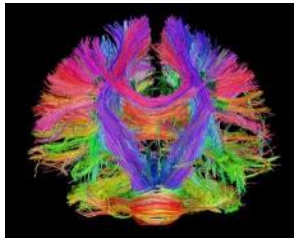



CS485
Data Science and Applications

Lecture 1: Introduction

Grigorios Tsagkatakis

Today's Objectives

- Course overview
- Introduction to topic
- Python primer
- Useful python libraries

About CS485

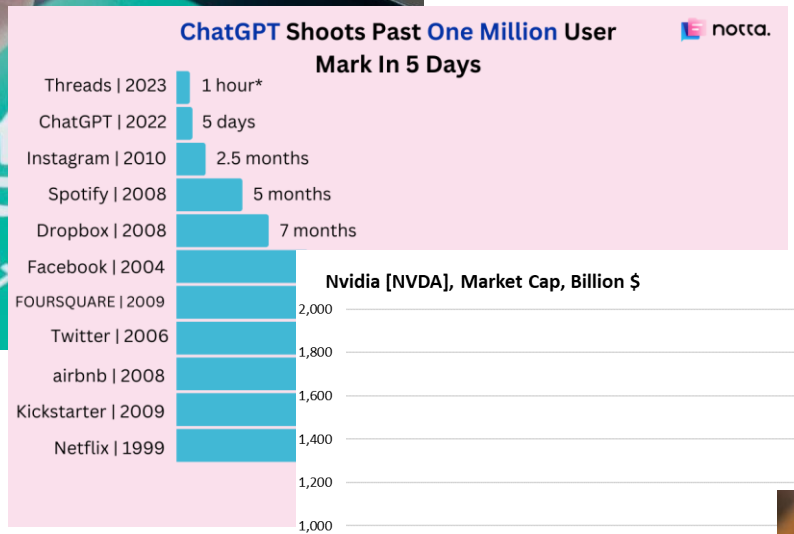
- Lectures A113
  - Monday 12:00-14:00, Theory
  - Wednesday 12:00-14:00, 1st hour theory, 2nd hour hands-on

- TA
  - Πολυχρονάκης Ελευθέριος, csdp1372@csd.uoc.gr
  - Κληρονόμου Ειρήνη, csdp1330@csd.uoc.gr
  - Γιάννης Φώτης, ifotis@csd.uoc.gr

- Prerequisites: HY-119 (Linear Algebra), HY-150 (Programming), HY-217 (Probabilities)
- Course material
  - https://elearn.uoc.gr/course/

Motivation


ChatGPT


ChatGPT Shoots Past One Million User Mark In 5 Days
notta.
Threads | 2023
1 hour*
ChatGPT | 2022
5 days
Instagram | 2010
2.5 months
Spotify | 2008
5 months
Dropbox | 2008
7 months
Facebook | 2004
FOURSQUARE | 2009
Twitter | 2006
airbnb | 2008
Kickstarter | 2009
Netflix | 1999


Nvidia [NVDA], Market Cap, Billion $
WTF
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
200
0
2018
2019
2020
2021
2022
Source: YCharts


deepseek

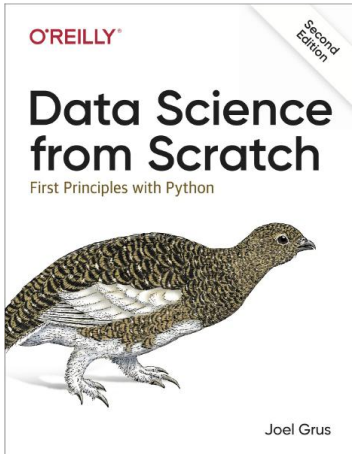
Topics

Week		Short description
1	10-Feb	Introduction to Data science. Python primer, Scripting, Visualization.
2	17-Feb	Matrix Calculus & Linear Algebra
3	24–Feb	Matrix Analysis, Regression
4	3-Mar	Probability & Statistics: Random Variables, Probability Distributions
5	10-Mar	Bayes rule & applications
6	17-Mar	Learning from Data: Machine Learning Principles
7	24-Mar	Data handling, Feature Engineering & Data Preprocessing
8	31-Mar	Deep Learning Foundations
9	7-Apr	Deep Learning Frameworks
10	28-Apr	Image & Video Analytics
11	5-May	Time Series Analysis
12	12-May	Natural Language Processing (NLP)
13	19-May	Large scale, Distributed & Edge analytics

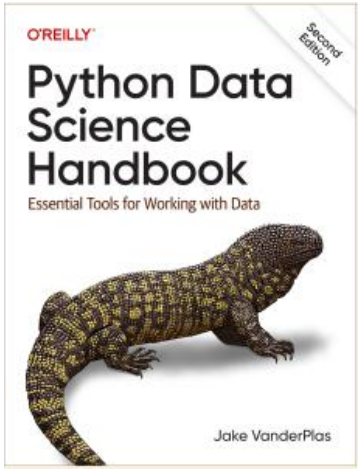
Books

- Grus Joel, Επιστήμη Δεδομένων: Βασικές Αρχές και Εφαρμογές με Python, 2η έκδοση, Εκδόσεις Παπασωτηρίου, 2021. (Μετάφραση του Data Science from Scratch: First Principles with Python 2nd Edition, O'Reilly Media)

  Κωδικός Βιβλίου στον Εύδοξο: 94690736



- Jake VanderPlas - Python Data Science Handbook- O'Reilly Media (2022)
- Βερύκιος, Β., Καγκλής, Β., & Σταυρόπουλος, Η. (2015). Η επιστήμη των δεδομένων μέσα από τη γλώσσα R [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, https://hdl.handle.net/11419/2965 (suggested)
- Jeff M. Phillips, Mathematical Foundations for Data Analysis, Springer Series in the Data Sciences, 1st ed. 2021 Edition https://mathfordata.github.io/ (suggested)



Grading (undergraduate)

- Weekly assignments: 40%
  - ➢Jupiter notebook with comments/text
- Quizzes/Midterm: 30%
  - ➢In class, online, every few weeks
- Final exam: 30%
  - ➢Multiple choice + open-ended

All above are compulsory for getting a grade at the end of the exam

Grading (graduates)

- Weekly assignments: 40%
  - ➢Jupiter notebook with comments/text
- Quizzes/Midterm: 20%
  - ➢In class, online, every few weeks
- Final exam: 20%
  - ➢Multiple choice + open-ended
- Term project: 20%
  - ➢Report & Presentation

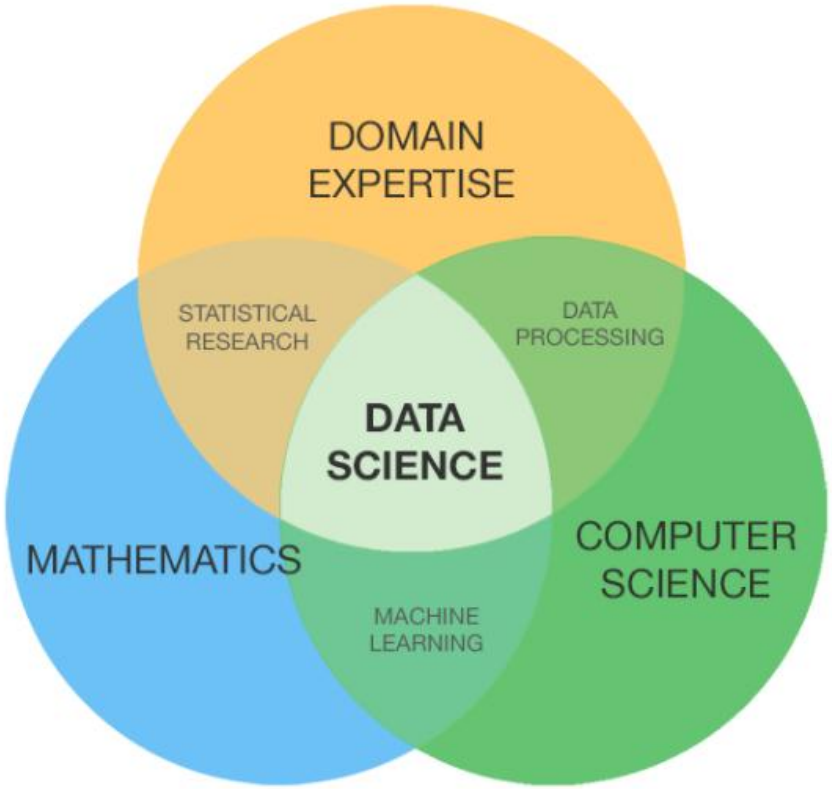
All above are compulsory for getting a grade at the end of the exam

Weekly assignments

Week	Short description
1	Perform data manipulation and basic analysis on a sample dataset using Pandas and create visualizations using Matplotlib and Seaborn.
2	Perform linear and non-linear regression on real data
3	Perform statistical analysis of real-world datasets, including hypothesis testing.
4	Build and validate machine learning models using scikit-learn.
5	Create and train a neural network using TensorFlow.
6	Develop a data cleaning and preprocessing pipeline (handling missing values and outliers).
7	Deploy a machine learning model on a cloud platform and create an API for predictions.
8	Analyze unstructured data formats with Pandas.
9	Apply signal processing techniques for noise reduction in audio.
10	Build a time series forecasting model using techniques like ARIMA.
11	Implement an image classification task using a pre-trained neural network.
12	Sentiment analysis using libraries like NLTK or spaCy.
13	Analyze network data using graph libraries (NetworkX) for identifying key nodes and communities.


HOW'S THE BIG DATA PROJECT COMING ALONG, HOSKINS?
© D.Fletcher for CloudTweaks.com

"Big Data"



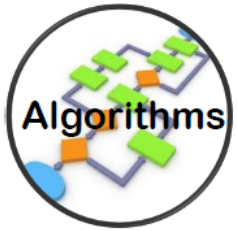
scientific, social, or business problem → data problem → collect & understand data → clean & format data → use data to create solution

What Is Data Analysis?

...using data to discover useful information...

- **data**: anything you can *measure* or *record*

- **statistics**: summarize (and visualize) *main characteristics* of the data

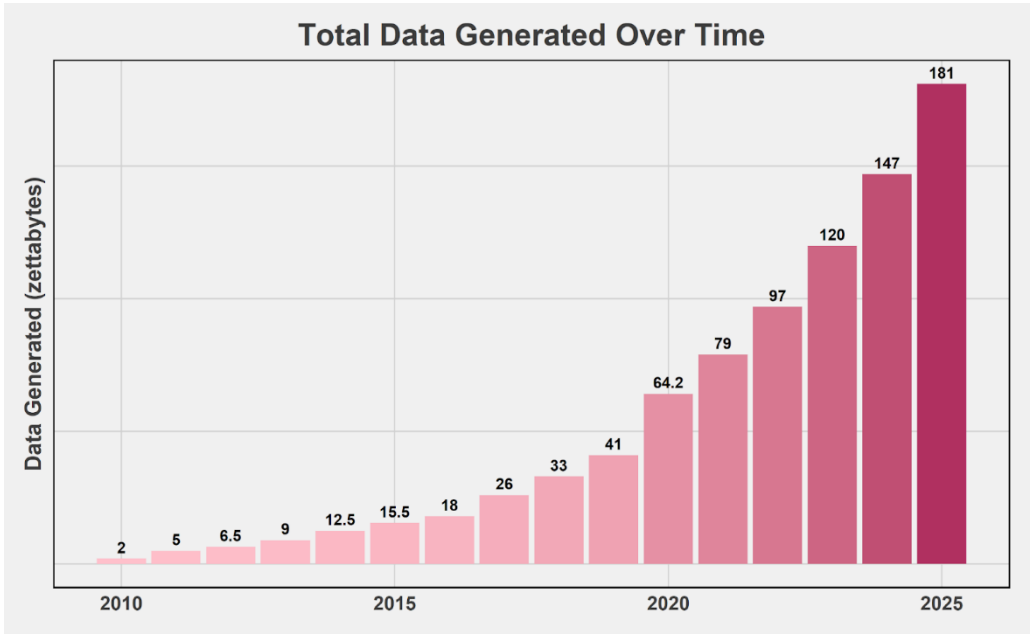


- **algorithms**: apply algorithms to find *patterns* in the data

Big Data

The 5Vs

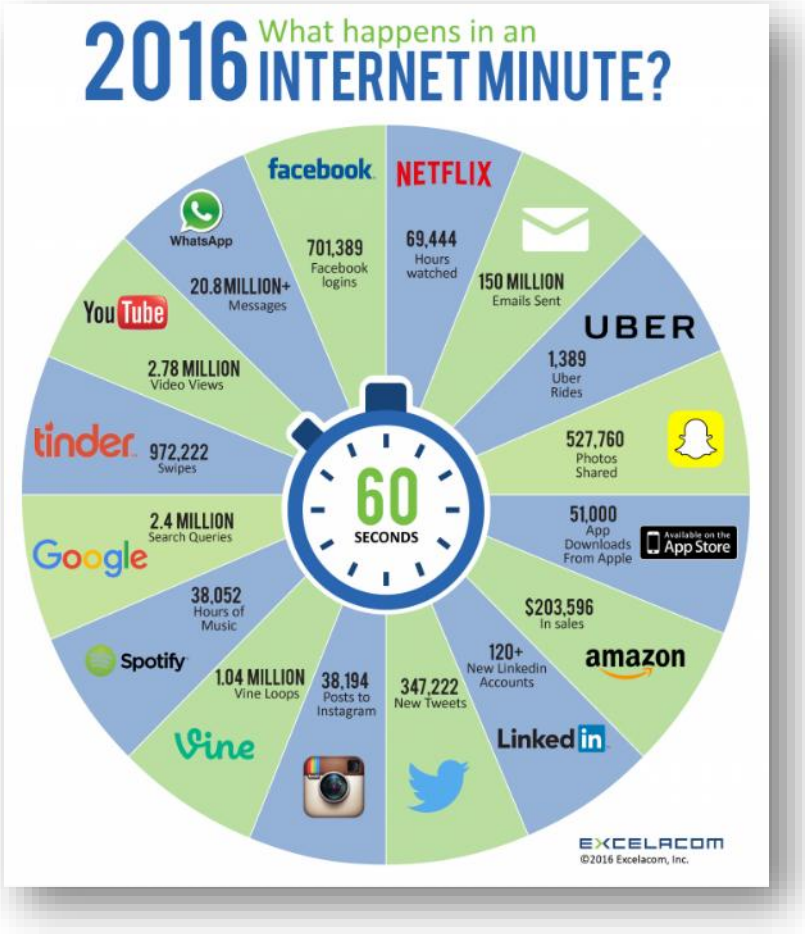
- **Volume**



Big Data

The 5Vs

- Volume
- **Velocity**



Big Data

The 5Vs

- Volume
- **Velocity**

Big Data

The 5Vs

- Volume
- **Velocity**

Big Data

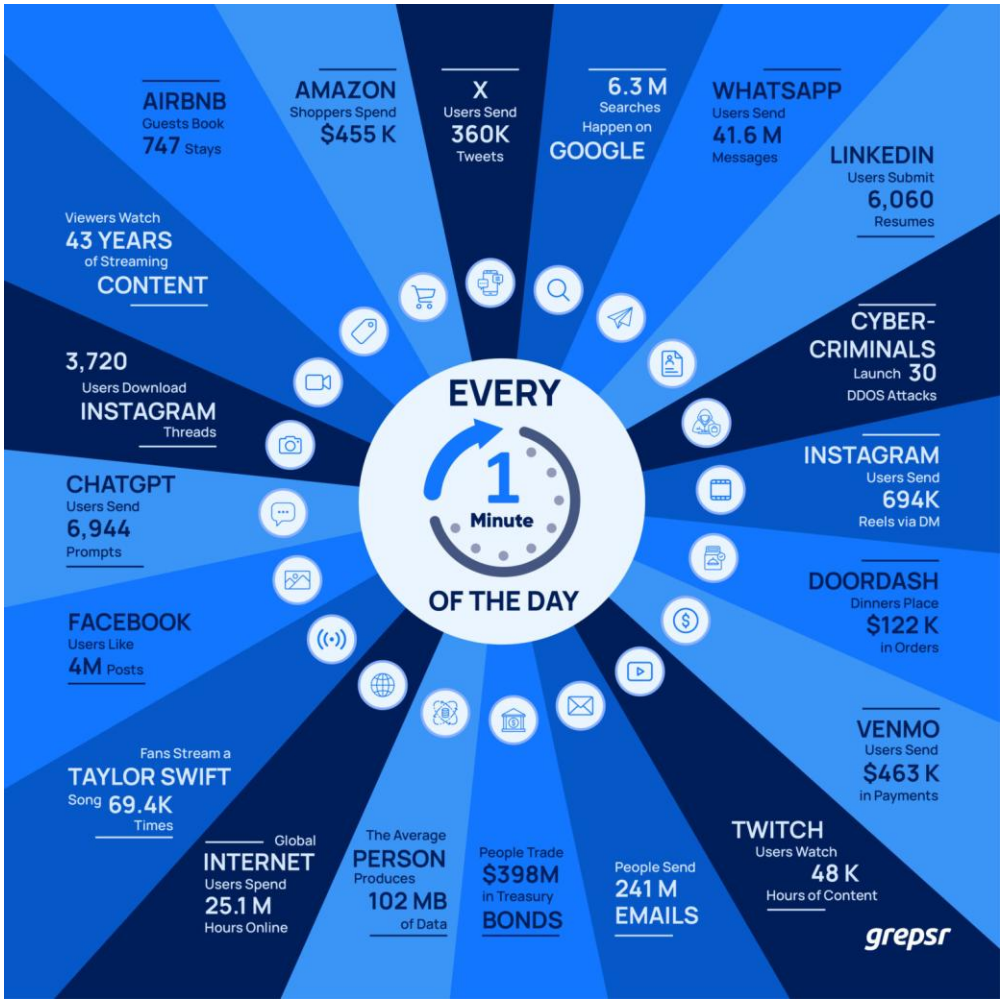
The 5Vs

- Volume
- **Velocity**

Big Data

The 5Vs

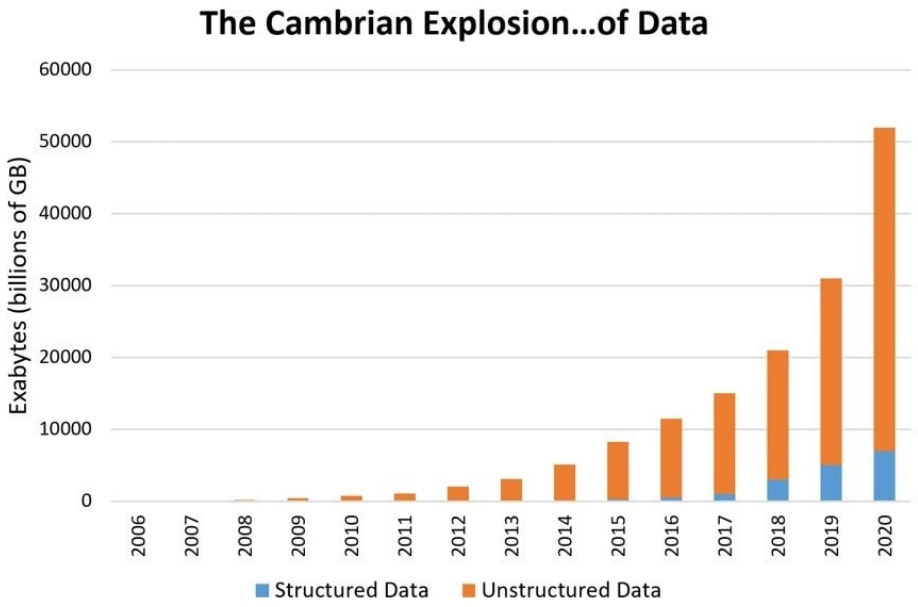
- Volume
- **Velocity**

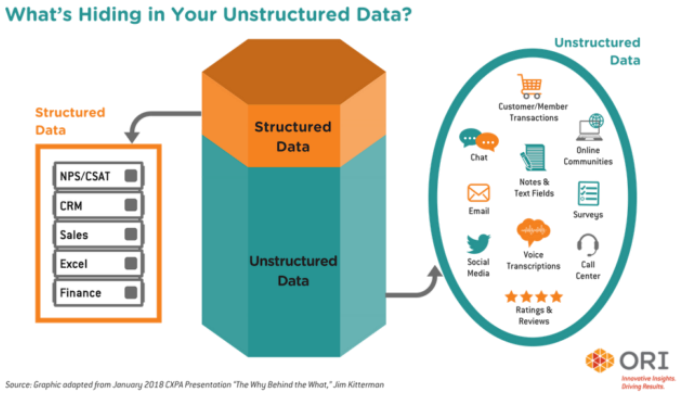


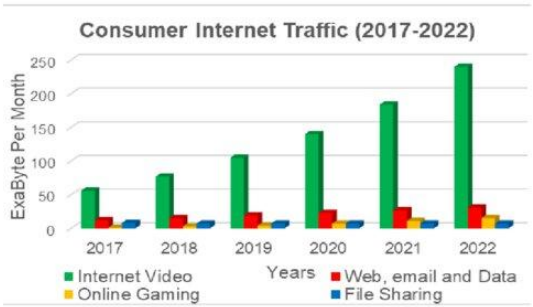
Big Data

The 5Vs

- Volume
- Velocity
- **Variety**



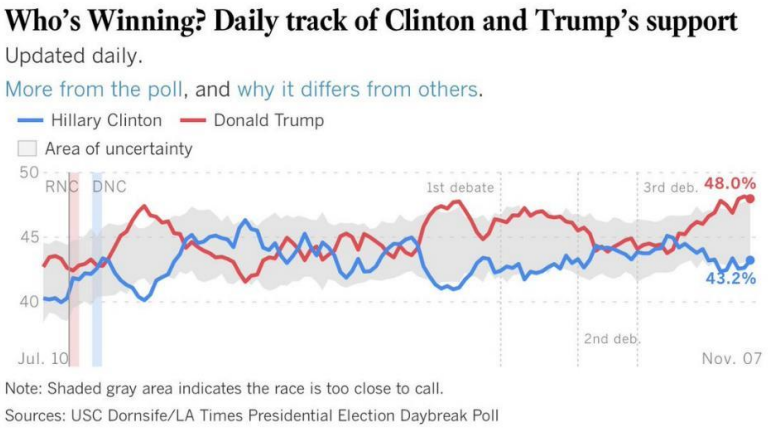


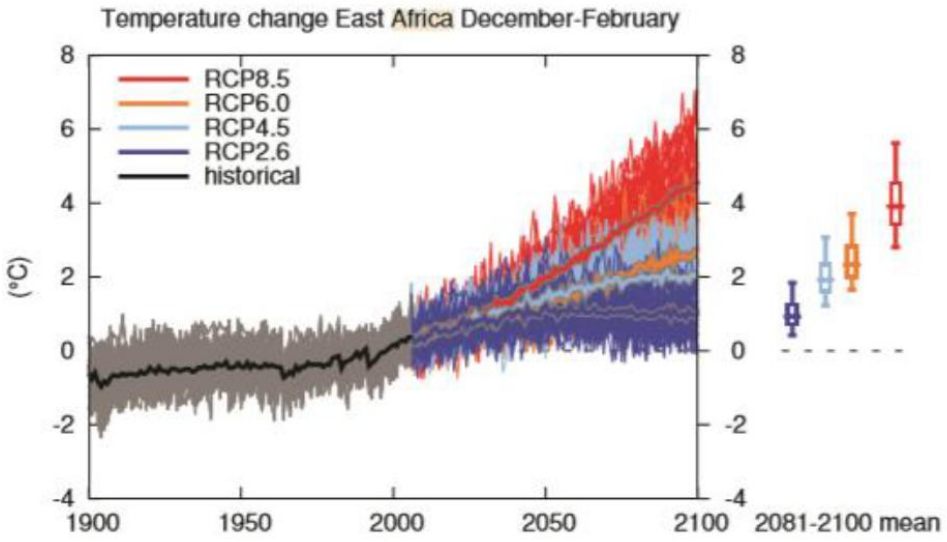


Big Data

The 5Vs

- Volume
- Velocity
- Variety
- **Veracity**

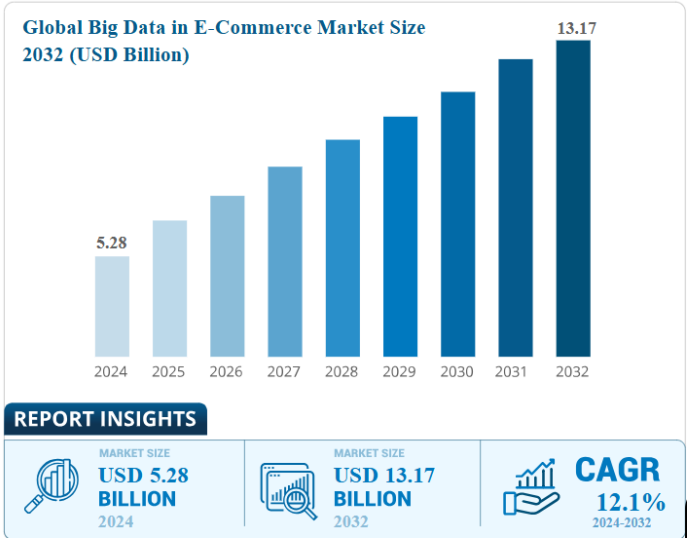


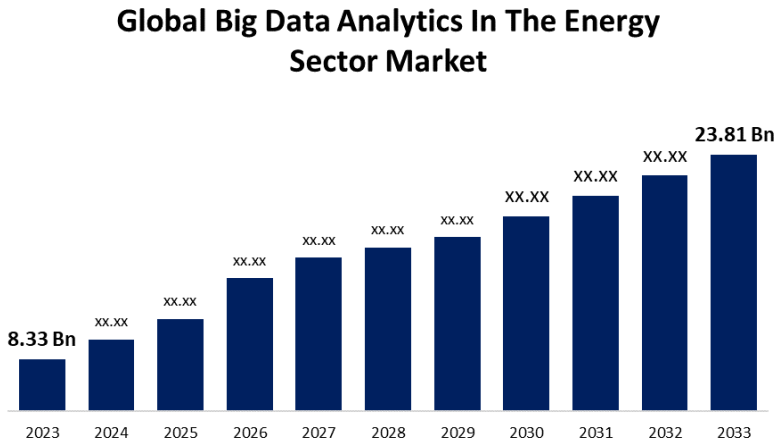


Big Data

The 5Vs

- Volume
- Velocity
- Variety
- Veracity
- **Value**

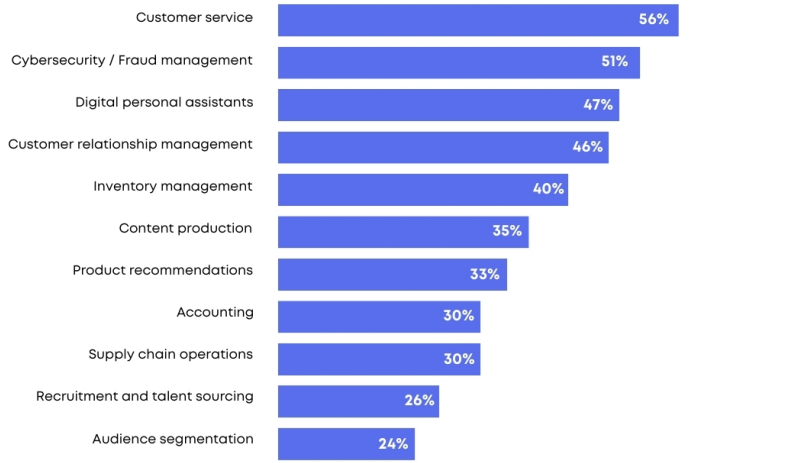


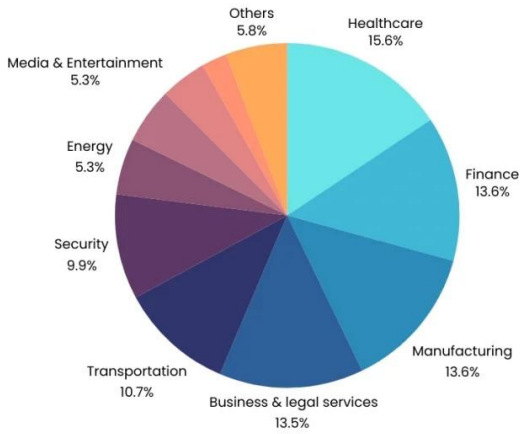


Business

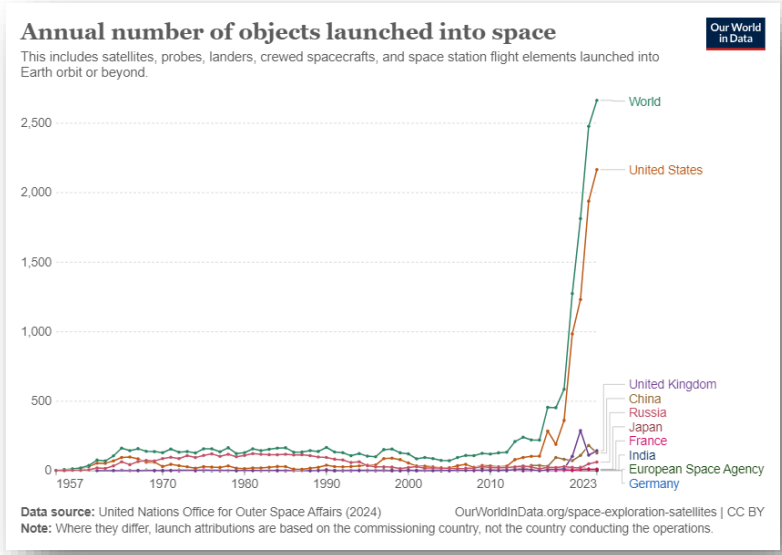

Big Data in Retail
Target increases revenue by
15%
Walmart reduces overstocked inventory and stockouts by
30%
Apple increases customer satisfaction rate by
25%
grepsr


Big Data in Finance
Big Data Analytics in FinTech to grow to
$141.5
billion
American Express reduces fraudulent transactions by
60%
Advanced Analytics and Big Data can generate a
$250 billion
annual value for banks
grepsr


Top Ways Business Owners Use Artificial Intelligence
Customer service 56%
Cybersecurity / Fraud management 51%
Digital personal assistants 47%
Customer relationship management 46%
Inventory management 40%
Content production 35%
Product recommendations 33%
Accounting 30%
Supply chain operations 30%
Recruitment and talent sourcing 26%
Audience segmentation 24%

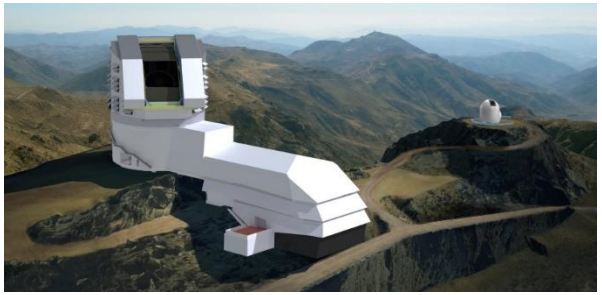

AI Technologies Market Share
Others 5.8%
Healthcare 15.6%
Media & Entertainment 5.3%
Energy 5.3%
Finance 13.6%
Security 9.9%
Manufacturing 13.6%
Transportation 10.7%
Business & legal services 13.5%

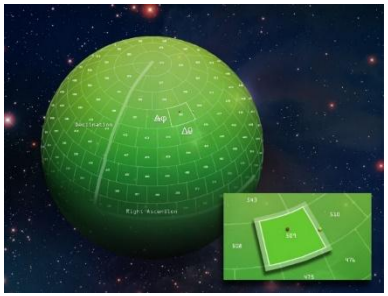
Earth Observation



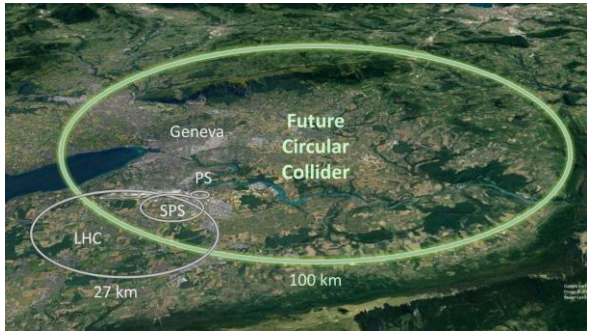
Astrophysics

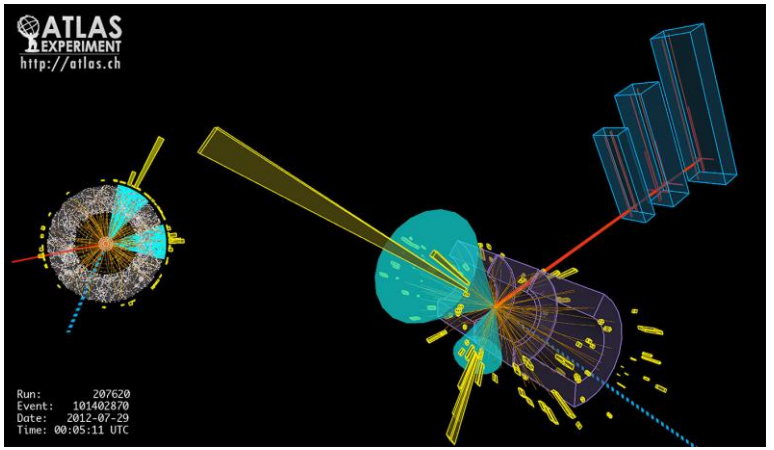
Sky Survey Project	Volume	Velocity	Variety
The Palomar Digital Sky Survey	3 PB		
Sloan Digital Sky Survey (SDSS)	50 TB	200 GB per day	Images, redshifts
Large Synoptic Survey Telescope (LSST)	~ 200 PB	10 TB per day	Images, catalogs
Square Kilometer Array (SKA)	~ 4.6 EB	150 TB per day	Images, redshifts

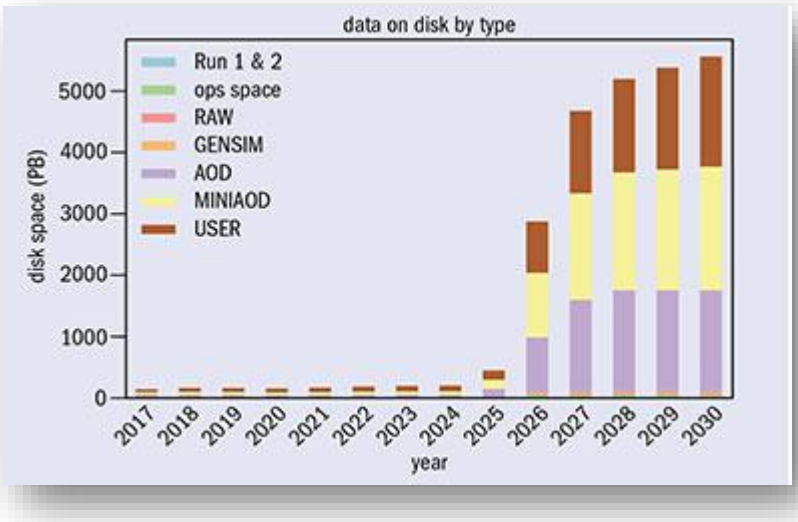


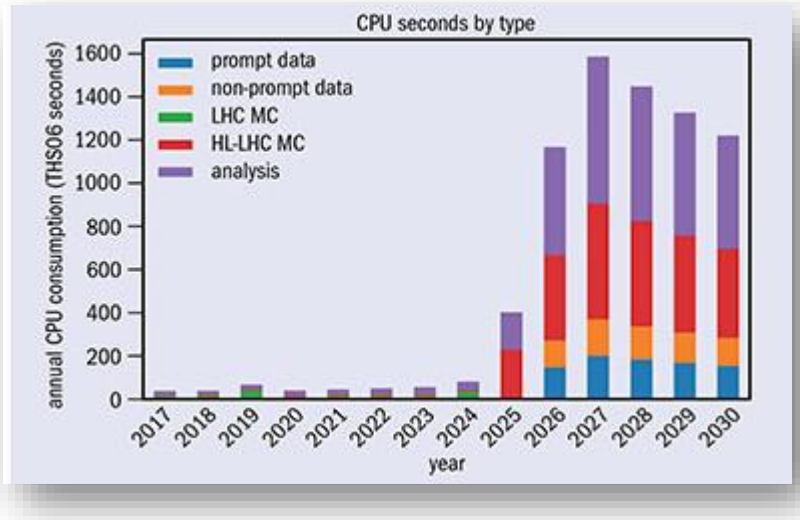


Nuclear physics

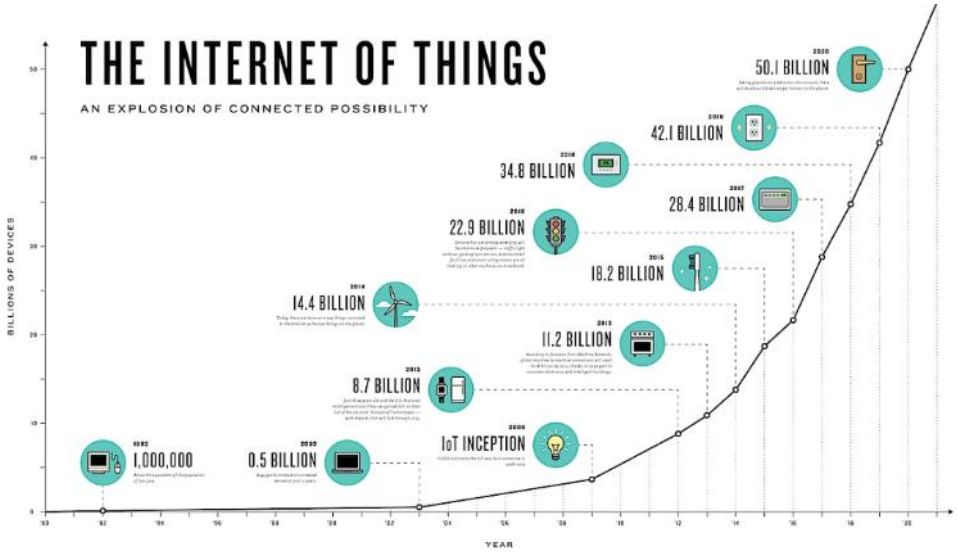


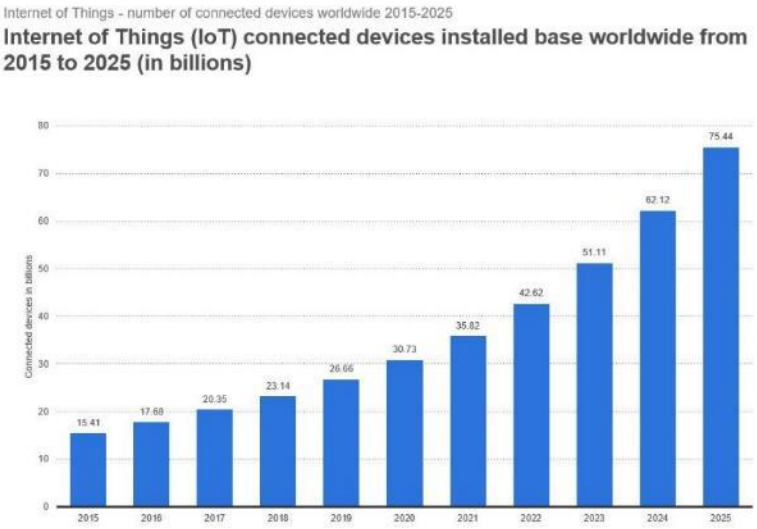




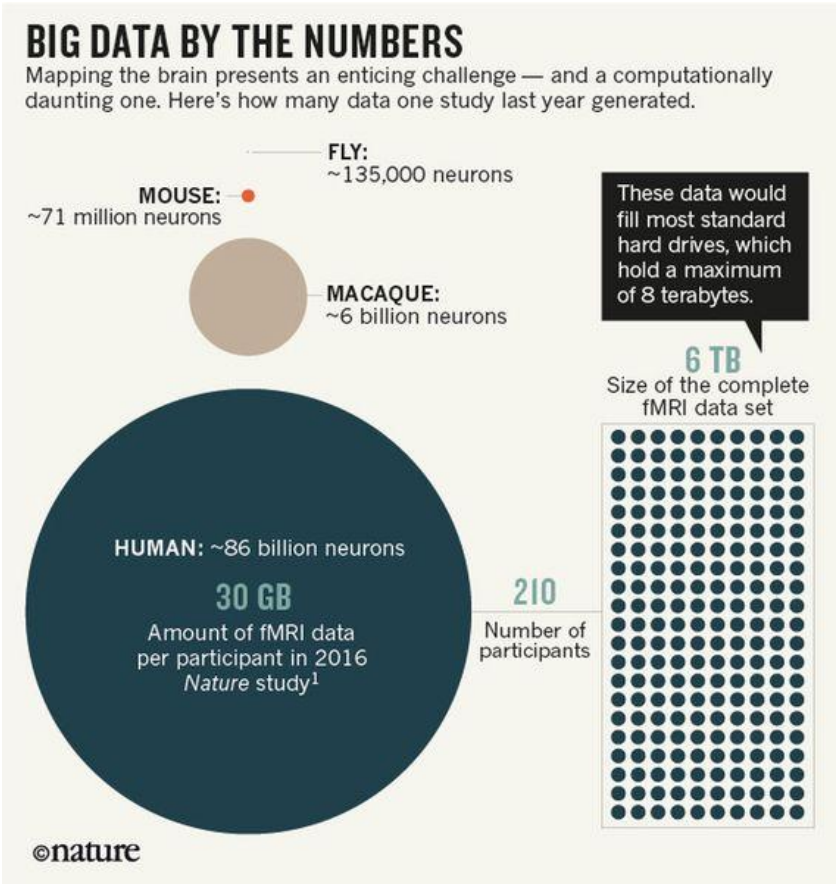


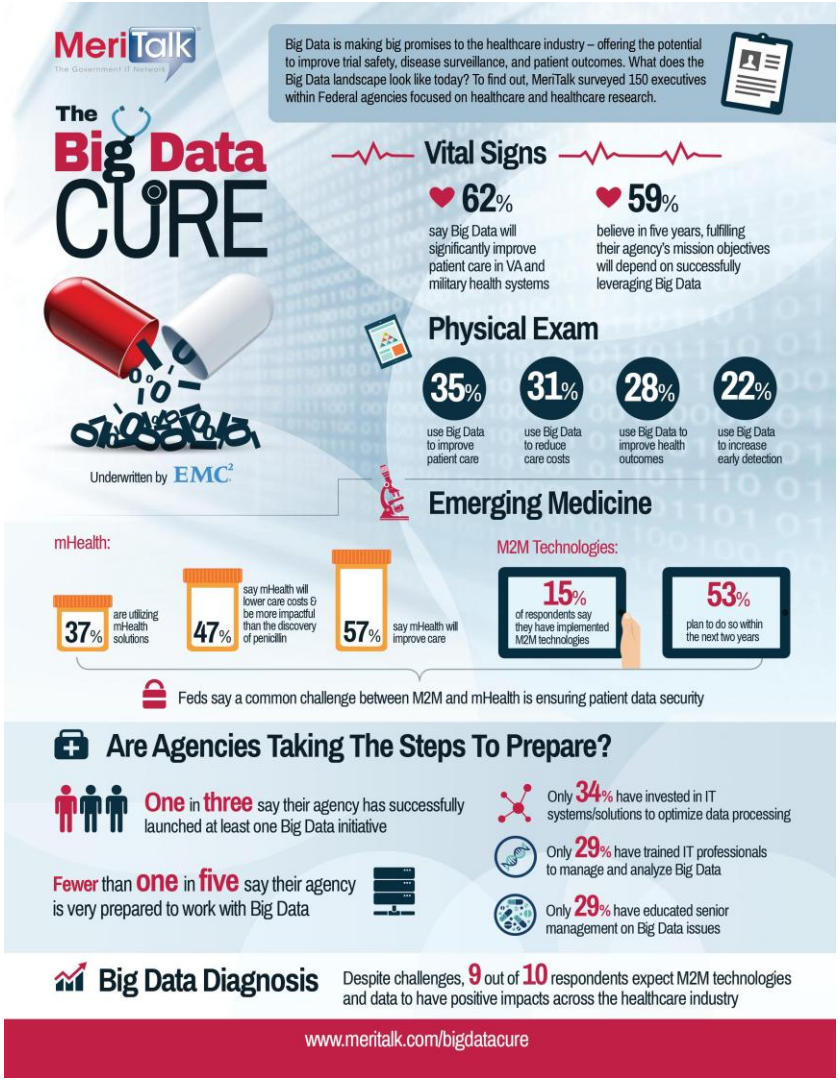
Internet-of-Things





Biology & Medicine signals





What Is Machine Learning?

...creating and using models that learn from data...

- **data**: anything you can *measure* or *record*

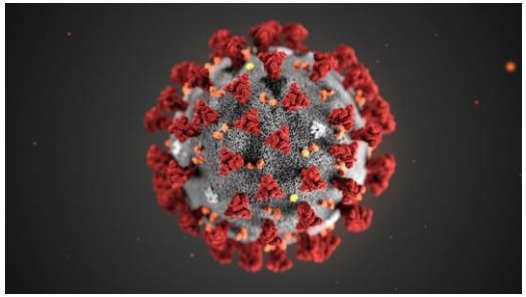
- **model**: specification of a (mathematical) *relationship* between different variables

- **evaluation**: how well does the model *work*?

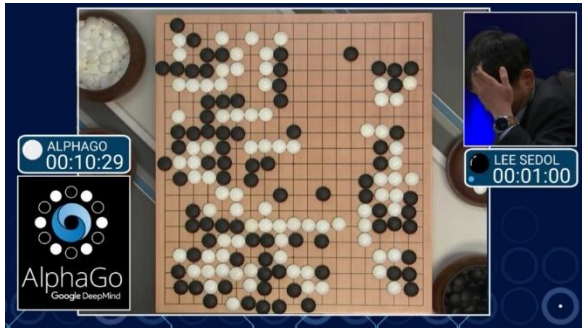
The evolution of AI

Artificial intelligence-created medicine to be used on humans for first time, BBC news, 30/1/2020

https://www.bbc.co.uk/sounds/play/m000dsc4



Coronavirus: Can AI Make A Difference? Forbes, 2/2/2020



Post 2023

D Make a song about a programmer and someone who isn't a programmer.

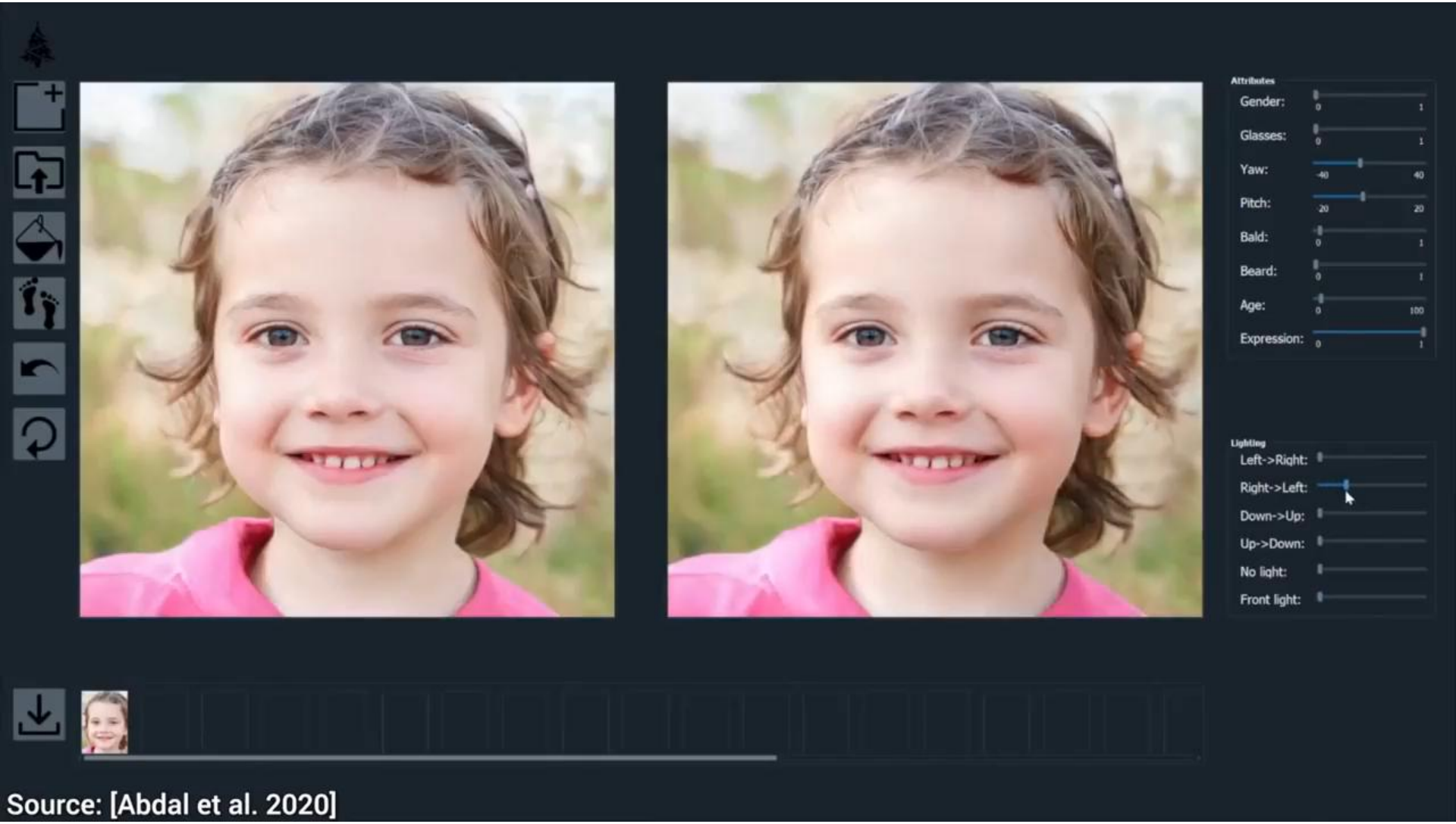
Here is a sample song about a programmer and someone who isn't a programmer:

Verse 1:
Programmer, programmer, sitting at the screen
Typing code all day, living in a dream
Creating apps and websites, making them just right
A master of technology, always up to date

Chorus:
Programmer, programmer, so smart and so bright
Always coding away, day and night

Verse 2:
Non-programmer, non-programmer, not a clue
About the world of code, what it can I

"An astronaut riding a horse in a photorealistic style"

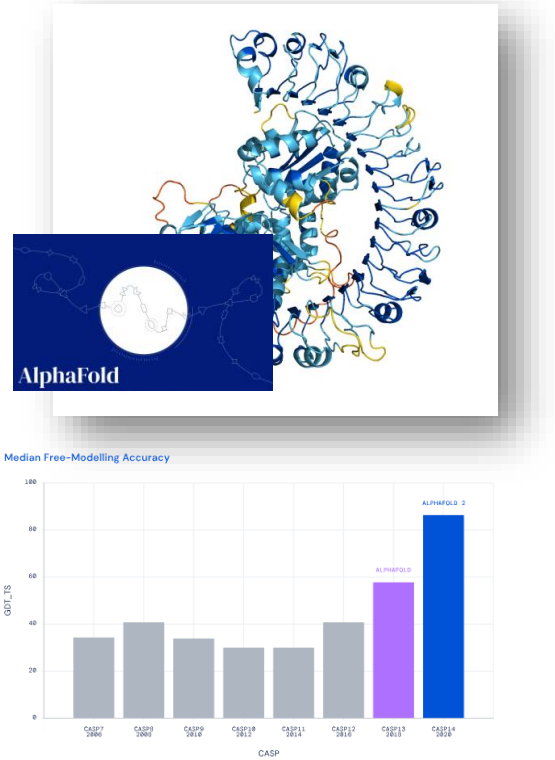


Source: [Abdal et al. 2020]

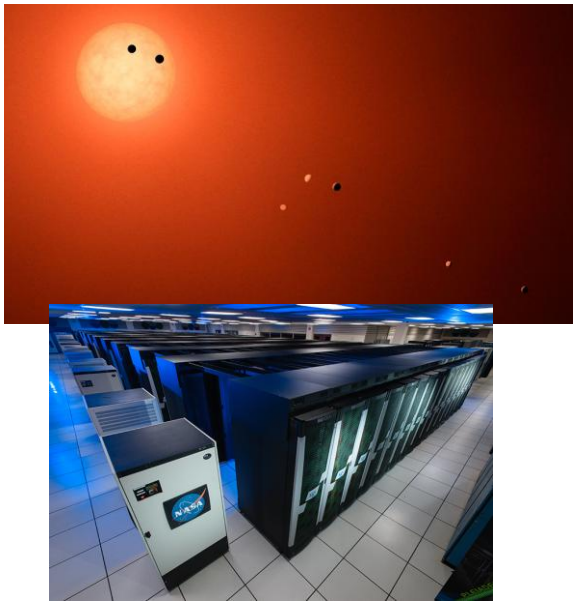
https://youtu.be/7akzhpx0EIU?si=SwPa0Cc1mUENy69C

Sora by OpenAI

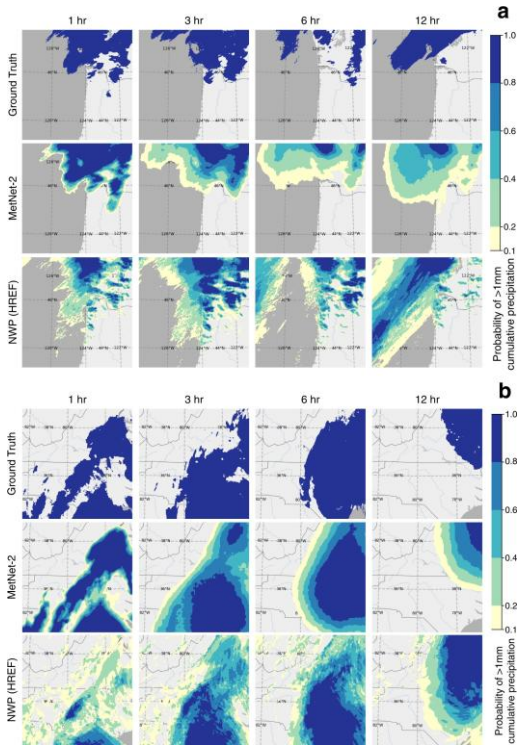
ML in scientific discovery



Jumper, John, et al. "Highly accurate protein structure prediction with AlphaFold." *Nature.* 2021.



Valizadegan, Hamed, et al. "ExoMiner: A highly accurate and explainable deep learning classifier that validates 301 new exoplanets." *The Astrophysical Journal,* 2022)



Espeholt, Lasse, et al. "Deep learning for twelve hour precipitation forecasts." *Nature communications,* (2022).

Brief history of Machine Learning

1958 Perceptron

1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition

1998

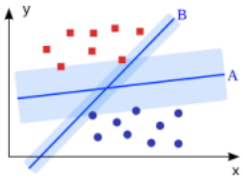
Google Brain Project on 16k Cores

2012

awkward silence (AI Winter)

1969
Perceptron criticized

1995
SVM reigns

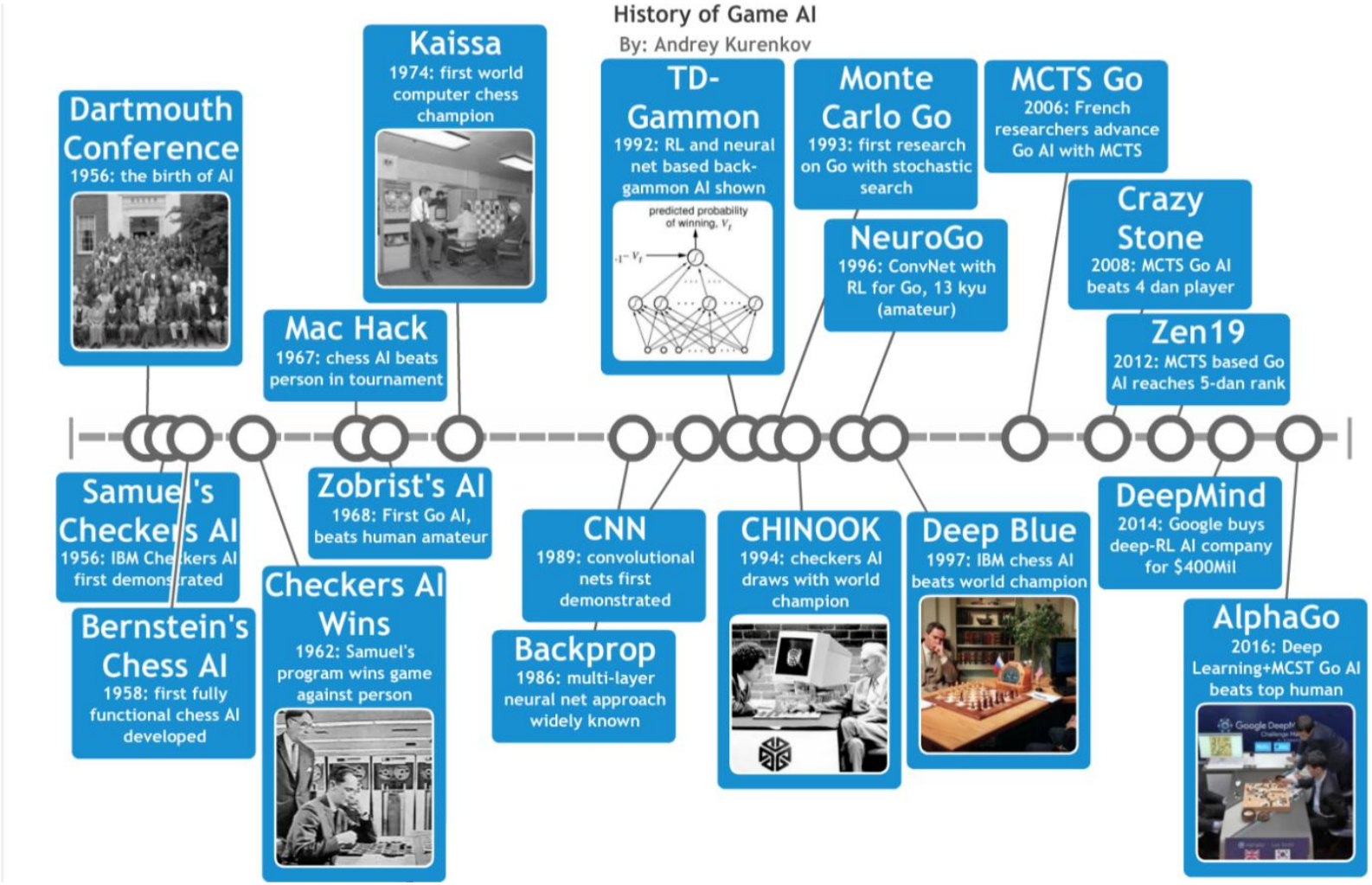


2006
Restricted Boltzmann Machine

2012
AlexNet wins ImageNet

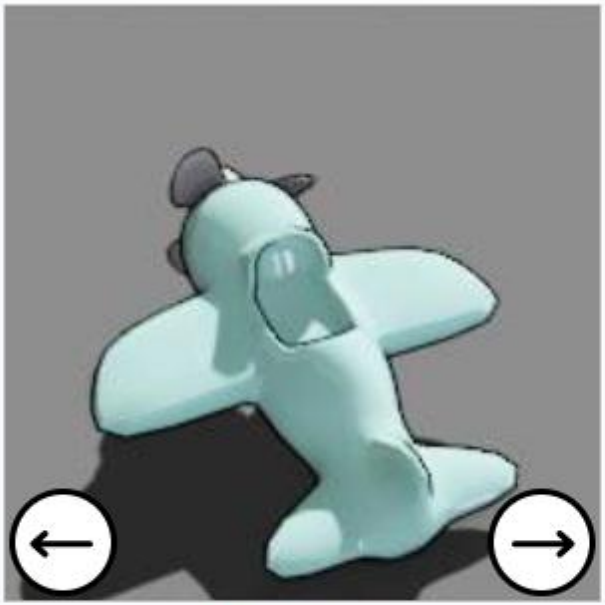
IMAGENET

Applications in games



Verify your account

Use the arrows to **rotate the object** to face in the **direction of the hand.** (1 of 1)



Submit

4451822c1b14e92f4.4001126505

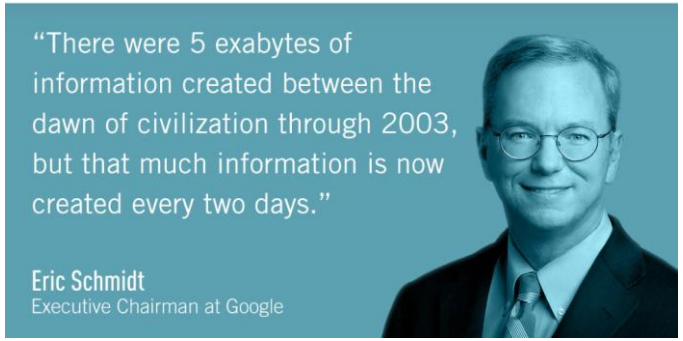
Audio

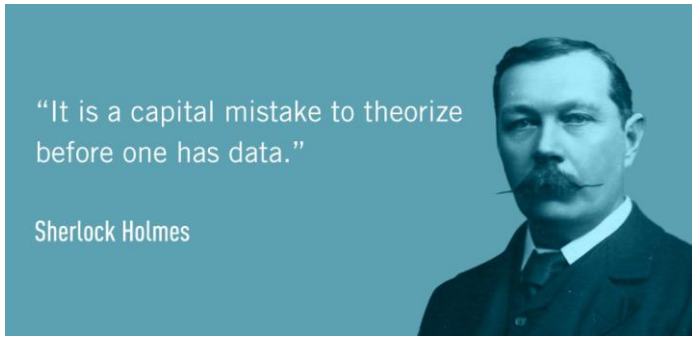
Restart

What is data science

What is data science

- Harvard Business Review: Data Scientist Is The 'Sexiest Job Of The 21st Century', 2012
- "Data scientist is someone who knows more statistics than a computer scientist and more computer science than a statistician."





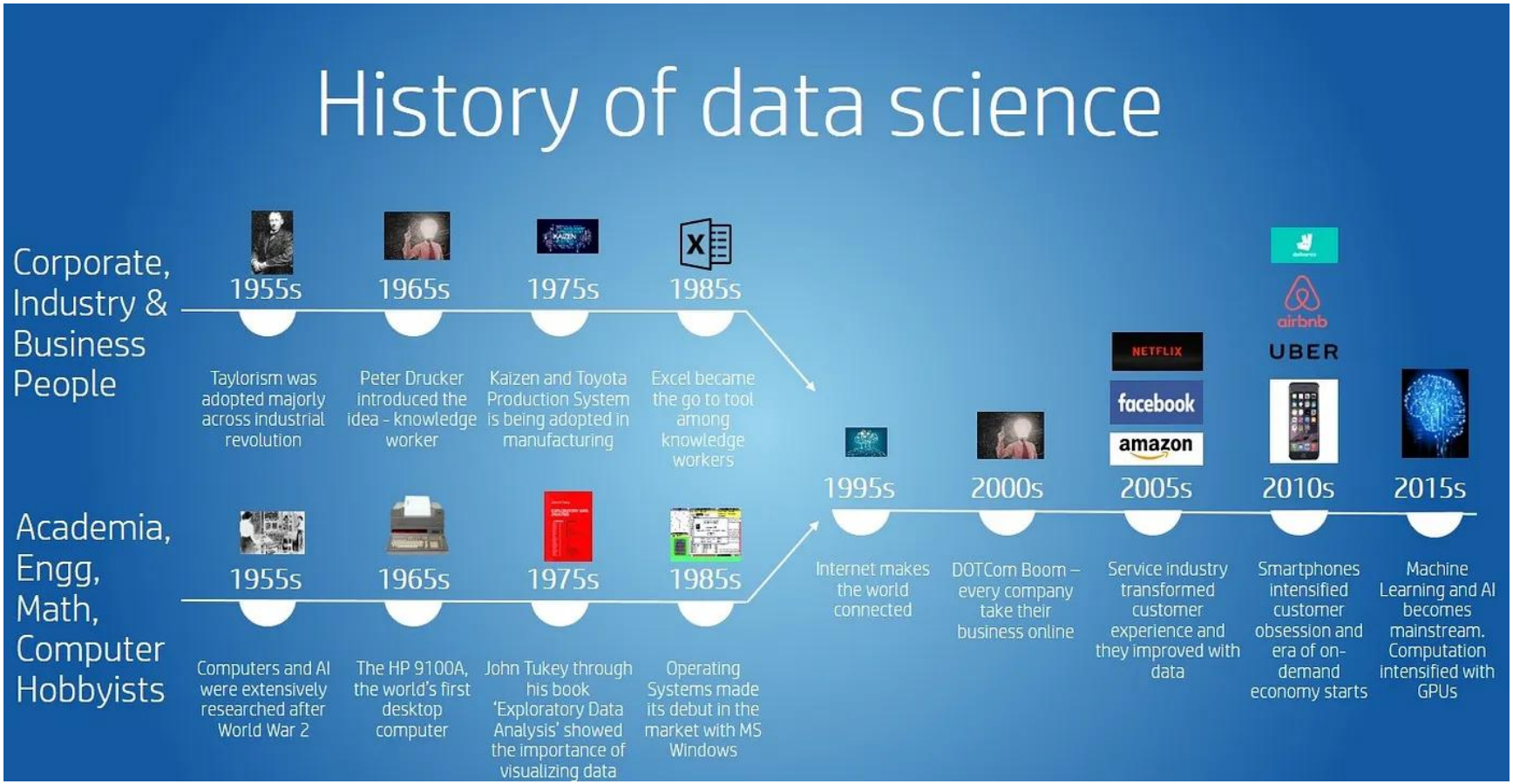
Customer Churn Prediction for a Telecom Company

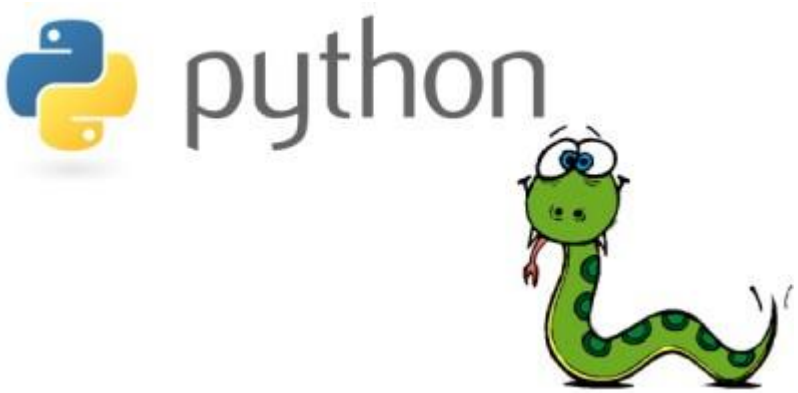
- Objective: To develop a predictive model that identifies customers at high risk of churning.
- Data Gathering and Preparation:
  - Collect Data: Gather historical data e.g. customer demographics, call records, billing history, customer service interactions, etc.
  - Clean Data: Address missing values, remove duplicates, and correct inconsistencies.
  - Feature Engineering: Create new features e.g. average call duration, frequency of calls to customer service
- Exploratory Data Analysis (EDA):
  - Statistical Analysis: Perform statistical tests to understand the distribution of key variables, outliers.
  - Data Visualization: Use graphs and plots to uncover patterns and relationships in the data.

Customer Churn Prediction for a Telecom Company

- Model Building and Validation:
  - Choose Algorithms: Select appropriate machine learning algorithms for analysis.
  - Train Models: Use a training dataset to build various predictive models.
  - Cross-Validation: Implement techniques like k-fold cross-validation to evaluate model performance.
- Deployment and Monitoring:
  - Deploy the Model: Integrate the best-performing model into the company's IT infrastructure so it can score customers in real-time or on a scheduled basis.
  - Monitor Performance: Regularly assess the model's performance, and update it as necessary to maintain accuracy over time.
- Insights and Business Strategy:
  - Interpret Results: Identify key factors contributing to customer churn.
  - Strategic Recommendations: Provide actionable insights e.g. targeted offers for at-risk customers

History


History of data science
Corporate, Industry & Business People
1955s
1965s
1975s
1985s
Taylorism was adopted majorly across industrial revolution
Peter Drucker introduced the idea - knowledge worker
Kaizen and Toyota Production System is being adopted in manufacturing
Excel became the go to tool among knowledge workers
Academia, Engg, Math, Computer Hobbyists
1955s
1965s
1975s
1985s
Computers and AI were extensively researched after World War 2
The HP 9100A, the world's first desktop computer
John Tukey through his book 'Exploratory Data Analysis' showed the importance of visualizing data
Operating Systems made its debut in the market with MS Windows
1995s
2000s
2005s
2010s
2015s
Internet makes the world connected
DOTCom Boom – every company take their business online
Service industry transformed customer experience and they improved with data
Smartphones intensified customer obsession and era of on-demand economy starts
Machine Learning and AI becomes mainstream. Computation intensified with GPUs
NETFLIX
facebook
amazon
airbnb
UBER



Introduction to Python

Overview of Python

Disclaimer!!!

- I am not a programmer
- Most of you are better than me (in python ☺)
- *Programming is the way to translate algorithms into software*

Key Technologies

Python

Jupyter Notebooks

Google Colab

Basic syntax

"Hello, World"

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
   printf("Hello, World!\n");
}
```

- Java

```
public class Hello
{
   public static void main(String argv[])
   {
      System.out.println("Hello, World!");
   }
}
```

- now in Python

```
print "Hello, World!"
```

Formatting

```
# The pound sign marks the start of a comment. Python itself
# ignores the comments, but they're helpful for anyone reading the code.
for i in [1, 2, 3, 4, 5]:
    print(i)                    # first line in "for i" block
    for j in [1, 2, 3, 4, 5]:
        print(j)                # first line in "for j" block
        print(i + j)            # last line in "for j" block
    print(i)                    # last line in "for i" block
print("done looping")
```

- Whitespace is ignored inside parentheses and brackets

```
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 +
                           13 + 14 + 15 + 16 + 17 + 18 + 19 + 20)

list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

easier_to_read_list_of_lists = [[1, 2, 3],
                                [4, 5, 6],
                                [7, 8, 9]]
```

Modules

- Certain features of Python are not loaded by default. These include both features that are included as part of the language as well as third-party features that you download yourself

```
import re
my_regex = re.compile("[0-9]+", re.I)
```

- If you already had a different re in your code, you could use an alias:

```
import re as regex
my_regex = regex.compile("[0-9]+", regex.I)
```


Modules

- Also useful for not writing a lot

```
import matplotlib.pyplot as plt

plt.plot(...)
```

- Import components explicitly

```
from collections import defaultdict, Counter
lookup = defaultdict(int)
my_counter = Counter()
```

- Don’t be a bad person

```
match = 10
from re import *    # uh oh, re has a match function
print(match)        # "<function match at 0x10281e6a8>"
```

Functions

- A function is a rule for taking zero or more inputs and returning a corresponding output.
- In Python, we typically define functions using def:

```
def double(x):
    """
    This is where you put an optional docstring that explains what the
    function does. For example, this function multiplies its input by 2.
    """
    return x * 2
```


Functions

- Python functions are *first-class*, which means that we can assign them to variables and pass them into functions just like any other arguments:

```
def apply_to_one(f):
    """Calls the function f with 1 as its argument"""
    return f(1)
```

```
my_double = double              # refers to the previously defined function
x = apply_to_one(my_double)     # equals 2
```

- It is also easy to create short anonymous functions, or *lambdas*:

```
y = apply_to_one(lambda x: x + 4)      # equals 5
```

Input arguments

- Function parameters can also be given default arguments, which only need to be specified when you want a value other than the default:

```
def my_print(message = "my default message"):
    print(message)

my_print("hello")   # prints 'hello'
my_print()          # prints 'my default message'
```

- specify arguments by name

```
def full_name(first = "What's-his-name", last = "Something"):
    return first + " " + last

full_name("Joel", "Grus")      # "Joel Grus"
full_name("Joel")              # "Joel Something"
full_name(last="Grus")         # "What's-his-name Grus"
```


Multiple returns

- You can have a function return multiple outputs.

In:

```
def multiply_function(a, b):
    result = a * b
    result2 = result * result
    return result, result2

numbers_list = [1, 2, 3]
multiplier_list = [2, 4]
for n in numbers_list:
    print("__________")
    for m in multiplier_list:
        current_answer, current_answer2 = multiply_function(n, m)
        print("The answer to %d * %d is: " %(n, m), current_answer)
        print("The result of this squared is: ", current_answer2)
```

Out:

```
__________
The answer to 1 * 2 is:  2
The result of this squared is:  4
The answer to 1 * 4 is:  4
The result of this squared is:  16
__________
The answer to 2 * 2 is:  4
The result of this squared is:  16
The answer to 2 * 4 is:  8
The result of this squared is:  64
__________
The answer to 3 * 2 is:  6
The result of this squared is:  36
The answer to 3 * 4 is:  12
The result of this squared is:  144
```

Strings

- Use single or double quotes

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
```

- Special characters

```
tab_string = "\t"       # represents the tab character
len(tab_string)         # is 1
```

- *raw* characters

```
not_tab_string = r"\t"  # represents the characters '\' and 't'
len(not_tab_string)     # is 2
```

- Multiline strings

```
multi_line_string = """This is the first line.
and this is the second line
and this is the third line"""
```

Formatted strings

```
full_name1 = first_name + " " + last_name             # string addition
full_name2 = "{0} {1}".format(first_name, last_name)  # string.format
```

- f-string

```
full_name3 = f"{first_name} {last_name}"
```

Variable types

- Integer (int)
- Float (float)
- String (str)
- Boolean (bool)
- Complex (complex)
- […]
- User defined (classes)

- A variable is assigned using the = operator; i.e:

In:
```
intVar = 5
floatVar = 3.2
stringVar = "Food"

print(intVar)
print(floatVar)
print(stringVar)
```

Out:
```
In [4]: runfi
lb690/.spyder
5
3.2
Food
```

- The print() function is used to print something to the screen.

- You can always check the type of a variable using the type() function.

In:
```
variable = 100
print(type(variable))
```

Out:
```
<class 'int'>
```

Type Casting

- Variables can be *cast* to a different type.

In:

```
share_of_rent = 295.50/2.0
print("1:", share_of_rent)
print(type(share_of_rent))
rounded_share = int(share_of_rent)
print("2:", rounded_share)
print(type(rounded_share))
```

Out:

```
1: 147.75
<class 'float'>
2: 147
<class 'int'>
```

Arithmetic operators

The arithmetic operators:

- Addition: +
- Subtract: -
- Multiplication: *
- Division: /
- Power: **

Comparison operators

- Greater than: >
- Lesser than: <
- Greater than or equal to: >=
- Lesser than or equal to: <=
- Is equal to: ==

- Write a couple of operations using comparison operators; i.e.

In:

```
intVar = 5
floatVar = 3.2
stringVar = "Food"

if intVar > floatVar:
    print("Yes")

if intVar == 5:
    print("A match!")
```

Out:

```
In [9]: run
lb690/.spyd
Yes
A match!
```

Working with strings

In:

```
greeting = 'Hello, Lew!'
print('1:', greeting)
print('2:', len(greeting))
print('3:', greeting[0])
print('4:', greeting[-1])
greeting = greeting.replace("Lew", "class")
print('5:', greeting)
string1 = "Hello"
string2 = "world"
print("1:", string1, string2)
cost = float(35.28)
print("Bar tab = £%f" %cost)
```

Out:

```
1: Hello, Lew!
2: 11
3: H
4: !
5: Hello, class!
1: Hello world
Bar tab = £35.280000
```

Dictionaries

- Dictionaries are lists of key-valued pairs.

In:
```
prices = {"Eggs": 2.30,
          "Steak": 13.50,
          "Bacon": 2.30,
          "Beer": 14.95}
print("1:", prices)
print("2:", type(prices))
print("The price of bacon is:", prices["Bacon"])
```

Out:
```
1: {'Eggs': 2.3, 'Steak': 13.5, 'Bacon': 2.3, 'Beer':
14.95}
2: <class 'dict'>
The price of bacon is: 2.3
```

Indexing

- Indexing in Python is 0-based, meaning that the first element in a string, list, array, etc, has an index of 0. The second element then has an index of 1, and so on.

In:
```
test_string = "Dogs are better than cats"
print('First element:', test_string[0])
print('Second element:', test_string[1])
```

Out:
```
First element: D
Second element: o
```

- You can cycle backwards through a list, string, array, etc, by placing a minus symbol in front of the index location.

In:
```
test_string = "Dogs are better than cats"
print('Last element:', test_string[-1])
print('Second to last element:', test_string[-2])
```

Out:
```
Last element: s
Second to last element: t
```

In:
```
test_string = "Dogs are better than cats"
print('Last element:', test_string[4:])
print('Second to last element:', test_string[:4])
```

Out:
```
Last element:  are better than cats
Second to last element: Dogs
```

In:
```
test_string = "Dogs are better than cats"
print(test_string[5:10])
```

Out:
```
are b
```

Lists

- Lists are essentially containers of arbitrary type.
- The elements of a list can be of different types.
- The difference between tuples and lists is in performance; it is much faster to 'grab' an element stored in a tuple, but lists are much more versatile.
- Note that lists are denoted by [] and not the () used by tuples.

In:

```
numbers = [1, 2, 3]
print("List 1:", numbers)
print("Type of list 1:", type(numbers))
arbitrary_list = [1, numbers, "Hello"]
print("Arbitrary list:", arbitrary_list)
print("Type of arbitrary list:", type(arbitrary_list))
```

Out:

```
List 1: [1, 2, 3]
Type of list 1: <class 'list'>
Arbitrary list: [1, [1, 2, 3], 'Hello']
Type of arbitrary list: <class 'list'>
```

Adding elements to a list

- Lists are mutable; i.e. their contents can be changed. This can be done in a number of ways.
- With the use of an index to replace a current element with a new one.

In:
```
numbers = [1, 2, 3]
print("List 1:", numbers)
numbers[1] = 5
print("Amended list 1:", numbers)
```

Out:
```
List 1: [1, 2, 3]
Amended list 1: [1, 5, 3]
```

- You can use the insert() function in order to add an element to a list at a specific indexed location, without overwriting any of the original elements.

In:
```
numbers = [1, 2, 3]
print("List 1:", numbers)
numbers.insert(2, 'Surprise!')
print("Amended list 1:", numbers)
```

Out:
```
List 1: [1, 2, 3]
Amended list 1: [1, 2, 'Surprise!', 3]
```

- You can add an element to the end of a list using the append() function.

In:
```
numbers = [1, 2, 3]
print("List 1:", numbers)
numbers.append(4)
print("Amended list 1:", numbers)
```

Out:
```
List 1: [1, 2, 3]
Amended list 1: [1, 2, 3, 4]
```

Removing elements from a list

- You can remove an element from a list based upon the element value.
- Remember: If there is more than one element with this value, only the first occurrence will be removed.

In:
```
numbers = [1, 2, 3, 3]
print("List 1:", numbers)
numbers.remove(3)
print("Amended list 1:", numbers)
```

Out:
```
List 1: [1, 2, 3, 3]
Amended list 1: [1, 2, 3]
```

- It is better practice to remove elements by their index using the del function.

In:
```
numbers = [1, 2, 3, 4]
print("List 1:", numbers)
del numbers[1]
print("Amended list 1:", numbers)
del numbers[-1]
print("Amended list 2:", numbers)
```

Out:
```
List 1: [1, 2, 3, 4]
Amended list 1: [1, 3, 4]
Amended list 2: [1, 3]
```

For loops

- The for loop is used to iterate over elements in a sequence, and is often used when you have a piece of code that you want to repeat a number of times.
- For loops essentially say:

“For all elements in a sequence, do something”

An example

- We have a list of species:

```
species = ['dog', 'cat', 'shark', 'falcon', 'deer', 'tyrannosaurus rex']
for i in species:
    print(i)
```

- The command underneath the list then cycles through each entry in the species list and prints the animal's name to the screen. Note: The i is quite arbitrary. You could just as easily replace it with 'animal', 't', or anything else.

```
In [1]: runfile('//is
dog
cat
shark
falcon
deer
tyrannosaurus rex
```

Another example

- We can also use for loops for operations other than printing to a screen. For example:

```
numbers = [1, 20, 18, 5, 15, 160]
total = 0
for value in numbers:
    total = total + value
print(total)
```

```
In [4]: runfile('
219
```


The range() function

- The range() function generates a list of numbers, which is generally used to iterate over within for loops.
- The range() function has two sets of parameters to follow:

range(stop)
stop: Number of integers (whole numbers) to generate, starting from zero. i.e:

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

```
In [6]: runfile('
0
1
2
3
4
```

range([start], stop[, step])
start: Starting number of the sequence.
stop: Generate numbers up to, but not including this number.
step: Difference between each number in the sequence
i.e.:

```
for i in range(3,6):
    print(i)
```

```
In [7]: runfile('/
3
4
5
```

```
for i in range(4, 10, 2):
    print(i)
```

```
In [8]: runfile
4
6
8
```

Note:

- All parameters must be integers.
- Parameters can be positive or negative.
- The range() function (and Python in general) is 0-index based, meaning list indexes start at 0, not 1. eg. The syntax to access the first element of a list is mylist[0]. Therefore the last integer generated by range() is up to, but not including, stop.

Example

- Create an empty list.

```
new_list = []
```

- Use the range() and append() functions to add the integers 1-20 to the empty list.

```
for i in range(1, 21):
    new_list.append(i)
```

- Print the list to the screen, what do you have?

```
print(new_list)
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Loop control

- The break() function
- To terminate a loop, you can use the break() function.
- The break() statement breaks out of the innermost enclosing for or while loop.

```
for i in range(1, 10):
    if i == 3:
        break
    print(i)
```

```
1
2
```

- The continue() statement is used to tell Python to skip the rest of the statements in the current loop block, and then to continue to the next iteration of the loop.

```
for i in range(1, 10):
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
7
8
9
```

While loops

- The while loop tells the computer to do something as long as a specific condition is met.
- When working with while loops, its important to remember the nature of various operators.
- While loops use the break() and continue() functions in the same way as a for loop does.

```
species = ['dog', 'cat', 'shark', 'falcon', 'deer', 'tyrannosaurus rex']
i = 0
while i < 3:
    print(species[i])
    i = i + 1
```

```
dog
cat
shark
```


Nested loops

- In some situations, you may want a loop within a loop; this is known as a nested loop.

In:

```
for x in range(1, 11):
    for y in range(1, 11):
        print('%d * %d = %d' %(x, y, x*y))
```

Out:

```
 1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
```

Conditionals

- There are three main conditional statements in Python; if, else, elif.
- We have already used if when looking at while loops.

In:
```
school_night = True
if school_night == True:
    print("No beer")
else:
    print("You may have beer")
```

Out: No beer

In:
```
school_night = False
if school_night == True:
    print("No beer")
else:
    print("You may have beer")
```

Out: You may have beer

An example of elif

In:

```
Lew_is_tired = False
Lew_is_hungry = True
if Lew_is_tired is True:
    print("Lew has to teach")
elif Lew_is_hungry is True:
    print("No food for Lew")
else:
    print("Go on, have a biscuit")
```

Out:

```
No food for Lew
```


Reading and writing to files in Python

- Before you can work with a file, you first have to open it using Python's in-built `open()` function.
- The `open()` function takes two arguments; the name of the file that you wish to use and the mode for which we would like to open the file

```
practiceFile = open('Practice_file_for_IOC.txt', 'r')
```

- By default, the `open()` function opens a file in 'read mode'; this is what the `'r'` above signifies.
- There are a number of different file opening modes. The most common are: 'r'= read, 'w'=write, 'r+'=both reading and writing, 'a'=appending.
- Likewise, once you're done working with a file, you can close it with the `close()` function.

```
practiceFile.close()
```


Reading in a file and printing to screen example

Using what you have now learned about for loops, it is possible to open a file for reading and then print each line in the file to the screen using a for loop.

In:

```
practiceFile = open('practice_file.txt', 'r')
for line in practiceFile:
    print(line)
```

Out:

```
In [42]: runfile('//isad.isadroot.
User/Desktop')
The first line of text

The second line of text

The third line of text

The fourth line of text

I'm bored now, you get the idea
```

The read() function

- However, you don't need to use any loops to access file contents. Python has three in-built file reading commands:

1. <file>.read() = Returns the entire contents of the file as a single string:

```
practiceFile = open('practice_file.txt', 'r')
print(practiceFile.read())
```

```
In [44]: runfile('//isad.isadroot.e
User/Desktop')
The first line of text
The second line of text
The third line of text
The fourth line of text
I'm bored now, you get the idea
```

2. <file>.readline() = Returns one line at a time:

```
practiceFile = open('practice_file.txt', 'r')
print(practiceFile.readline())
```

```
In [51]: runfile('//isad.i
User/Desktop')
The first line of text
```

3. <file>.readlines() = Returns a list of lines:

```
practiceFile = open('practice_file.txt', 'r')
print(practiceFile.readlines())
```

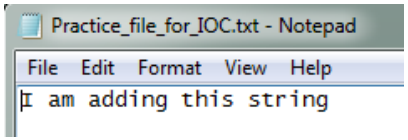
```
In [52]: runfile('//isad.isadroot.ex.ac.uk/UOE/User/Desktop/IOC_test.py',
wdir='//isad.isadroot.ex.ac.uk/UOE/User/Desktop')
['The first line of text\n', 'The second line of text\n', 'The third line of
text\n', 'The fourth line of text\n', "I'm bored now, you get the idea\n"]
```

The write() function

- Likewise, there are two similar in-built functions for getting Python to write to a file:

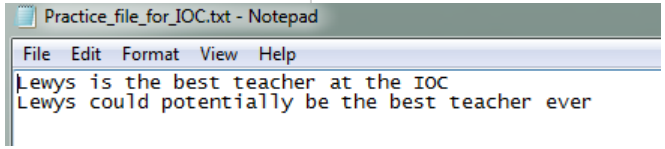
1. <file>.write() = Writes a specified sequence of characters to a file:

```
practiceFile = open('Practice_file_for_IOC.txt', 'w')
practiceFile.write('I am adding this string')
```



2. <file>.writelines() = Writes a list of strings to a file:

```
testList = ['Lewys is the best teacher at the IOC\n', 'Lewys could potentially be the best teacher ever\n']
practiceFile = open('Practice_file_for_IOC.txt', 'w')
practiceFile.writelines(testList)
```

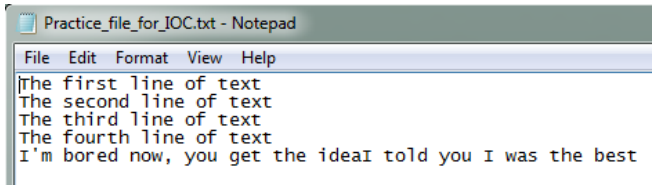


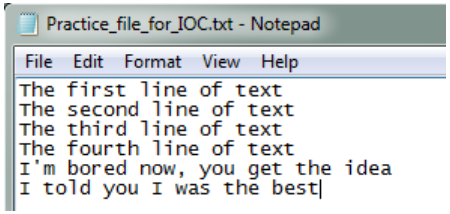
- Important: Using the write() or writelines() function will overwrite anything contained within a file, if a file of the same name already exists in the working directory.

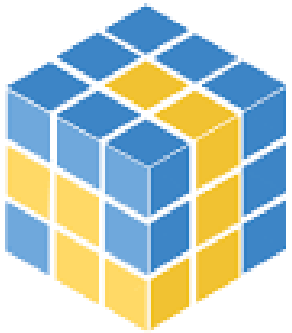
The append() function

- If you do not want to overwrite a file's contents, you can use the append() function.
- To append to an existing file, simply put 'a' instead of 'r' or 'w' in the open() when opening a file.

```
practiceFile = open('Practice_file_for_IOC.txt', 'a')
testLine = '\nI told you I was the best'
practiceFile.write(testLine)
```







NumPy

NumPy

- Stands for Numerical Python
- Is the fundamental package required for high performance computing and data analysis
- NumPy is so important for numerical computations in Python is because it is designed for efficiency on large arrays of data.
- It provides
  - ndarray for creating multiple dimensional arrays
  - Internally stores data in a contiguous block of memory, independent of other built-in Python objects, uses much less memory than built-in Python sequences.
  - Standard math functions for fast operations on entire arrays of data without having to write loops
  - NumPy Arrays are important because they enable you to express batch operations on data without writing any *for* loops. We call this *vectorization*.

Dr. Sampath Jayarathna, Old Dominion University

NumPy ndarray vs list

- One of the key features of NumPy is its N-dimensional array object, or ndarray, which is a fast, flexible container for large datasets in Python.
- Whenever you see “array,” “NumPy array,” or “ndarray” all refer to the same thing: the ndarray object.
- NumPy-based algorithms are generally 10 to 100 times faster (or more) than their pure Python counterparts and use significantly less memory.

In:

```
import numpy as np
list1 = [0, 1, 2, 3, 4]
arr = np.array(list1)

print(type(arr))
print(arr)
```

Out:

```
In [1]: runfile('C:/Users/
wdir='C:/Users/Lew_laptop/
<type 'numpy.ndarray'>
[0 1 2 3 4]
```

ndarray

- ndarray is used for storage of homogeneous data
  - i.e., all elements the same type
- Every array must have a shape and a dtype
- Let's suppose you want to add the number 2 to every item in the list. The intuitive way to do this is something like this:

In:
```
import numpy as np
list1 = [0, 1, 2, 3, 4]
list1 = list1+2
```

Out
```
File "C:/Users/Lew_laptop/.spyder-py3/temp.py", line 9, i
  list1 = list1+2

TypeError: can only concatenate list (not "int") to list
```

- That was not possible with a list, but you can do that on an array:

In:
```
import numpy as np
list1 = [0, 1, 2, 3, 4]
arr = np.array(list1)
print(arr)
arr = arr+2
print(arr)
```

Out:
```
In [7]: runfile('C:/Users
Lew_laptop/.spyder-py3')
[0 1 2 3 4]
[2 3 4 5 6]
```


Creating ndarrays

Using list of lists

In:

```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr2=np.array(list2)
print(arr2)
```

Out:

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Creating ndarrays

```
array = np.array([[0,1,2],[2,3,4]])
[[0 1 2]
 [2 3 4]]
```

```
array = np.zeros((2,3))
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
array = np.ones((2,3))
[[1. 1. 1.]
 [1. 1. 1.]]
```

```
array = np.eye(3)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
array = np.arange(0, 10, 2)
[0, 2, 4, 6, 8]
```

```
array = np.random.randint(0, 10,
(3,3))
[[6 4 3]
 [1 5 6]
 [9 8 5]]
```

arange is an array-valued version of the built-in Python range function

Arithmetic with NumPy Arrays

- Any arithmetic operations between equal-size arrays applies the operation element-wise:

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
print(arr)
[[1. 2. 3.]
 [4. 5. 6.]]
print(arr * arr)
[[ 1.  4.  9.]
 [16. 25. 36.]]
print(arr - arr)
[[0. 0. 0.]
 [0. 0. 0.]]
```


Arithmetic with NumPy Arrays

- Arithmetic operations with scalars propagate the scalar argument to each element in the array:

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
print(arr)
[[1. 2. 3.]
 [4. 5. 6.]]
print(arr **2)
[[ 1.  4.  9.]
 [16. 25. 36.]]
```

- Comparisons between arrays of the same size yield boolean arrays:

```
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
print(arr2)
[[ 0.  4.  1.]
 [ 7.  2. 12.]]

print(arr2 > arr)
[[False  True False]
 [ True False  True]]
```

Indexing and Slicing

- One-dimensional arrays are simple; on the surface they act similarly to Python lists:

```
arr = np.arange(10)
print(arr)        # [0 1 2 3 4 5 6 7 8 9]
print(arr[5])     #5
print(arr[5:8]) #[5 6 7]
arr[5:8] = 12
print(arr)      #[ 0 1 2 3 4 12 12 12 8 9]
```

Indexing and Slicing

- As you can see, if you assign a scalar value to a slice, as in arr[5:8] = 12, the value is propagated (or *broadcasted*) to the entire selection.
- An important first distinction from Python's built-in lists is that array slices are *views* on the original array.
  - This means that the data is not copied, and any modifications to the view will be reflected in the source array.

```
arr = np.arange(10)
print(arr)        # [0 1 2 3 4 5 6 7 8 9]

arr_slice = arr[5:8]
print(arr_slice)          # [5 6 7]
arr_slice[1] = 12345
print(arr)                # [    0     1     2     3     4     5 12345     7     8     9]
arr_slice[:] = 64
print(arr)                # [ 0  1  2  3  4 64 64 64  8  9]
```


Indexing

- In a two-dimensional array, the elements at each index are no longer scalars but rather one-dimensional arrays:

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d[2])        # [7 8 9]
```

- Thus, individual elements can be accessed recursively. But that is a bit too much work, so you can pass a comma-separated list of indices to select individual elements.
- So these are equivalent:

```
print(arr2d[0][2])      # 3
print(arr2d[0, 2])      #3
```


The dtype argument

- You can specify the data-type by setting the dtype() argument.
- Some of the most commonly used NumPy dtypes are: float, int, bool, str, and object.

In:
```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print(arr3)
```

Out:
```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
```

The astype argument

- You can also convert it to a different data-type using the astype method.

In:
```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print(arr3)
arr3_s = arr3.astype('int').astype('str')
print(arr3_s)
```

Out:
```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
[['0' '1' '2']
 ['3' '4' '5']
 ['6' '7' '8']]
```

- Remember that, unlike lists, all items in an array have to be of the same type.

Inspecting a NumPy array

- There are a range of functions built into NumPy that allow you to inspect different aspects of an array:

In:
```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print('Shape:', arr3.shape)
print('Data type:', arr3.dtype)
print('Size:', arr3.size)
print('Num dimensions:', arr3.ndim)
```

Out:
```
Shape: (3, 3)
Data type: float64
Size: 9
Num dimensions: 2
```

Boolean indexing

- A boolean index array is of the same shape as the array-to-be-filtered, but it only contains TRUE and FALSE values.

In:
```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
boo = arr3>2
print(boo)
```

Out:
```
[[False False False]
 [ True  True  True]
 [ True  True  True]]
```