Python for Data Science and Machine Learning

School Year 2023-2024

IST

Course Structure

1	Introduction to Python	OCT 19	OCT 26	NOV 9	NOV 16	NOV 23	NOV 30	DEC 7
2	Python Challenge	DEC 14						
3	Introduction to Data Science	DEC 21	JAN 11	JAN 18	JAN 25			
4	Introduction to Machine Learning	FEB 1	FEB 22					
5	Data Science & ML Challenge	MAR 7						

🟥 = Core Topics ⬚ = Optional Topics

Jupyter Notebook Setup

In a browser:

```
192.168.10.4:8888
```

Password: **ist**

Recap: Comparisons

- 5 is larger than 3

```
5 > 3
```

- -5 is larger than 9

```
-5 > 9
```

- 2 is the same as 2

```
2 == 2
```

- **not** (negation)

```
not True
```

```
not (5 < 3)
```

- **and** (both must be true)

```
(5 < 6) and (5 < 10)
```

- **or** (either must be true)

```
(5 < 3) or (5 < 10)
```

Recap: If-Statements

You can chain multiple conditions with **`elif`**.

What is the difference between these two snippets of code?

```
x = int(input())

if x < 3:
   print("X is less than 3")
elif x < 10:
   print("X is less than 10")
elif x < 25:
   print("X is less than 25")
```

```
x = int(input())

if x < 3:
   print("X is less than 3")
if x < 10:
   print("X is less than 10")
if x < 25:
   print("X is less than 25")
```

Recap: While-Loops

Allows you to repeat instructions

With an **if-statement**:

```
x = int(input("Insert num < 5: "))

if x >= 5:
    print("ERROR! Wrong number")
    x = int(input("Insert num < 5: "))

print("CORRECT!")
```

With a **while-loop**:

```
x = int(input("Insert num < 5: "))

while x >= 5:
    print("ERROR! Wrong number")
    x = int(input("Insert num < 5: "))

print("CORRECT!")
```

Recap: For-Loops

Repeat a specific amount of times

With a **while-loop**:

```
x = 0

while x < 10:
    print(x)
    x += 1
```

With a **for-loop**:

```
for x in range(10):
    print(x)
```

```
for x in range(2, 10):
    print(x)
```

```
for x in range(2, 10, 3):
    print(x)
```

Recap: Lists

Modifiable containers for data.

With **variables**:

```
num1 = 42
num2 = 100
num3 = 10

print(num1)
print(num2)
print(num3)
```

With a **list**:

```
nums = [42, 100, 8]

print(nums)
```

Recap: Accessing List Elements

To access list elements you can use the **[index]** operator.

NOTE: List indices start from **0**

index:		**0**	**1**	**2**	**3**	**4**
	nums =	[17,	28,	33,	56,	6]
index:		**-5**	**-4**	**-3**	**-2**	**-1**

```
print(nums[0])
```

```
print(nums[3])
```

```
print(nums[-2])
```

Recap: Modifying Lists

Adding new elements:

1. To insert at the back: **append**
2. To insert in any position: **insert**

Removing elements:

1. To an element: **pop**

   You may optionally pass an index, default is **-1**.

```
nums = [42, 100]

nums.append(8)
nums.insert(0, 200)
elem = nums.pop(1)

print(nums)
```

Recap: Additional List Functions

Additional functions that operate on lists

- Get the length of the list: **len**

```
len([4, 8, 10, 12])
```

```
len([-3])
```

```
len([])
```

- Get the max/min elements in a list: **max** and **min**

```
max([4, 8, -2, 0])
```

```
min([4, 8, -2, 0])
```

- Get the sum of all elements in a list: **sum**

```
sum([4, 8, -2, 0])
```

```
sum([-3])
```

Recap: Iterating Lists

Python provides multiple ways to **iterate over lists**.

The most used methodologies are:

Index-iteration:

```
nums = [10, 20, 30, 40]
for i in range(len(nums)):
    print(nums[i])
```

For-each loop:

```
nums = [10, 20, 30, 40]
for num in nums:
    print(num)
```

The output of the two snippets is identical

Recap: Dictionaries

Group data together using keys

With **variables**:

```
num1 = 42
num2 = 100
num3 = 10

print(num1)
print(num2)
print(num3)
```

With a **dict**:

```
nums = {"num1": 42, "num2": 100, "num3": 8}

print(nums)
```

Recap: Accessing Dictionary Elements

To access dictionary elements you can use the **[index]** operator.

NOTE: You can only access keys that exist

```
heights = {"Charles": 175, "Adam": 160, "Florence": 180}
```

```
print(heights["Adam"])
```

```
print(heights["Florence"])
```

ERROR:

```
print(heights["Dan"])
```

Recap: Modifying Dictionaries

```python
data = {"a": 42, "b": 3}
```

1. To insert a new key:

```python
data["c"] = 800
```

```python
data["d"] = 4.5
```

2. To modify an existing elements you can assign to the key

```python
data["a"] = 10
```

```python
data["b"] = 3.2
```

3. You can remove elements in a dict with the **del** function.

```python
del data["a"]
```

```python
del data["c"]
```

Recap: Iterating Dictionaries

Python provides multiple ways to **iterate over dicts**.

The most used methodologies are:

Key-iteration:

```
data = {"a": 4, "f": 1, "z": 8}

for key in data:
   value = data[key]
   print(key, value)
```

For-each loop:

```
data = {"a": 4, "f": 1, "z": 8}

for key, value in data.items():
  print(key, value)
```

The output of the two snippets is identical

Recap: Sets

Unordered collections of unique elements

With **variables**:

```
num1 = 42
num2 = 100
num3 = 42

print(num1)
print(num2)

if (num3 != num1) and (num3 != num2):
   print(num3)
```

With a **list**:

```
nums = {42, 100, 42}

print(nums)
```

Recap: Anatomy of a Set

Anatomy of a set:

1. Uses curly brackets **{ }**
2. Elements separated by comma **,**
3. Can take any values (will remove duplicates)

```
nums = {42, 100, 42}
```

```
data = {"A", "C", "D"}
```

Recap: Modifying Sets

Adding new elements:

1. To insert an element: **add**
2. To remove an element: **remove**

```
nums = {42, 100}

nums.add(8)
nums.remove(100)
nums.add(50)

print(nums)
```

Recap: Set Theory

Set theory operations:

```
set1 = {"A", "B", "C"}
set2 = {"B", "C", "D"}
```

1. Union: **set1 | set2** `{"A", "B", "C", "D"}`

2. Intersection: **set1 & set2** `{"B", "C"}`

3. Difference: **set1 - set2** `{"A"}`

Recap: Iterating Sets

Python provides one way to **iterate over sets**.

This makes set and list iteration very similar:

For-each loop:

```python
nums = {40, 10, 30, 20}
for num in nums:
    print(num)
```

Remember sets are unordered (so no ordering guarantees!)

Recap: Data-Structure Membership

You can use the **in** keyword to check if an element is in a given data structure. This applies to **lists**, **sets** and **dictionaries**.

```
data1 = ["a", "b", "c"]
x = "b"

print(x in data1)
```

```
data2 = {"a", "b", "c"}
y = "b"

print(y in data2)
```

```
data3 = {"a": 10, "b": 20}
z = "b"

print(z in data3)
```

Recap: Functions

Repeatable snippets of code

With **variables**:

```
num1 = 42
num2 = 10

x = num1 + 100
y = num2 + 100
```

With a **function**:

```
def add_100(a):
    return a + 100


num1 = 42
num2 = 10

x = add_100(num1)
y = add_100(num2)
```

Recap: Anatomy of a Function

Anatomy of a function:

1. Begins with the **def** keyword
2. Arguments are in brackets **( )** separated by comma **,**
3. Uses the **return** keyword to give output

```
def add_100(a):
    return a + 100

add_100(42)
```

```
def multiply(a, b):
    return a * b

multiply(4, 5)
```

Recap: Calling a Function

To call a function you must use the **function name** followed by all the **parameters** within **brackets**.

```python
def is_even(n):
   return n % 2 == 0
```

```python
def create_list(a, b, c):
  return [a, b, c]
```

```python
x = is_even(2)
y = is_even(5)

print(x)
print(y)
```

```python
list1 = create_list(1, 2, 3)
list2 = create_list(4, 5, 6)
```

Recap: Calling a Function

Functions are not required to take arguments.

```
def create_list():
   my_list = []
   for i in range(1, 4):
       my_list.append(i)
   return my_list
```

```
data1 = create_list()
data1.append(50)

data2 = create_list()

print(data1)
print(data2)
```

Part 1: Competition Time!

In a browser:

```
192.168.10.4:8421
```

Username: **<team-color>**

Recap: Pandas

Pandas is a powerful Python data analysis toolkit.

It provides flexible data structures like **Series** and **DataFrame**.

Widely used in data science, finance, and many other fields.

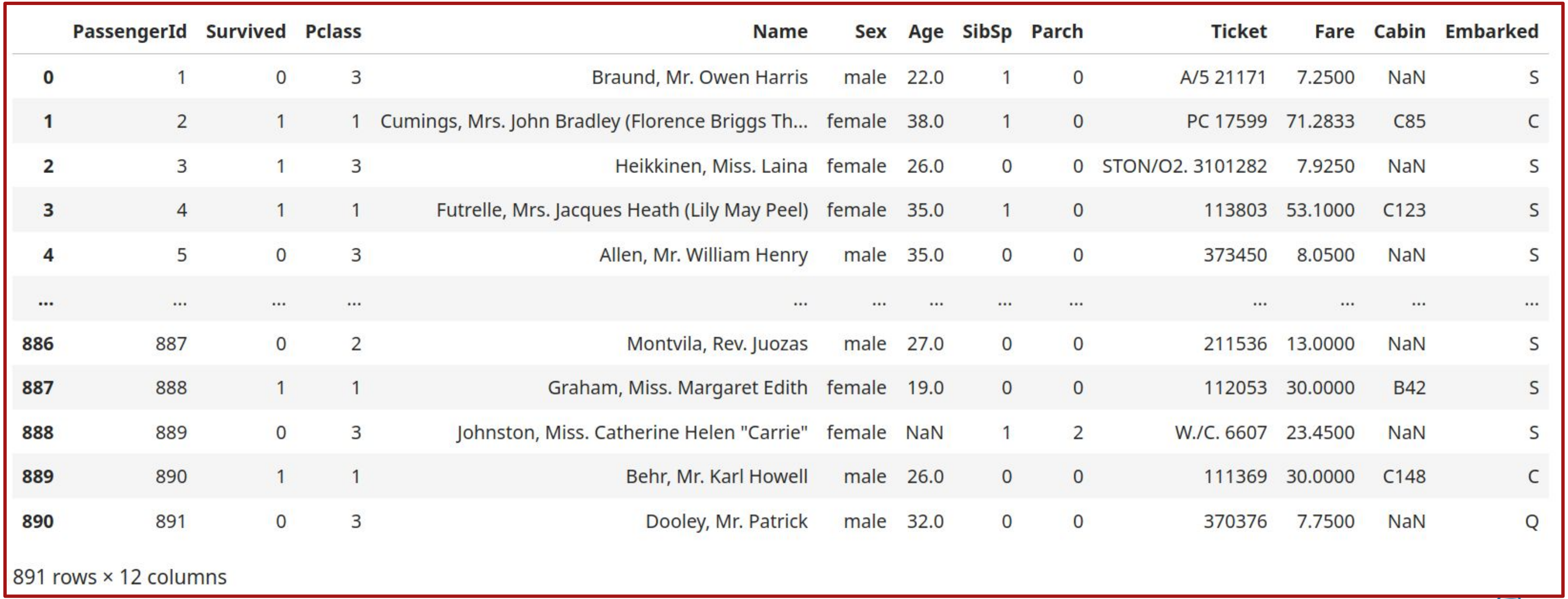
```python
import pandas as pd
import numpy as np
```

Recap: DataFrame

A **DataFrame** is a two-dimensional data structure with labeled axes (rows and columns).

```
df = pd.read_csv("titanic_dataset.csv")
df
```

Recap: DataFrame



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Recap: Selecting DataFrame Data

- The **loc** method in Pandas can be used for selecting rows but also for columns.
- By specifying the row and column labels, you can access specific portions of the dataset.

```
df.loc[0, "Name"]
```

```
df.loc[0:4, "Name"]
```

```
df.loc[:4, "Name"]
```

```
df.loc[4, ["Name", "Age"]]
```

```
df.loc[0:4, ["Name", "Age"]]
```

```
df.loc[:, ["Name", "Age"]]
```

Recap: Boolean Indexing

- **Boolean indexing** in Pandas allows you to select data subsets based on the actual values in the data.

```
df[df.loc[0:9, "Age"] > 30]
```

- **SHORTHAND:** If you wish to **select specific columns** across **all rows** you can use the following:

```
df.loc[:, 'Age']
```

→

```
df['Age']
```

```
df[df.loc[:, "Age"] > 30]
```

→

```
df[df["Age"] > 30]
```

Recap: Chaining Indexing

You can **chain** multiple boolean indexing operations by using:

- | for "or"
- **&** for "and"

IMPORTANT! You must use **brackets**!

```
df[(df["Pclass"] == 1) | (df["Pclass"] == 2)]
```

```
df[(df["Pclass"] == 1) & (df["Age"] < 18)]
```

Recap: Data Analysis

We can use the **.mean()**, **.count()**, **.max()** and **.min()** functions to analyse our data.

```
df["Age"].mean()
```

```
df["Fare"].max()
```

```
df[df["Survived"] == 1]["Age"].min()
```

Recap: Grouping

Before we analyse our data we can group pieces of information together. We use the **.groupby()** function. We pass in the **column** to group the data with.

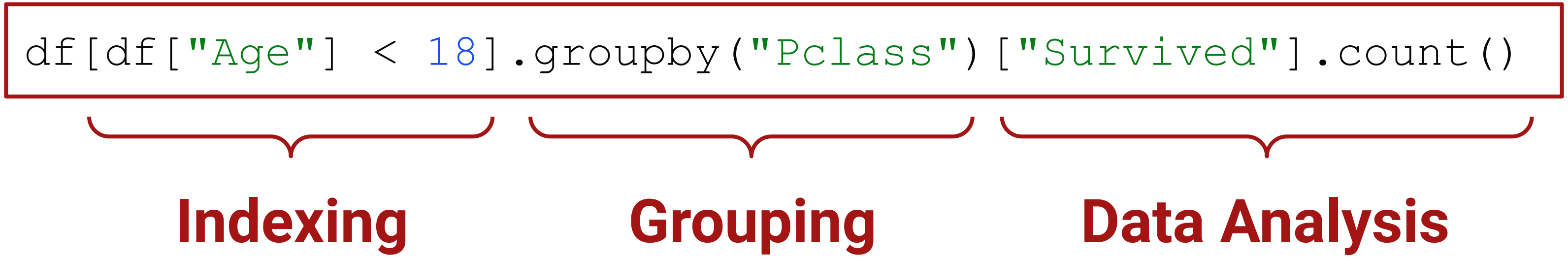
```
df.groupby("Embarked")["Name"].count()
```

```
df.groupby("Pclass")["Survived"].mean()
```

Recap: Indexing, Grouping & Analysis

When using them all together, in order we:

1. First use boolean indexing
2. Secondly use grouping
3. Finally we select the analysis function we’d like



```
df[df["Age"] < 18].groupby("Pclass")["Survived"].count()
```

Indexing **Grouping** **Data Analysis**

Recap: Feature Engineering

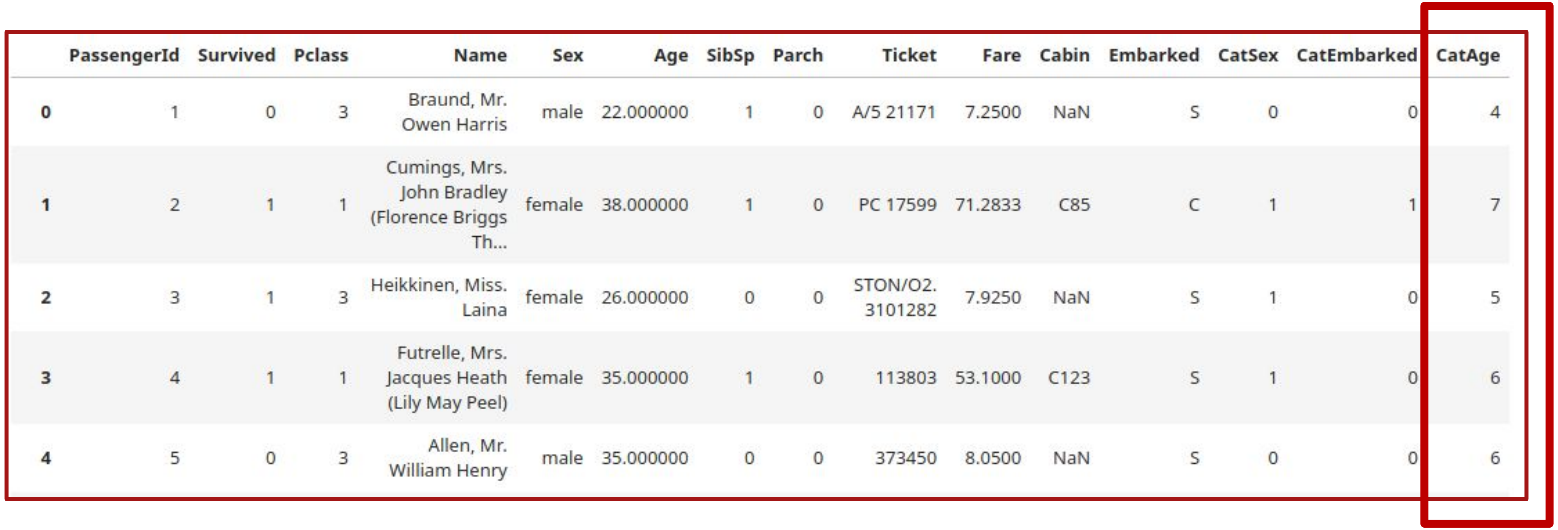
Feature engineering or feature extraction or feature discovery is the process of **extracting features** (characteristics, properties, attributes) **from raw** data **to support training** a downstream statistical model.

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome H. (2009).

The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer. ISBN 978-0-387-84884-6.

Recap: Categorization

Let’s apply our categorization to the **Age** column values, by creating a new column **CatAge**:

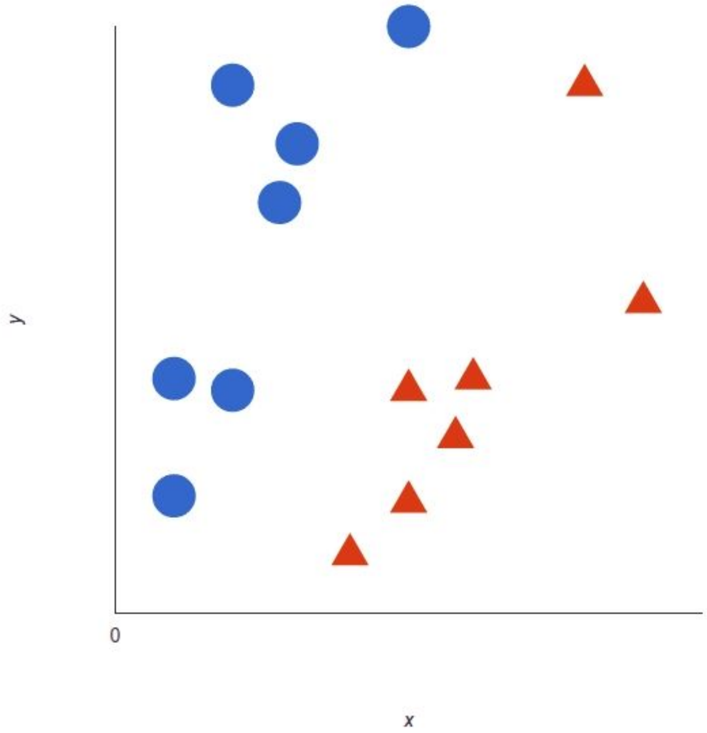


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	CatSex	CatEmbarked	CatAge
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	NaN	S	0	0	4
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C	1	1	7
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	NaN	S	1	0	5
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S	1	0	6
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	NaN	S	0	0	6

Recap: Classifiers

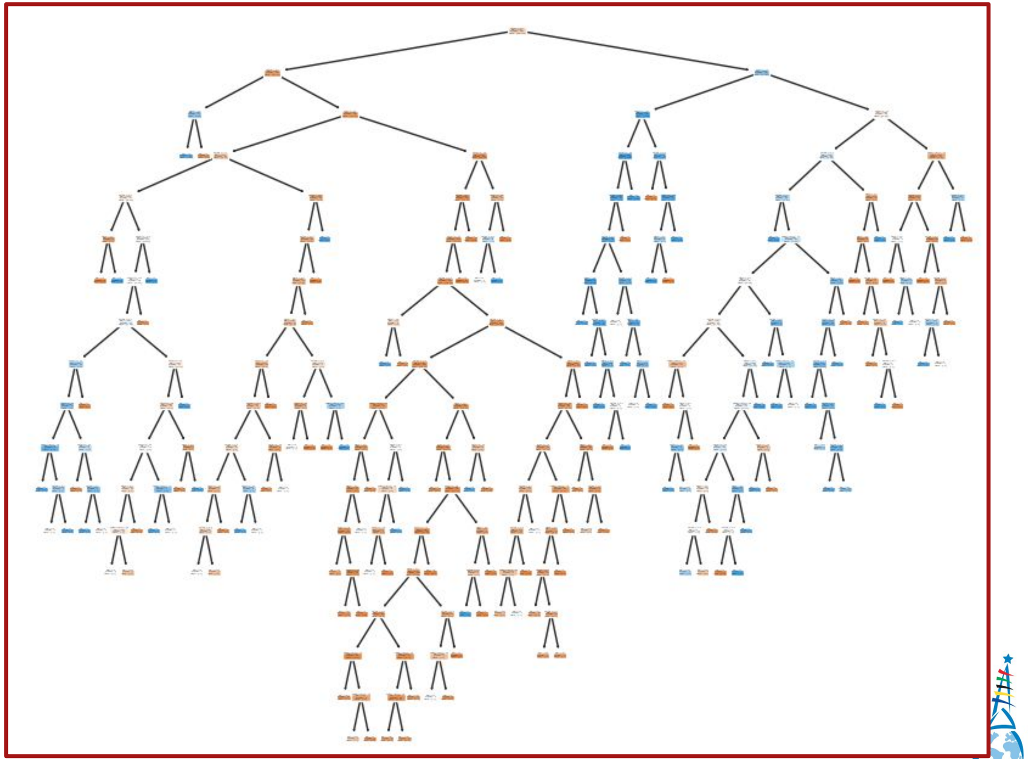
A classifier in machine learning is an algorithm that automatically orders or **categorizes data** into one or more of a set of "**classes**."

https://monkeylearn.com/blog/what-is-a-classifier/



Recap: Decision Tree Classifiers

It classifies data into **finer and finer categories**: from “tree trunk,” to “branches,” to “leaves.”

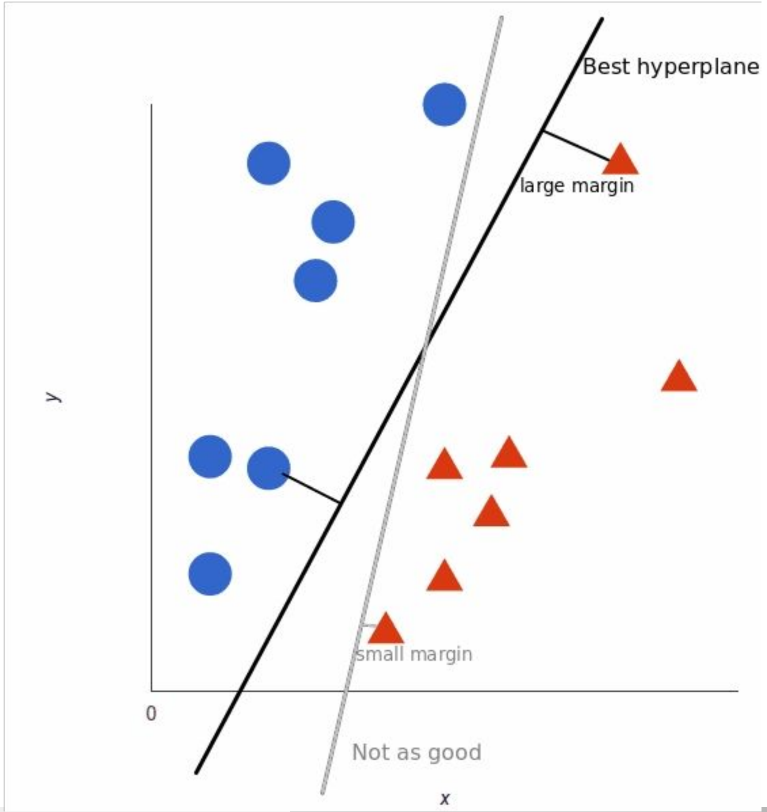


Recap: Random Forest

A **Random Forest** is like a **group decision-making** team in machine learning. It combines the opinions of many “trees” (individual models) to make **better predictions**, creating a more robust and accurate overall model.

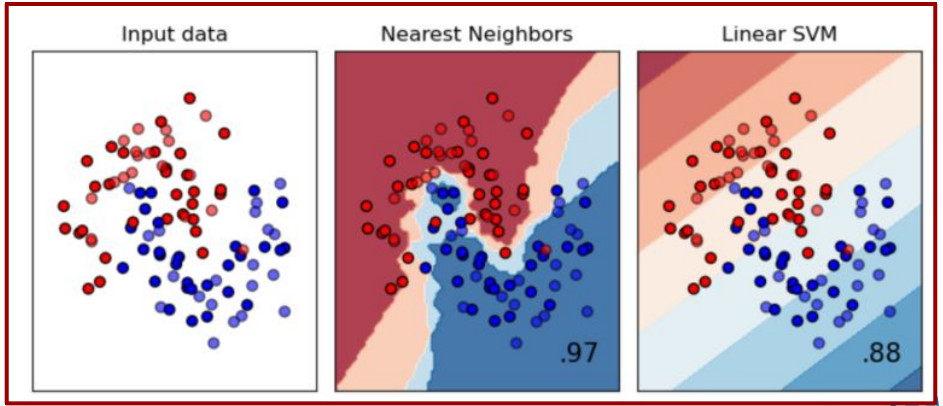
Recap: Support Vector Machines

SVM algorithms classify data and train models within super finite degrees of polarity, creating a **3-dimensional classification model** that goes beyond just X/Y predictive axes.



Recap: K-Nearest Neighbors

K-nearest neighbors (k-NN) is a pattern recognition algorithm that stores and learns from training data points by **calculating how they correspond to other data** in n-dimensional space. K-NN aims to find the **k closest related data points** in future, unseen data.



Recap: Boosted Trees

Random forests also have drawbacks. They can’t deal with mistakes (if any) created by their individual decision trees.

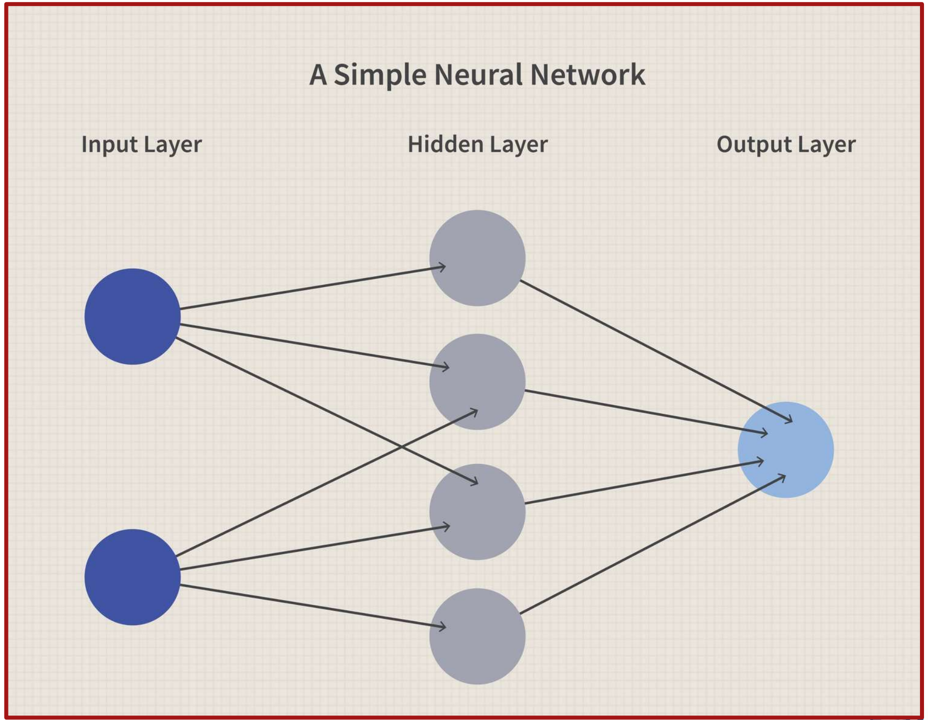
Boosting is a method of **combining many weak learners** (trees) into a strong classifier.

Recap: Deep Learning

Deep Learning is a type of machine learning based on **artificial neural networks** in which multiple layers of processing are used to **extract progressively higher level features** from data.

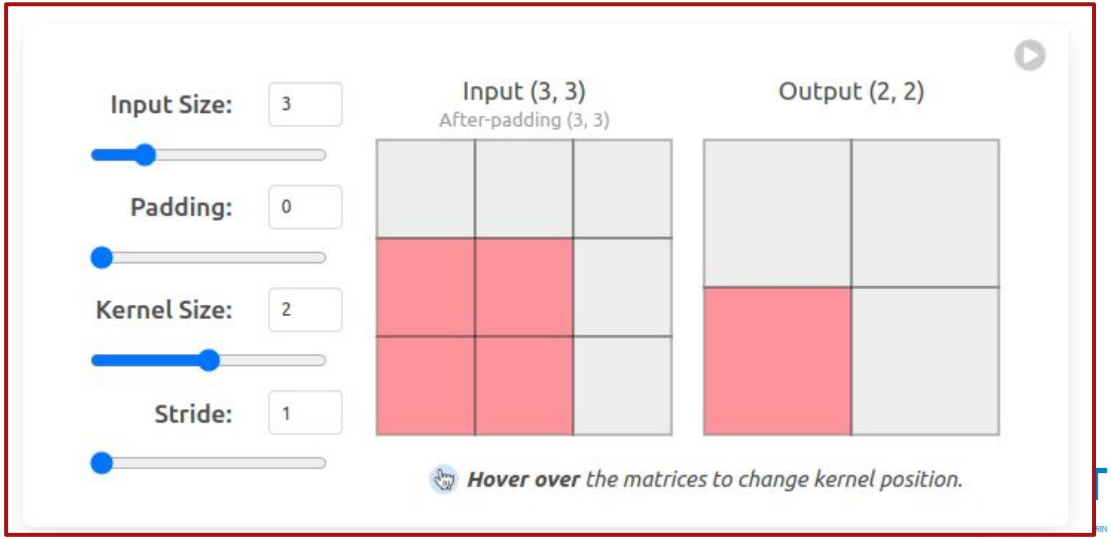
Recap: Dense Neural Networks

A **neural network** consists of **layers of nodes**, or artificial neurons—an **input layer**, one or more **hidden layers**, and an **output layer**. Each node connects to others, and has weights and a threshold.



Recap: Convolutional Neural Networks

A **Convolutional Neural Network**, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a **grid-like topology**, such as an image.



Input Size: 3

Padding: 0

Kernel Size: 2

Stride: 1

Hover over *the matrices to change kernel position.*

Part 2: Competition Time!

In a browser:

https://ahaslides.com/LGSPZ

Username: **<your-name>**

Team: **<team-color>**

End of Course

My contact details:

alberto.spina.1996@gmail.com