends, iterate over that person’s friends, and collect all the results:

```
1 def friends_of_friend_ids_bad(user):
2         # "foaf" is short for "friend of a friend"
3         return [foaf["id"]
4  for friend in user["friends"] # for each of user's friends
5  for foaf in friend["friends"]] # get each of _their_ friends
```

When we call this on users[0] (Hero), it produces:

```
1 [0, 2, 3, 0, 1, 3]
```

This response includes duplicates. WHY is that?!

Datascientis you may know

We definitely should use a helper function to exclude people already known to the user:

```
from collections import Counter # not loaded by default
def not_the_same(user, other_user):
    """two users are not the same if they have different ids"""
    return user["id"] != other_user["id"]
def not_friends(user, other_user):
    """
    other_user is not a friend if he's not in user["friends"];
    that is, if he's not_the_same as all the people in user["friends"]
    """
        return all(not_the_same(friend, other_user)
    for friend in user["friends"])
def friends_of_friend_ids(user):
        return Counter(foaf["id"]
 for friend in user["friends"] # for each of my friends
 for foaf in friend["friends"] # count *their* friends
 if not_the_same(user, foaf) # who aren't me
 and not_friends(user, foaf)) # and aren't my friends
print friends_of_friend_ids(users[3]) # Counter({0: 2, 5: 1})
```

Finding key connectors

This correctly tells Chi (id 3) that she has two mutual friends with Hero (id 0) but only one mutual friend with Clive (id 5). As a data scientist, you know that you also might enjoy meeting users with similar interests.

```
# call for example the functic
pairs(user_id, interest)
```

You get the following data:

```
interests = [
 (0, "Hadoop"), (0, "Big Data"), (0, "HBase"), (0, "Java"),
 (0, "Spark"), (0, "Storm"), (0, "Cassandra"),
 (1, "NoSQL"), (1, "MongoDB"), (1, "Cassandra"), (1, "HBase"),
 (1, "Postgres"), (2, "Python"), (2, "scikit-learn"), (2, "scipy"),
 (2, "numpy"), (2, "statsmodels"), (2, "pandas"), (3, "R"), (3, "Python
 (3, "statistics"), (3, "regression"), (3, "probability"),
 (4, "machine learning"), (4, "regression"), (4, "decision trees"),
 (4, "libsvm"), (5, "Python"), (5, "R"), (5, "Java"), (5, "C++"),
 (5, "Haskell"), (5, "programming languages"), (6, "statistics"),
 (6, "probability"), (6, "mathematics"), (6, "theory"),
 (7, "machine learning"), (7, "scikit-learn"), (7, "Mahout"),
 (7, "neural networks"), (8, "neural networks"), (8, "deep learning"),
 (8, "Big Data"), (8, "artificial intelligence"), (9, "Hadoop"),
 (9, "Java"), (9, "MapReduce"), (9, "Big Data")
]
```

Finding key connectors

```
def data_scientists_who_like(target_interest):
        return [user_id
for user_id, user_interest in interests if user_interest == target_interest]
```

We can write better code that does the same as follows:

```
from collections import defaultdict
# keys are interests, values are lists of user_ids with that interest
user_ids_by_interest = defaultdict(list)
for user_id, interest in interests:
        user_ids_by_interest[interest].append(user_id)
# keys are user_ids, values are lists of interests for that user_id
interests_by_user_id = defaultdict(list)
for user_id, interest in interests:
        interests_by_user_id[user_id].append(interest)
```

Finding key connectors... but what about salaries vs expertise

If you have the data of salary (salary and years of experience) as follows:

```
salaries_and_tenures = [(83000, 8.7), (88000, 8.1),
 (48000, 0.7), (76000, 6),
 (69000, 6.5), (76000, 7.5),
 (60000, 2.5), (83000, 10),
 (48000, 1.9), (63000, 4.2)]
```

People with more experience tend to earn more.

Your first idea is to look at the average salary for each tenure:

```
# keys are years, values are lists of the salaries for each tenure
salary_by_tenure = defaultdict(list)
for salary, tenure in salaries_and_tenures:
        salary_by_tenure[tenure].append(salary)
# keys are years, each value is average salary for that tenure
average_salary_by_tenure = {
        tenure : sum(salaries) / len(salaries)
        for tenure, salaries in salary_by_tenure.items()
}
```

Finding key connectors... but what about salaries vs expertise

This turns out to be not particularly useful, as none of the users have the same tenure (experience), which means we're just reporting the individual users' salaries.

It might be more helpful to bucket the tenures:

```
def tenure_bucket(tenure):
        if tenure < 2:
                return "less than two"
        elif tenure < 5:
                return "between two and five"
        else:
                return "more than five"

# keys are tenure buckets, values are lists of salaries for that bucket
salary_by_tenure_bucket = defaultdict(list)
for salary, tenure in salaries_and_tenures:
        bucket = tenure_bucket(tenure)
        salary_by_tenure_bucket[bucket].append(salary)
# keys are tenure buckets, values are average salary for that bucket
average_salary_by_bucket = {
 tenure_bucket : sum(salaries) / len(salaries)
 for tenure_bucket, salaries in salary_by_tenure_bucket.iteritems()
}
```

Finding key connectors... but what about salaries vs expertise

```
# which results in something more interesting
{'between two and five': 61500.0,
 'less than two': 48000.0,
 'more than five': 79166.66666666667}
```

Practicalities of the course

- Course Organization
- Course Material
- Course Evaluation
- Course Outline

Practicalities

- Course organization
  - 3 hours lecture per week
  - Theory & Exercises
  - Attendance is (mostly) not compulsory, but strongly advised!
  - Behave yourself!
  - **Evaluation:** 100% Oral exam according to the ECTS

Practicalities

- Course material
  - Compulsory: Slides (on Canvas)
  - Preparation notebooks (on Canvas)
  - Exercises (on Canvas)
  - Anaconda & Editor of choice Optional
  - Additional Documents (Canvas)
  - Books:
    - Python for Data Analysis, Wes McKinney
    - Python Data Science Handbook, Jake VanderPlas
    - Hands-on Machine Learning with (...), Aurelien Geron

Practicalities

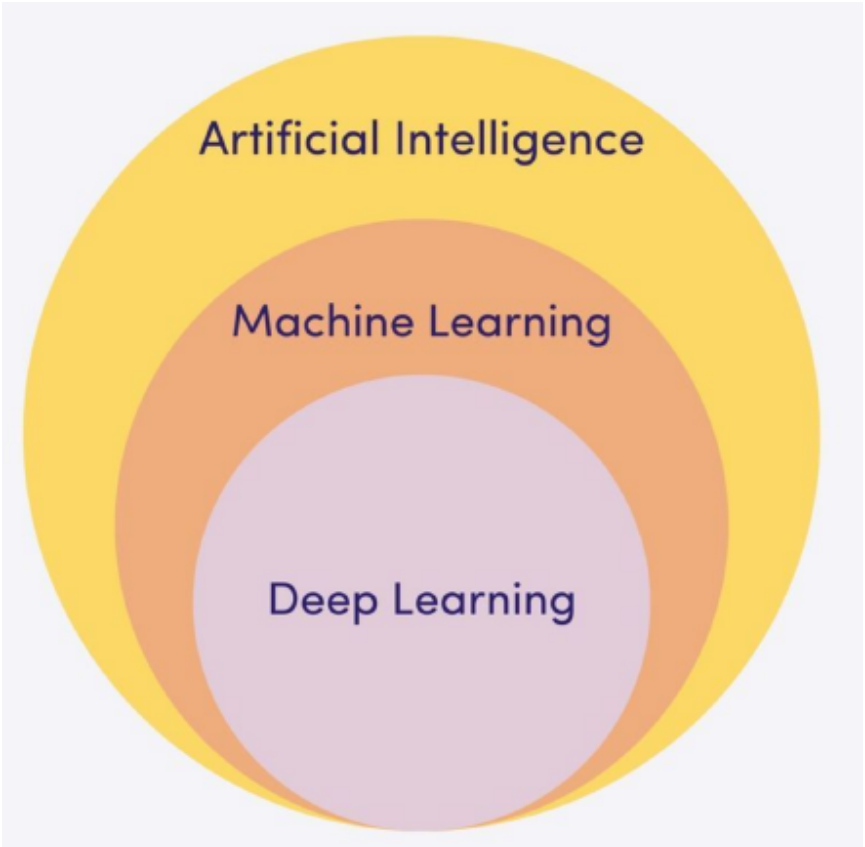
- Course development
    - Week 1 Introduction + Recap Data Exploration
    - Week 2 Regression 1: Linear Regression
    - Week 3 Regression 2: Polynomial Regression
    - Week 4 Feature Engineering + Cross-validation
    - Week 5 Classification 1: Logistic Regression
    - Week 6 Performance Evaluation (ROC)
    - Week 7 Classification 2: kNN

Practicalities

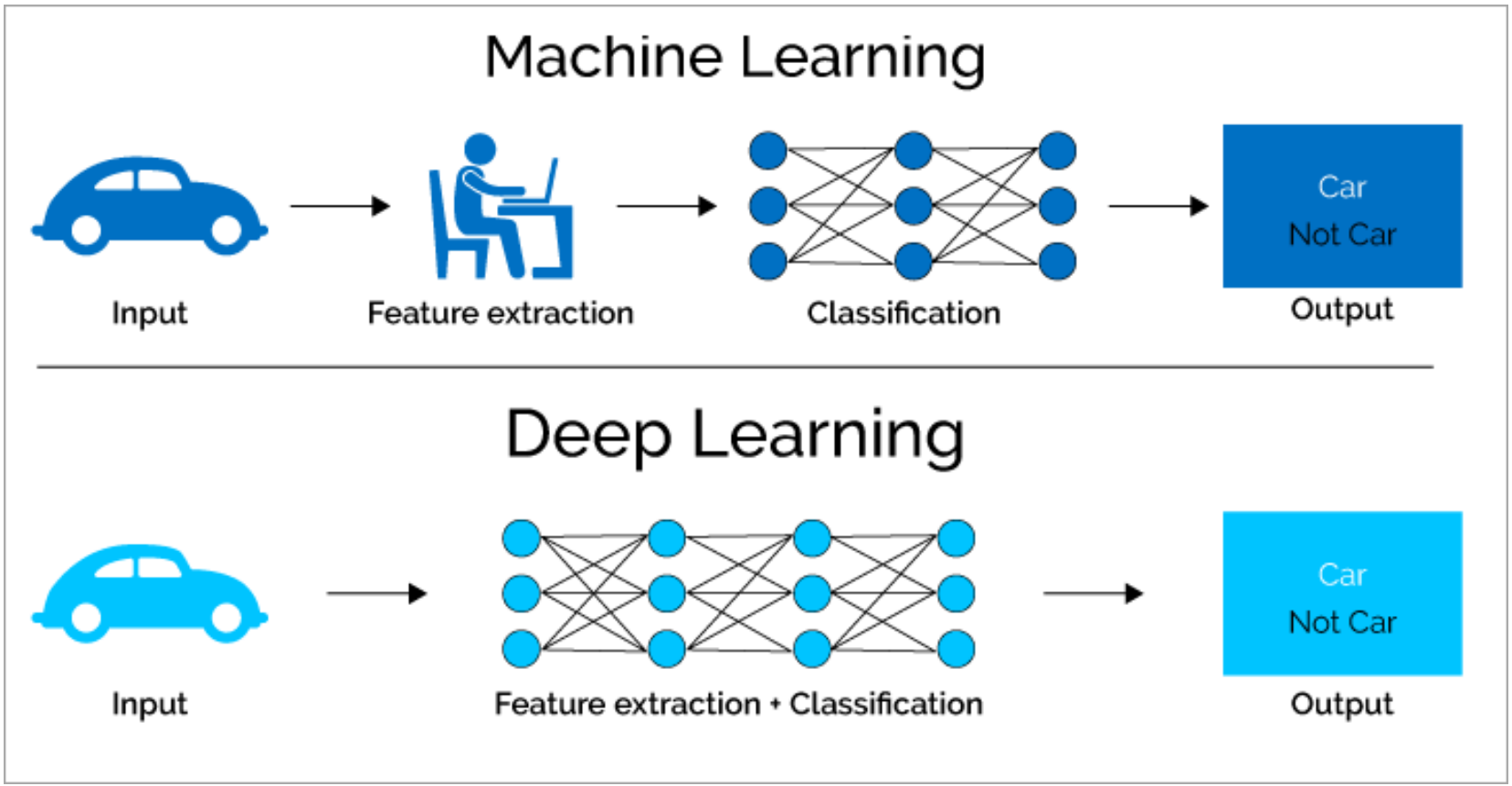
- Course development
    - Week 8 Clustering 1: k-Means
    - Week 9 Clustering 2: Hierarchical Clustering + DBSCAN
    - Week 10 Dimensionality Reduction
    - Week 11 Model Deployment
    - Week 12 Recap Project
    - Week 13 Exam Preparation

What is Machine Learning?

What is AI, Machine Learning & Deep Learning?



What is AI, Machine Learning & Deep Learning?



What is Machine learning?

Machine Learning is an approach to achieve artificial intelligence through systems that can learn from experience to find patterns in a set of data.

- It relies on teaching a computer to recognize patterns by example, rather than programming it with specific rules
- A way to make predictions
  - Takes in data
  - Learns patterns from said data
  - Classifies new data it has not seen before

Types of Machine learning

Supervised learning

- Regression
- Classification

Unsupervised learning

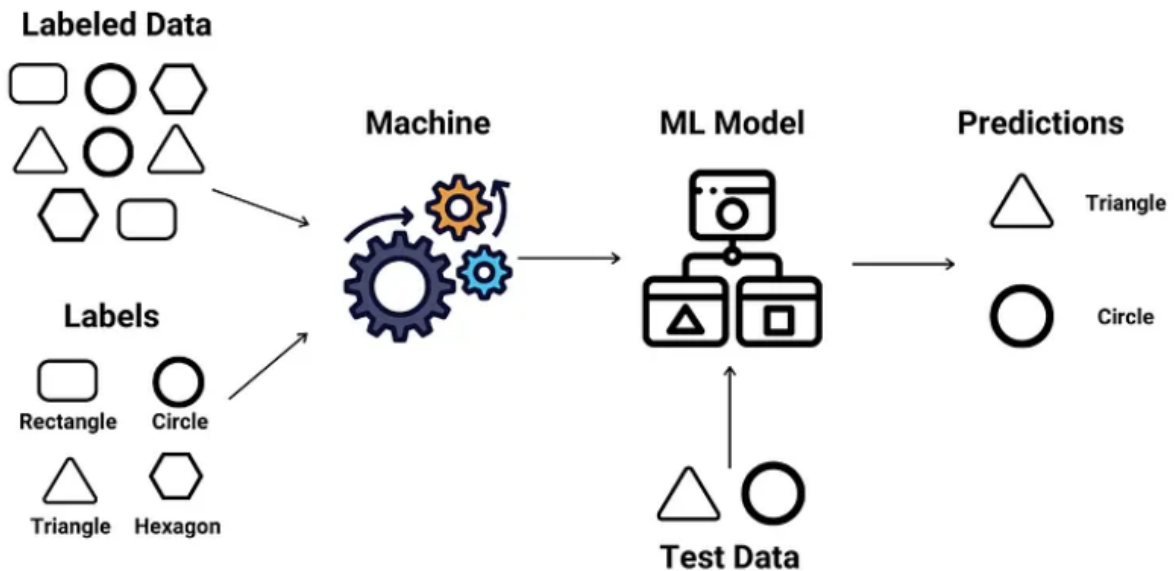
- Clustering
- Dimensionality Reduction

Semi-Supervised learning

Reinforcement Learning

Supervised learning

Supervised Learning is a ML task when the program is provided with training data that is labeled. Given the inputs, the system knows what the expected output label is.



Supervised learning

Regression

A Regression problem is a Supervised Learning problem wherein the variable we wish to predict (the dependent variable) is of a continuous nature.

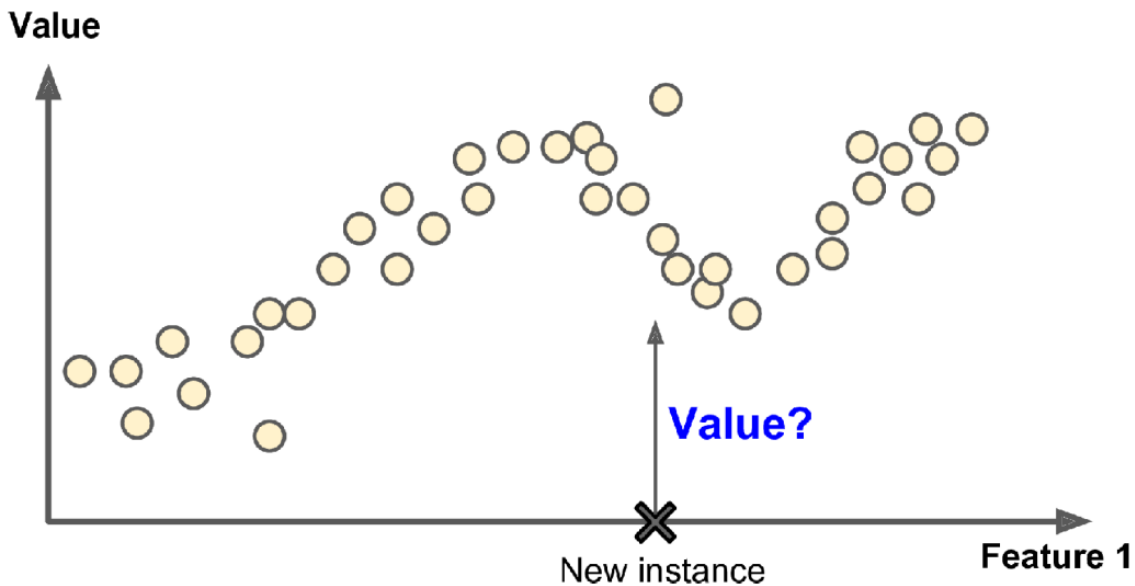
Classification

A Regression problem is a Supervised Learning problem wherein the variable we wish to predict (the dependent variable) is of a categorical nature.

Supervised learning

Regression

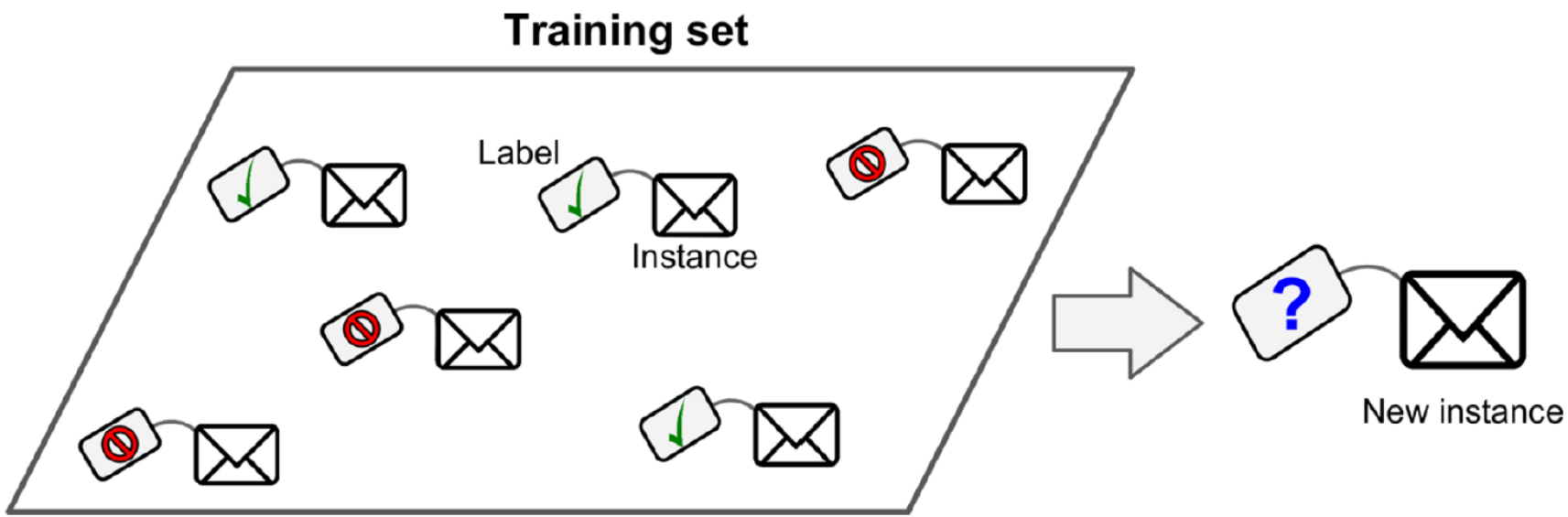
A Regression problem is a Supervised Learning problem wherein the variable we wish to predict (the dependent variable) is of a continuous nature.



Supervised learning

Classification

A Regression problem is a Supervised Learning problem wherein the variable we wish to predict (the dependent variable) is of a categorical nature.



Supervised learning: Algorithms

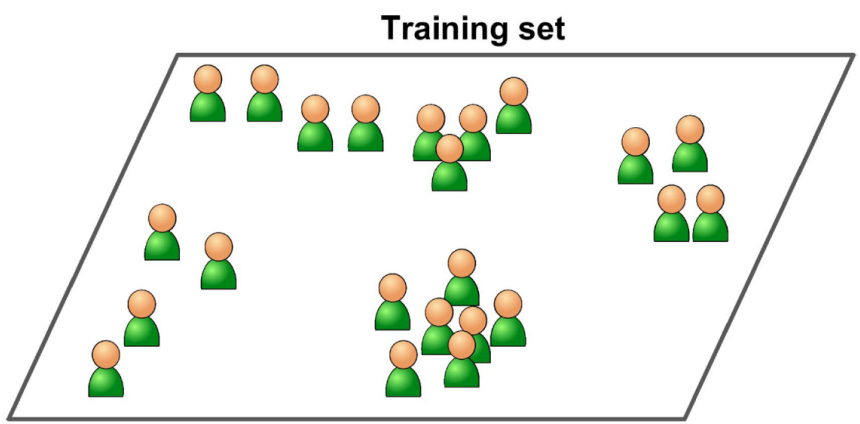
Some of the most important supervised learning algorithms

- k-NearestNeighbors
- LinearRegression
- LogisticRegression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks

Unsupervised learning

Unsupervised Learning

Unsupervised Learning is a ML task when the program is provided with training data that is unlabeled. Given the inputs, we do not know what the output should be! We can only rely on the inherent structure of the dataset



Unsupervised learning

Clustering

A Clustering problem is a learning problem in which we try to detect clusters of observations in our dataset.

Dimensionality Reduction

A Dimensionality Reduction problem is a ML problem in which we try to exploit the inherent structure of the dataset in order te reduce the amount of dimensions in our dataset, without losing any information that is stored within them.

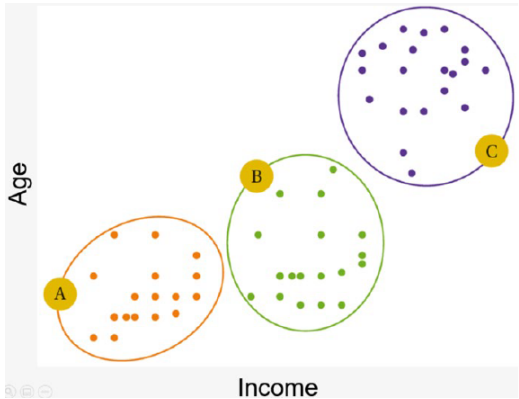
Unsupervised learning: Algorithms

Clustering

K-Means

DBSCAN

Hierarchical Cluster Analysis (HCA)



Visualization and dimensionality reduction

PrincipalComponent Analysis (PCA)

Anomaly detection and novelty detection

One-class SVM

IsolationForest

Semi-supervised Learning

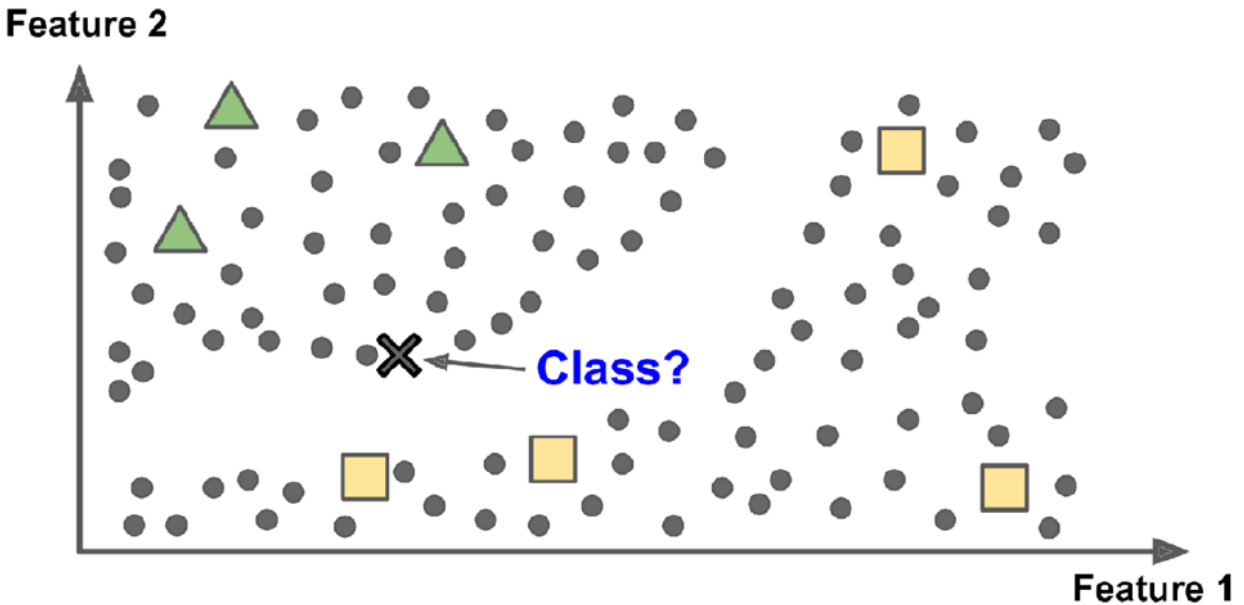
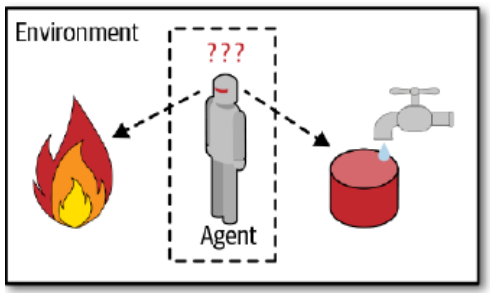


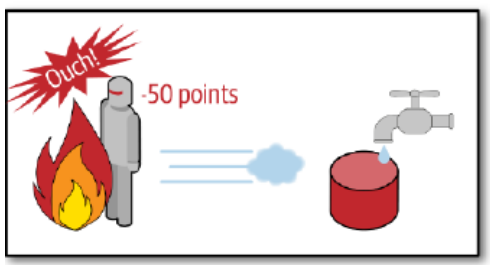
Figure 1-11. Semisupervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares

First a supervised learning algorithm is trained based on the labeled data only. This classifier is then applied to the unlabeled data to generate more labeled examples as input for the supervised learning algorithm. Generally only the labels the classifier is most confident in are added at each

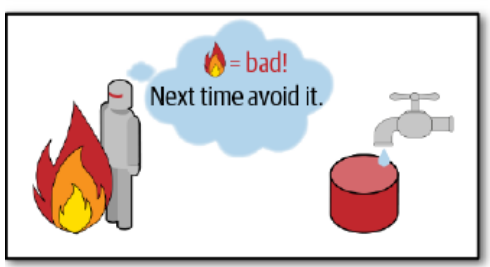
Reinforcement Learning



1. Observe
2. Select action using policy



3. Action!
4. Get reward or penalty



5. Update policy (learning step)
6. Iterate until an optimal policy is found

A short and very incomplete overview of Machine Learning

Many different types of ML aims:

- Clustering (unsupervised)
- Computer vision (detecting cancer in images, self driving cars)
- Natural Language Processing (Lernout & Hauspi / Alexa /
- Siri)
- Classification (binary / multi-class)
- Recommender systems

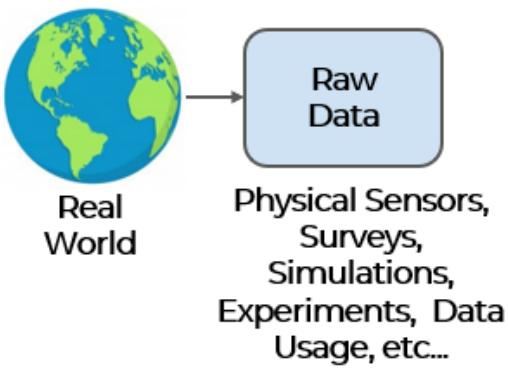
The Machine Learning Pathway

The Machine Learning Pathway

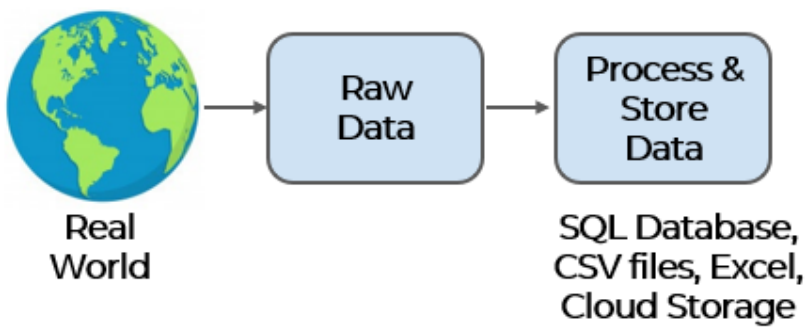
Let's discuss the general idea of a pathway of using Machine Learning and Data Science for a useful Real World Application.

Keep in mind: This overview is very broad and in reality there is a lot of overlap between the various stages presented here. There is also a lot of overlap in the roles that are presented and different organizations label role titles differently.

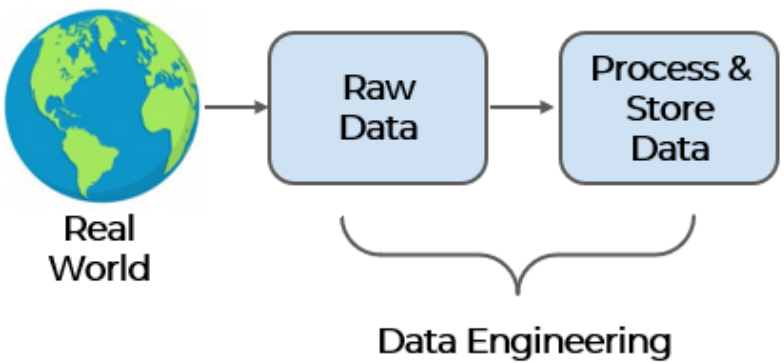
The Machine Learning Pathway



The Machine Learning Pathway

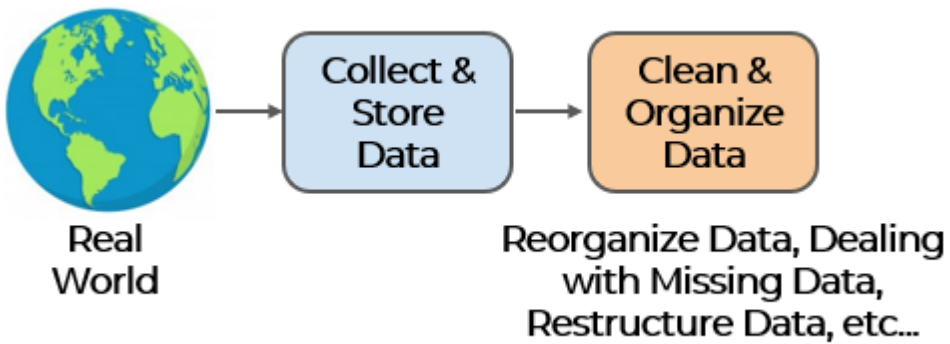


The Machine Learning Pathway

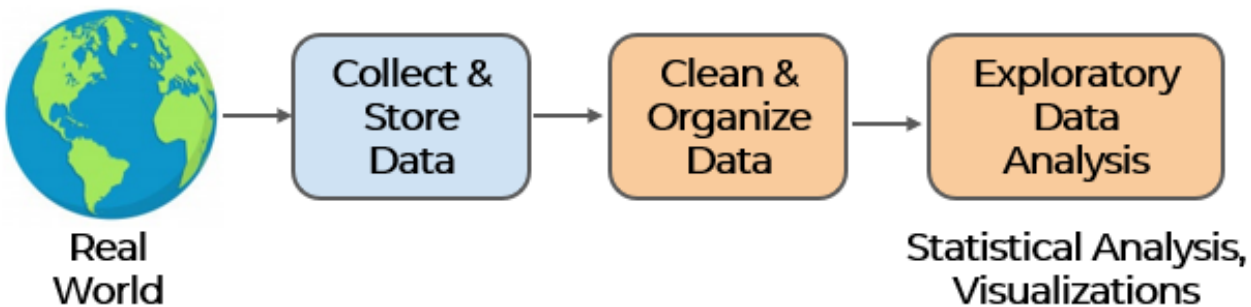


The Machine Learning Pathway

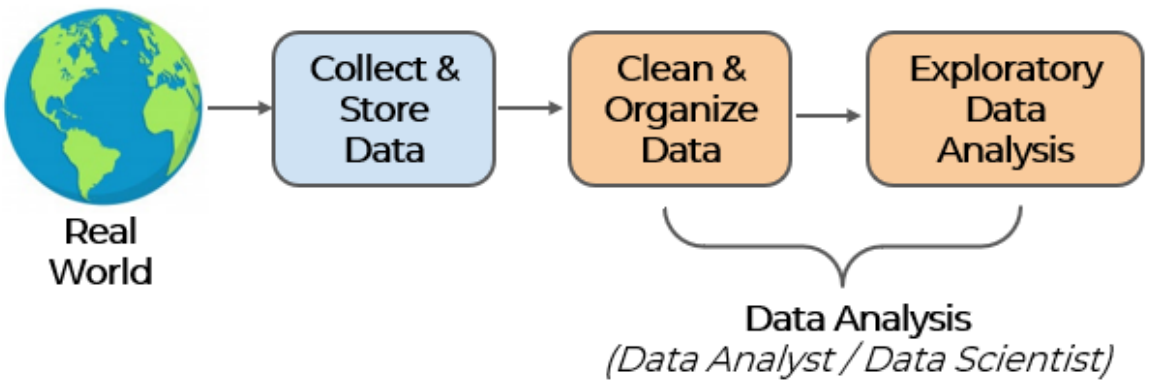
The Machine Learning Pathway



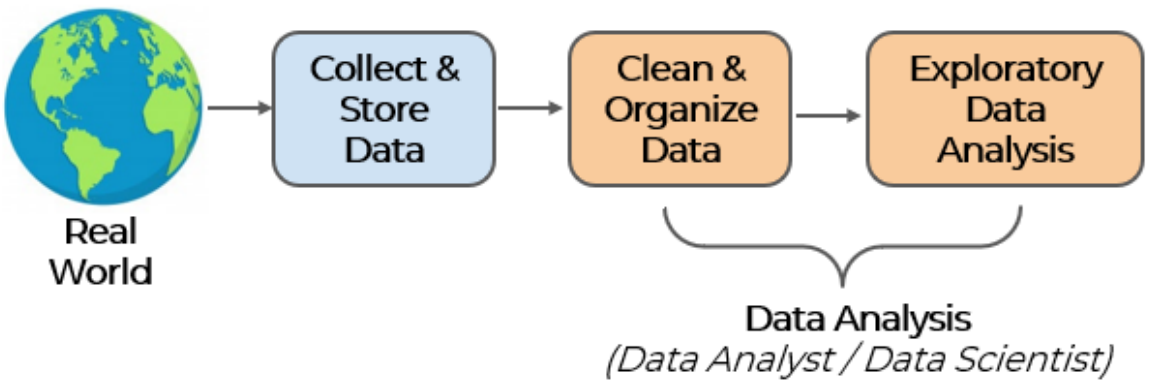
The Machine Learning Pathway



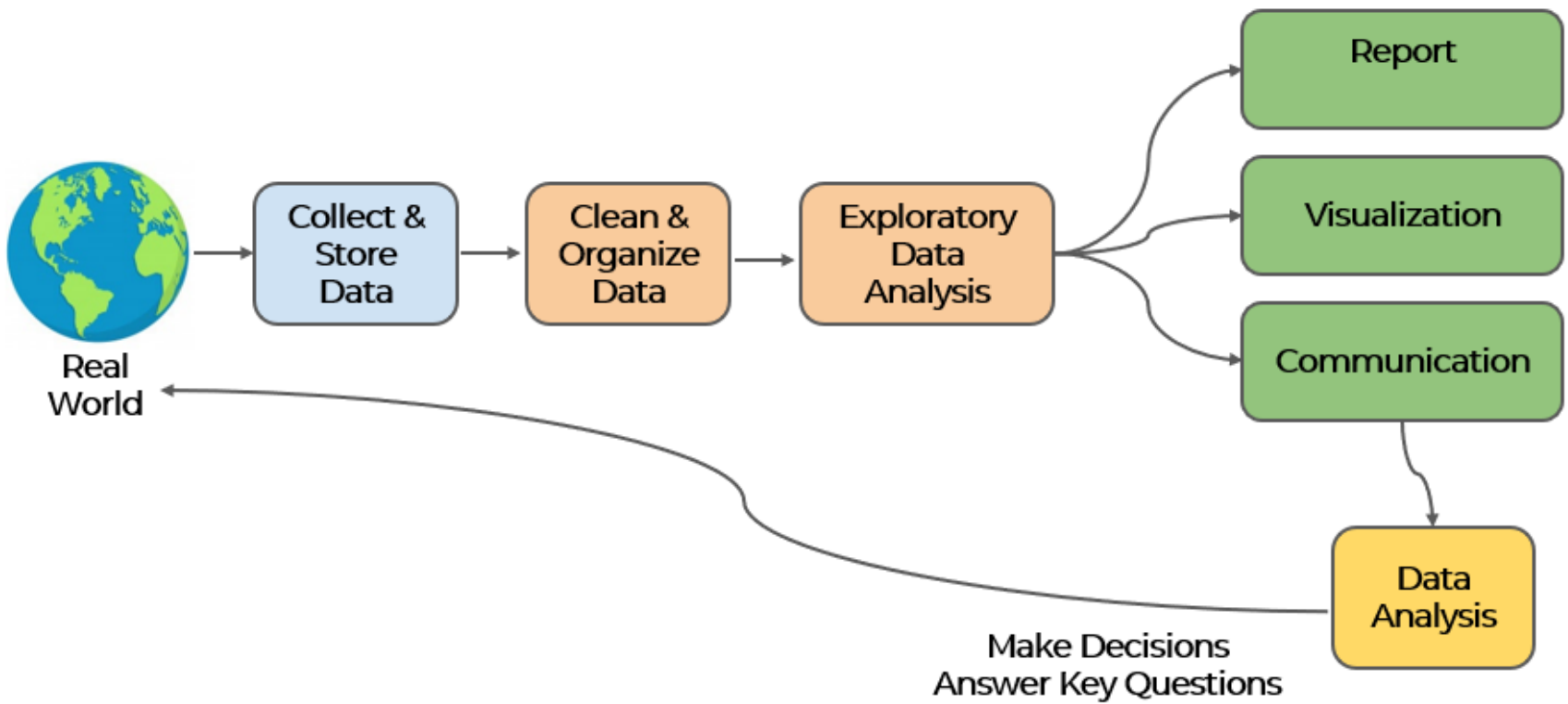
The Machine Learning Pathway



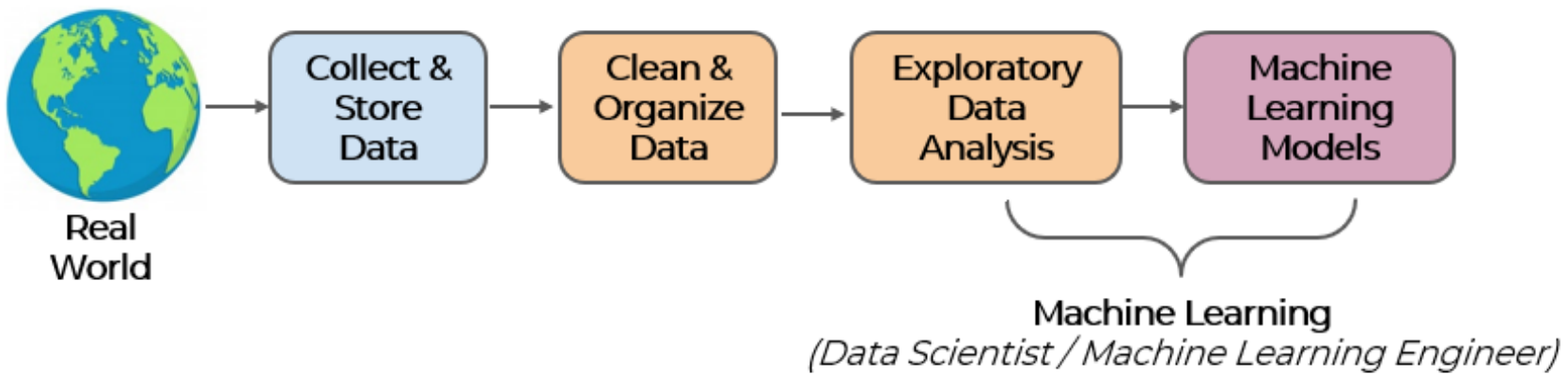
The Machine Learning Pathway



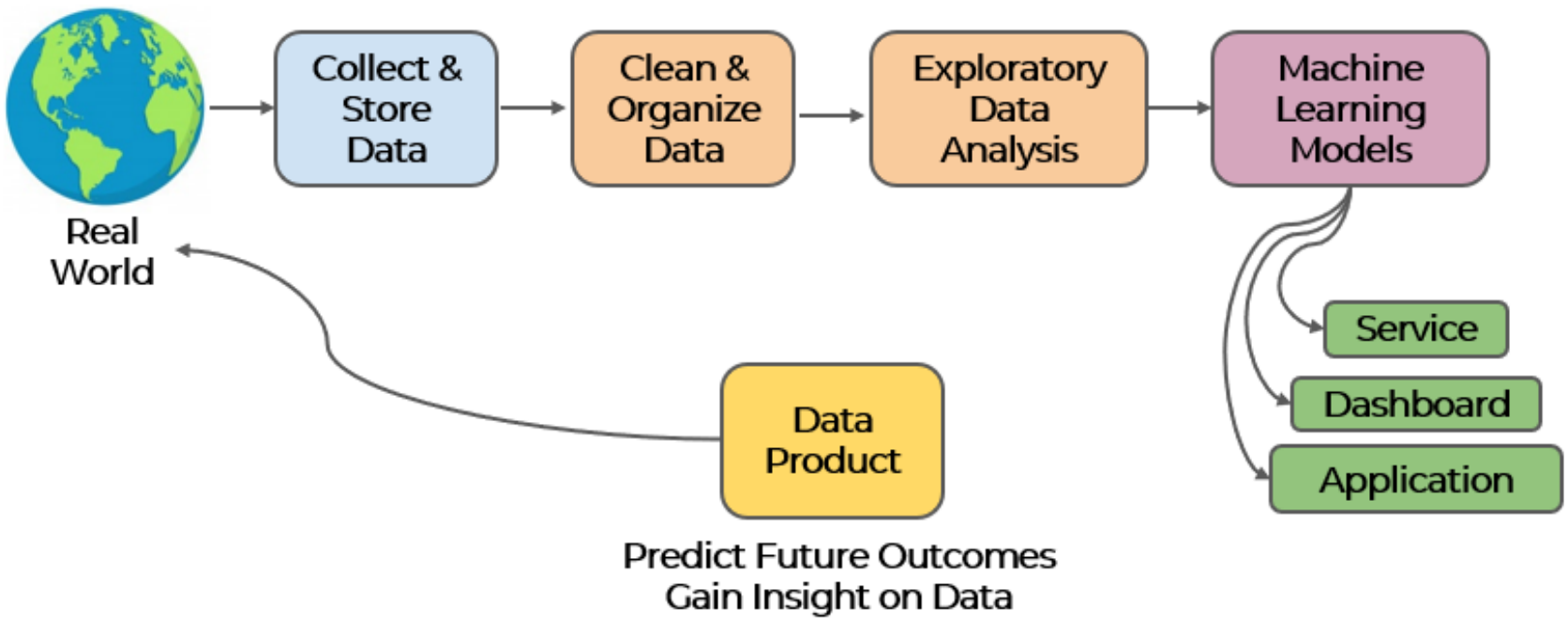
The Machine Learning Pathway



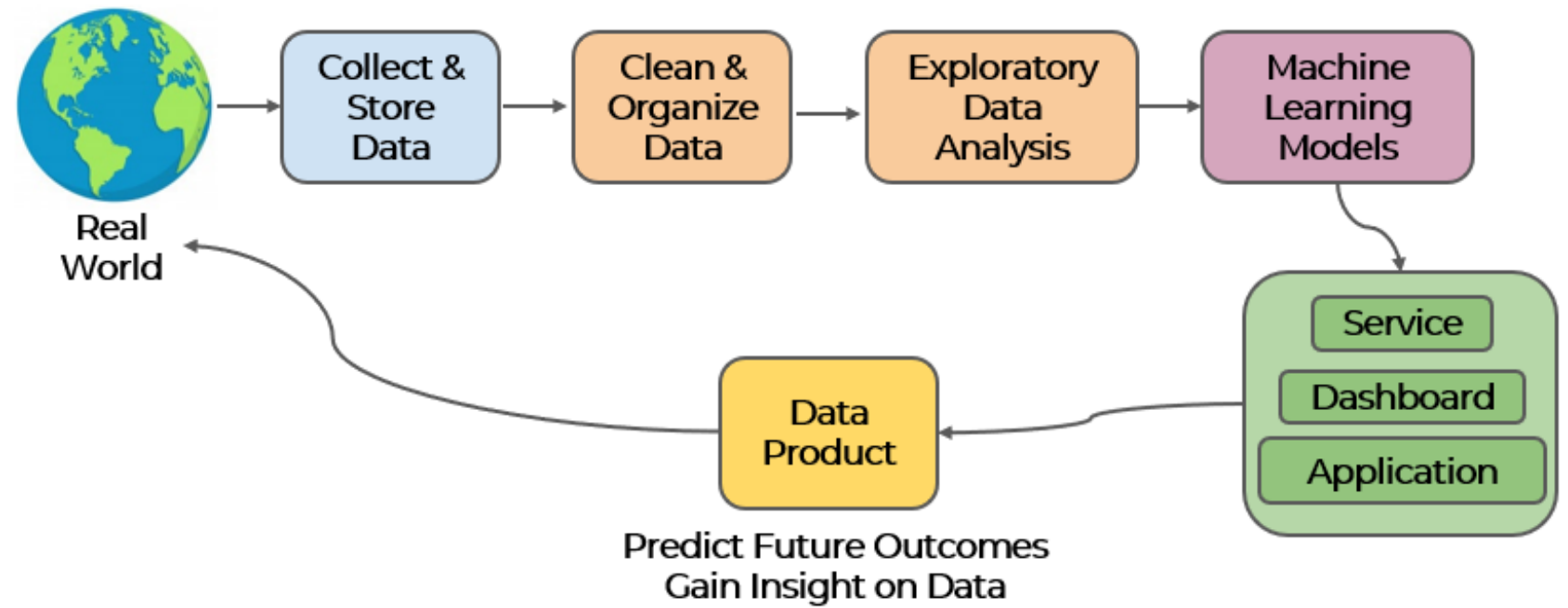
The Machine Learning Pathway



The Machine Learning Pathway



The Machine Learning Pathway



The Machine Learning Pathway

Now that we understand the general dynamics of the Data Science and Machine Learning Pathway, let's see if you are still able to perform the first steps with a quick recap project.

Recap Data cleaning & visualisation project

Recap Project

Let's travel back in time to the year 2015…

Recap Project

Fandango sells movie tickets online and also displays movie ratings

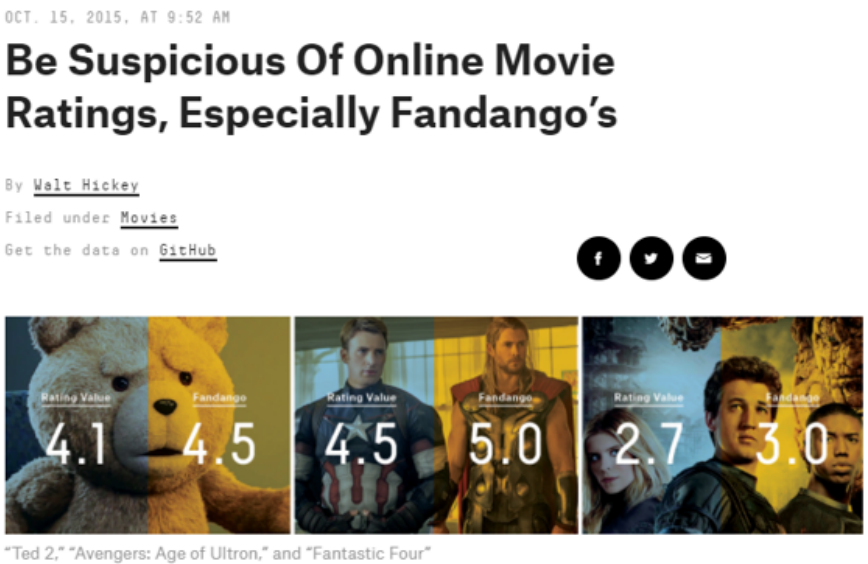
Recap Project

Taken 3 has a 4.5 star rating on Fandango.

[1080p] Taken 3 - Fence jump scene - 10 hours

But... you may be disappointed by the film…

Recap Project



"Ted 2," "Avengers: Age of Ultron," and "Fantastic Four"

Question to Answer:

- Is there a conflict of interest for a website that both sells movie tickets **and** displays review ratings?

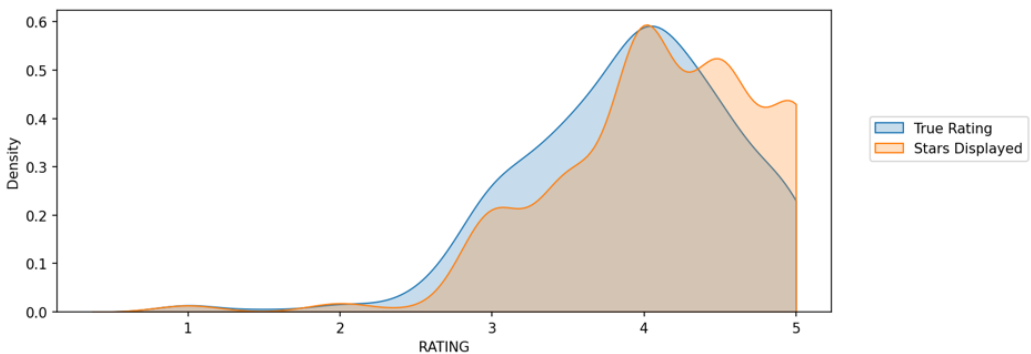
More Specifically:

- Does a website like Fandango artificially display higher review ratings to sell more movie tickets?

Recap Project

Fandango has two ratings: "**STARS**" & "**RATING**"

First we will compare these ratings to check for discrepancies.



Then we'll compare Fandango's ratings to other rating website scores: