

# 50.039 Theory and Practice of Deep Learning

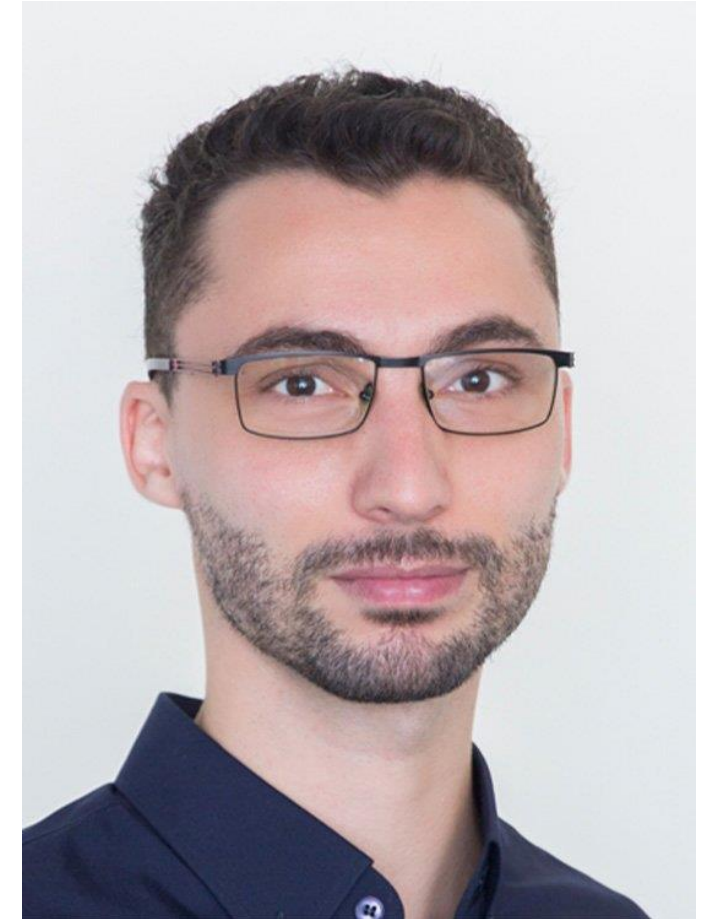
## W1-S1 Introduction and Machine Learning Reminders

Matthieu De Mari



# A quick word about instructors

- Matthieu (**Matt**) De Mari
- Lecturer at SUTD (Python, Deep Learning, AI, and more)
- Information Systems Technology and Design (ISTD) pillar/faculty
- PhD from CentraleSupélec (France)
- Email: [matthieu\\_demari@sutd.edu.sg](mailto:matthieu_demari@sutd.edu.sg)
- Office @ SUTD: 2.401.07



# Objectives of 50.039 DL

- Introduce the technical aspects related to the **implementation of Deep Neural Networks from scratch**.
- Teach the students about the **most popular framework** for implementing Deep Neural Networks (as of Jan 2023): **PyTorch**.
- Discuss the **mathematical foundations and intuitions** behind the Deep Learning framework, layers and some techniques.
- Give the students a **global overview of the advanced techniques** related to Deep Learning and Deep Neural Networks.
- Describe **examples of practical applications** of Deep Learning, showing how some key AIs were implemented using Deep Learning.

# Objectives of 50.039 DL

- Eventually, **assemble all concepts and tricks of the trade** discussed in this course to produce big architecture models that are close to the current state-of-the-art ones (e.g. ChatGPT, Dall-E, etc.).
- Discuss **ethical aspects** related to some of the concepts discussed.
- Give students **an understanding of the current state of research in the Deep Learning community**, and more specifically pointers as to where the most up-to-date information can be found.
- This includes **tracking big names** in the field, identifying **key scientific papers** that have changed the field, but also **newsletters, websites or magazines** that disclose information in layman terms.

# Skills needed for this course

- **Must-have CS:** Python, Numpy, Matplotlib.
- **Must-have Math:** Linear Algebra, multiple variables calculus, derivatives, optimization probability and statistics.
- **Good-to-have CS:** Machine Learning, Data Processing, Scipy, Sklearn.
- **Good-to-have Math:** Graph Theory, Game theory.
- **Need to revise?** Time to check your 10.014 CTD, 10.013 M&A, 10.018 MS&S, 10.022 MU, 10.020 DDW, and 50.007 ML!

# The way I teach things

## **An expert is not someone who**

- Undertook a 6h-long online course,
- Implements Neural Networks using basic layers (Linear, Conv, and that is it),
- Using a random framework he does not understand (or autoML!),
- and is happy to get an 80% accuracy on a simple dataset like MNIST.

## **An expert is someone who**

- Understands how Neural Network operate and the mathematical intuition behind typical (advanced) operations,
- Understands how the frameworks have been built and what they do behind the scene.
- Someone who knows how to stay up to date when it comes to AI.

# The way I teach things

- **One topic per week** (or so) and hopefully, we will cover all the important concepts and important directions of DL these days.
- **Lectures slides** uploaded the day before, and **supporting notebooks**.
- **Mathematical aspects** discussed in class (behind the scene of Neural Networks/Deep Learning).
- Made a choice: providing lots of code to demonstrate concepts, but **need you to play with them autonomously** to explore concepts!
- **Extra reading**, for curiosity (supporting papers, articles, etc.) and suggestions of pointers to follow to stay up to date.
- Suggestions for **continuing your learning**, after this course.

# Syllabus

- **Week 1:** Introduction to course, some ML jargon reminders, linear and polynomial regression, generalization, ridge regression and regularization, overfitting/underfitting, logistic regression, neurons objects and how they relate to biology, our first shallow neural network, gradient descent reminders, training and testing procedure.
- **Week 2:** Introduction to PyTorch framework, tensors and dataloaders, implementing a shallow neural network in PyTorch, backpropagation in PyTorch with AutoGrad, advanced optimizers, multi-label classification with shallow neural networks, moving from shallow to deep neural networks.



# Syllabus

- **Week 3:** Guided project and good practices for Deep Learning projects (train/test/dev, bias/variance, advanced regularization, dropout, normalizing inputs/outputs/layers, trainer functions, savers/loader functions for reproducibility and transfer learning).
- **Week 4:** The image data type, image processing techniques and typical computer vision operations, the convolution operation and layers, Convolutional Neural Networks, advanced CNNs and SotA. Preparing transition to the 50.035 Computer Vision course.

# Syllabus

- **Week 5:** Adversarial machine learning, attacking a Neural Network with basic gradient-based attacks, fundamental limits of Neural Networks, defense mechanisms and state-of-the-art of some advanced attacks techniques. MidTerm exam (based on W1-5).
- **Week 6:** Sequential data (times series, text, etc.), vanilla Recurrent Neural Networks, Gated Recurrent Units, Long-Short Term Memory cells, advanced RNN networks, mixing models for advanced architectures.

# Syllabus

- **Week 8:** The embedding problem, more advanced concepts on RNNs, introduction to Natural Language Processing (NLP) and Word Embeddings for NLP, brief state-of-the-art on NLP, attention and transformers architectures.  
Preparing transition to the 50.040 Natural Language Processing course.
- **Week 9:** Quick introduction to Graph Theory and typical graph datasets and problems, basics of Graph Convolutional Networks, brief state-of-the-art of advanced Graph Convolutional Networks.

# Syllabus

- **Week 10:** Generative Models, Autoencoders and Variational Autoencoders, Generative Adversarial Networks (GANs), Advanced concepts on Generative Adversarial Networks, Practice on GANs.
- **Week 11:** Brief introduction to reinforcement learning, and state-action-rewards systems, multi-armed bandit problem and the exploration/exploitation trade-off, Q-learning and Deep Q-Learning. Brief state-of-the-art discussion about further works in Reinforcement Learning.

# Syllabus

- **Week 12:** Explainability/Interpretability and open questions in research about Neural Networks. Deep belief models and diffusion models. What will be the next revolution in AI? (a word on ChatGPT, Dall-E, etc.). Closing and future directions for studying Deep Learning.
- **Week 13:** Recap. Project presentations and guest conferences (TBA).
- **Week 14:** Final exam (probably W1-13).

# Supporting Textbooks

Supporting textbooks, for your curiosity (not needed to understand this course).

- Michael A. Nielsen, “Neural networks and deep learning”, 2015.  
(<http://neuralnetworksanddeeplearning.com/>)
- Ian Goodfellow, Yoshua Bengio and Aaron Courville, “Deep learning”, 2016.  
(<https://www.deeplearningbook.org/>)
- Stevens et al., “Deep Learning with PyTorch”, 2020.  
(<https://www.manning.com/books/deep-learning-with-pytorch>)

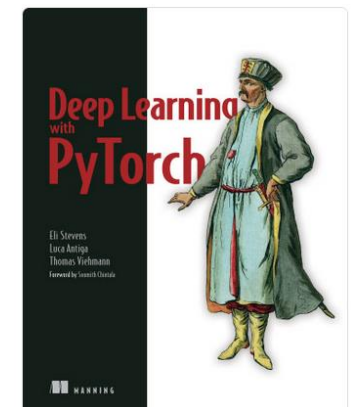
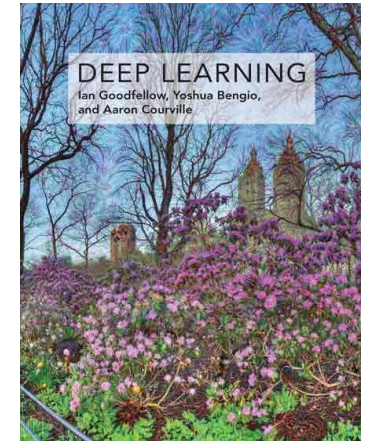
## Neural Networks and Deep Learning

*Neural Networks and Deep Learning* is a free online book. The book will teach you about:

- Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data
- Deep learning, a powerful set of techniques for learning in neural networks

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. This book will teach you many of the core concepts behind neural networks and deep learning.

For more details about the approach taken in the book, [see here](#). Or you can jump directly to [Chapter 1](#) and get started.



# Evaluation and grading

- **Homeworks (24%):** Given on Weeks 2, 5, 8, 10.  
Usually come in the form of a Jupyter Notebook, containing explanations, code snippets and questions.  
Submissions on eDimension, two weeks later or so, as a small PDF report containing code, figures and answers to questions.  
When time allows, debrief of homeworks in class.
- **MidTerm Exam (20%):** Given on Week 6, March 1<sup>st</sup> 2023.  
Theoretical, paper exam, notions of Week 1-5 to be tested.  
More details about venue and exam details to be announced closer to the exam date.

# Evaluation and grading

- **Final Exam (20%):** Given on Week 14, April 26<sup>th</sup> 2023.  
Theoretical, paper exam, notions of Week 1-13 to be tested.  
More details about venue and exam details to be announced.
- **Project (29%):** Groups of 2-3 students. Submission for project (code, report, presentation) expected on Week 13. Problem statement to be freely decided by students, as long as it matches a list of given requirements. We will have a guided demo project on Week 3, more details about the project will be given on Week 5.
- **Participation (5%):** to my discretion.
- **Student Feedback Survey (2%):** the usual.



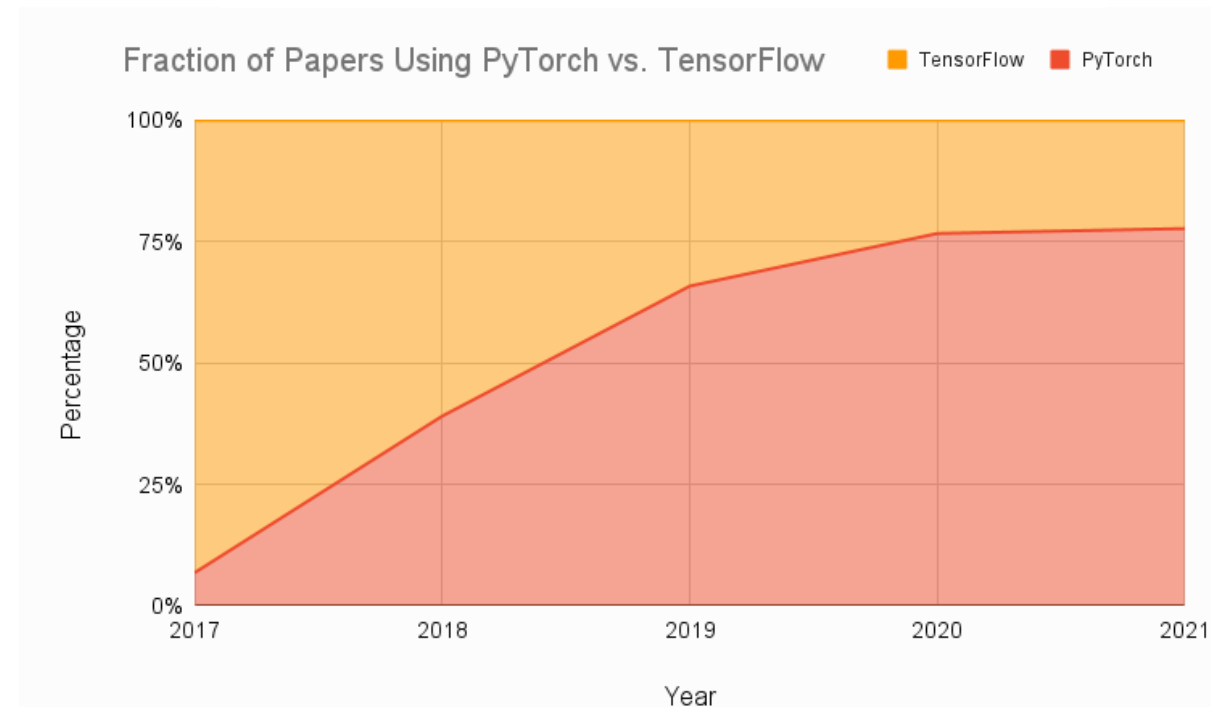
# Technical pre-requisites

- Install **Python 3**, if you have not done so already (this course does not cover C++/Java).
- **Libraries needed (maybe more based on projects/homeworks):** numpy, matplotlib, scipy, sklearn, networkx, pillow, gym.
- **Jupyter notebooks** for demos of code, along with the slides.
- In doubt, you can always use **Google Colab** to run the codes.



# Technical pre-requisites

- Framework of choice will be **PyTorch!** (not Tensorflow, not Keras, not MXNet, etc.)
- Increasing popularity and preferred to Google's Tensorflow these days for many reasons.
- Learn more, if curious:  
<https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/>



# Installing PyTorch and CUDA

- Install PyTorch, by getting the right version from <https://pytorch.org/get-started/locally/>

PyTorch Build	Stable (1.13.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.6	CUDA 11.7	ROCm 5.2	CPU
Run this Command:	<pre>pip3 install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu117</pre>			

# Installing PyTorch and CUDA

Compromised PyTorch-nightly dependency chain between December 25th and December 30th, 2022.



If you installed PyTorch-nightly on Linux via pip between December 25, 2022 and December 30, 2022, please uninstall it and torchtriton immediately, and use the latest nightly binaries (newer than Dec 30th 2022).

```
$ pip3 uninstall -y torch torchvision torchaudio torchtriton  
$ pip3 cache purge
```

PyTorch-nightly Linux packages installed via pip during that time installed a dependency, torchtriton, which was compromised on the Python Package Index (PyPI) code repository and ran a malicious binary. This is what is known as a supply chain attack and directly affects dependencies for packages that are hosted on public package indices.

**NOTE:** Users of the PyTorch **stable** packages **are not** affected by this issue.\*\*

# Installing PyTorch and CUDA

- Check if your GPU is in the list of acceptable GPUs.  
<https://developer.nvidia.com/cuda-gpus>
- If so, install CUDA (check version number matches PyTorch install!)  
<https://developer.nvidia.com/cuda-downloads>

## CUDA Toolkit 11.8 Downloads

[Home](#)

### Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the [CUDA EULA](#).

Operating System

Linux

Windows

Architecture

x86\_64

Version

10

11

Server 2016

Server 2019

Server 2022

Installer Type

exe (local)

exe (network)

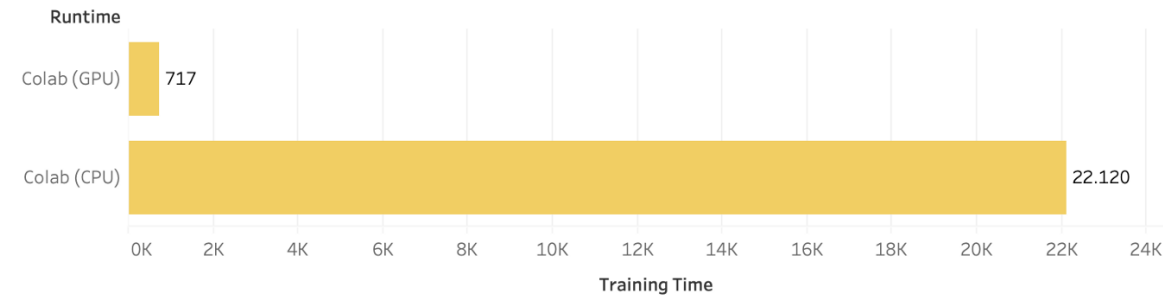
# Installing PyTorch and CUDA

- GPU not in the list of CUDA-enabled GPUs?

**Most notebooks can still run on CPU (but they might take significantly longer).**

- Always the option of using Google Colab, or create an education account on AWS.

CPU vs GPU Training Time Comparison in Seconds



# Checking your PyTorch and CUDA install

- **Hello World for PyTorch:** to check you have PyTorch installed correctly. The code below should run and display a tensor.

```
import torch
x = torch.rand(5, 3)
print(x)
```

```
tensor([[0.3380, 0.3845, 0.3217],
        [0.8337, 0.9050, 0.2650],
        [0.2979, 0.7141, 0.9069],
        [0.1449, 0.1132, 0.1375],
        [0.4675, 0.3947, 0.1426]])
```

- **Hello World for CUDA:** to check you have correctly installed CUDA on top of PyTorch. The code below should print *True*, as the output of *torch.cuda.is\_available()*

```
import torch
torch.cuda.is_available()
```

# A quick word about PyTorch 2.0

**PyTorch 2.0** has been announced end of 2022 and will have several new features, including

- **`torch.compile()`**, which allows for faster training,
- and the ability to move parts of the PyTorch framework from C++ back into Python.

It will also be cloud-agnostic, open-source.





# A quick word about PyTorch 2.0

PyTorch 2.0 is currently still in its experimental phase. **Stable PyTorch 2.0 release date is slotted for March 2023.**

*(Some nightlies releases are available, but I would advise against using them for now!*

*Allow me to explore it a bit more first and let us discuss it again in Week 13!)*



# About this week (Week 1)

1. What are the **typical concepts of Machine Learning** to be used as a starting point for this course?
2. What are the **different families of problems** in Deep Learning?
3. What is the **typical structure of a Deep Learning problem**?
4. What is **linear regression** and how to implement it?
5. What is the **gradient descent algorithm** and how is it used to **train Machine Learning models**?
6. What is **polynomial regression** and how to implement it?
7. What is **regularization** and how to implement it in **Ridge regression**?

# About this week (Week 1)

8. What is **overfitting** and why is it bad?
9. What is **underfitting** and why is it bad?
10. What is **generalization** and how to evaluate it?
11. What is a **train-test split** and why is it related to **generalization**?
12. What is a **sigmoid** function? What is a **logistic** function?
13. How to perform **binary classification** using a **logistic regressor** and how is it related to linear regression?

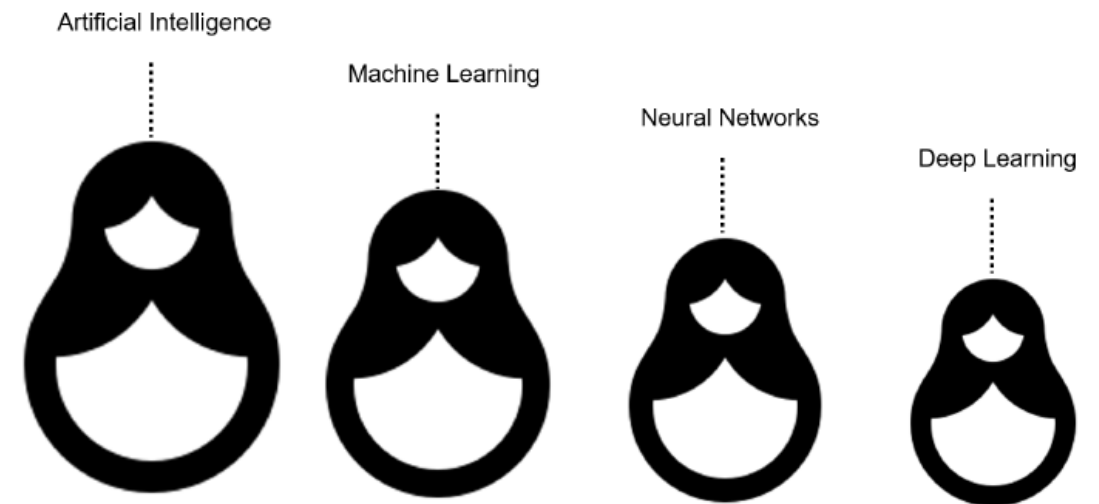
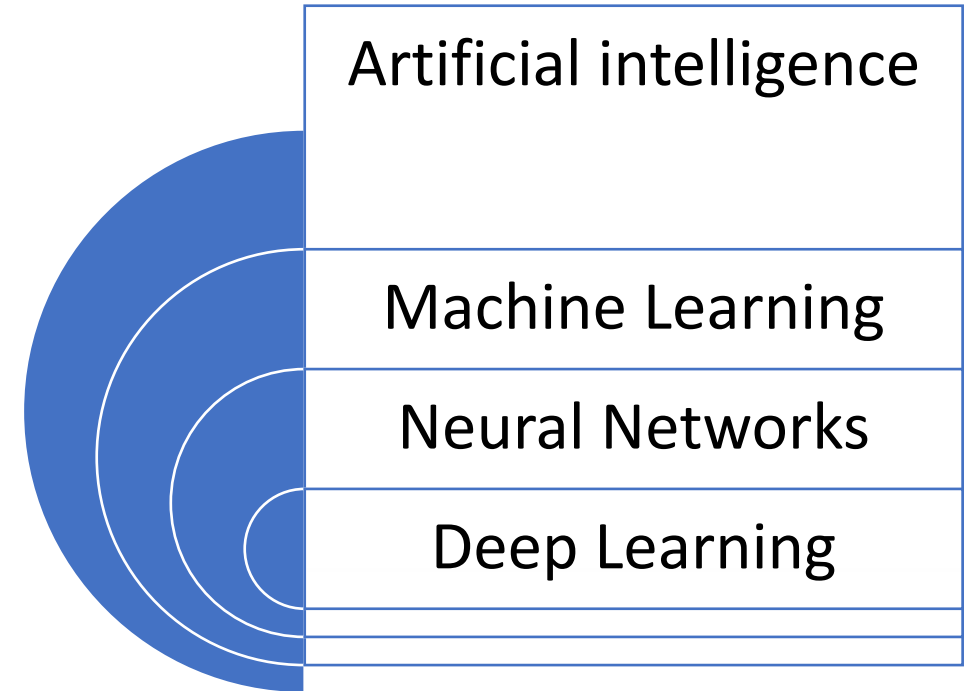
# About this week (Week 1)

14. What are **Neural Networks** and how do they relate to the **biology of a human brain**?
15. What is a **Neuron** in a Neural Network and how does it relate to linear/logistic regression?
16. What is the **difference** between a **shallow** and a **deep neural network**?
17. How to **implement a shallow Neural Network** manually and define a **forward propagation** method for it?
18. How to **train a shallow Neural Network** using **backpropagation**?  
How to define **backward propagation** and **trainer** functions?

# What is AI/ML/NN/DL?

## Definition (**Artificial Intelligence**):

“In Computer Science, **Artificial Intelligence (AI)** refers to the theory and development of computer systems capable to **perform cognitive tasks** that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.”



# What is AI/ML/NN/DL?

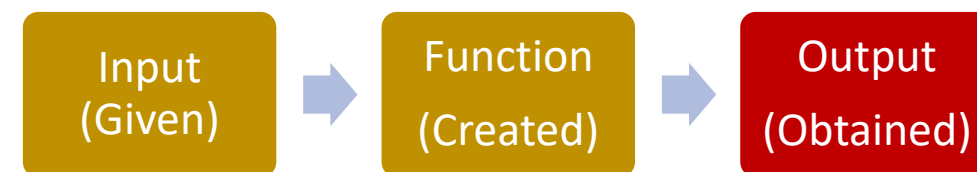
## Definition (**Machine Learning**):

In Computer Science, **Machine Learning (ML)** refers to the field of study that describes **techniques** and **algorithms** that give computers the **ability to learn without being explicitly programmed**.

Some implementations of machine learning may rely on data and neural networks.

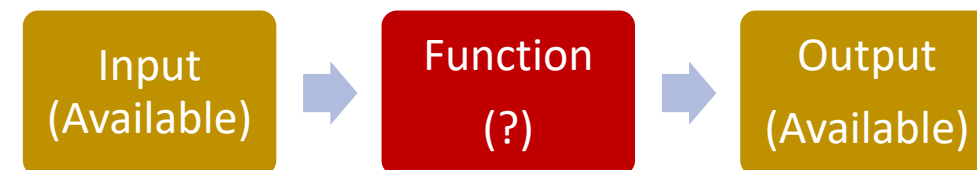
- Arthur Samuel (1959).

### Conventional programming



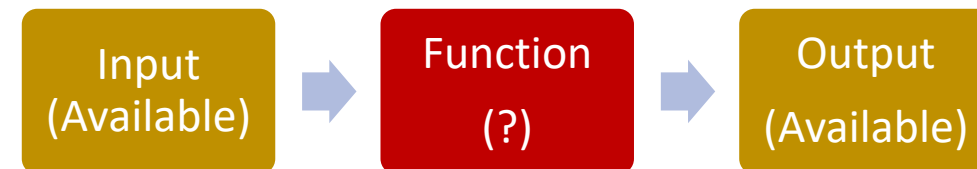
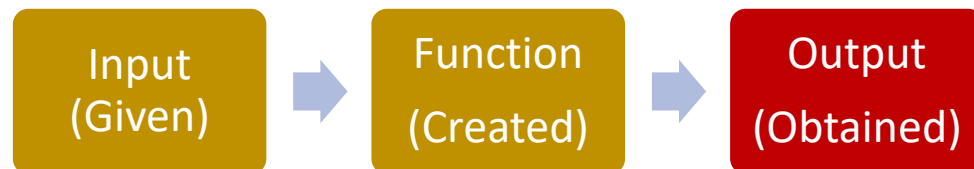
VS.

### Machine Learning



# What is AI/ML/NN/DL?

- What we have done in programming so far was to design functions,
  - which would do **specific operations**,
  - and return **outputs**,
  - for any **input** we could give it.
- But sometimes, we can encounter problems where
  - we can easily find **inputs** and **expected** outputs,
  - but the **function** to be coded is **not simple** to figure out.
  - **E.g., what animal is in the picture?**



# What is AI/ML/NN/DL?

E.g., what animal is in the picture?

Typical problem in **Computer Vision**, called **Image Recognition**.

Input  $x$  (available)



Function  $f$   
(?)



Output  $y = f(x)$   
(available)



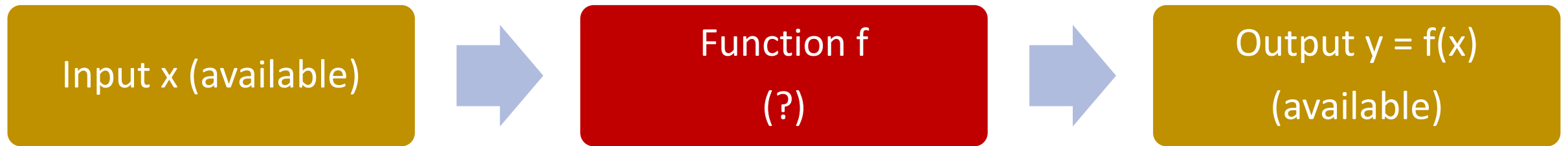
It's a cat!



# What is AI/ML/NN/DL?

E.g., what animal is in the picture?

Typical problem in **Computer Vision**, called **Image Recognition**.



Very easy for a human...  
**But, how would we do it with a computer?**

It's a cat!

# What is AI/ML/NN/DL?

Other scenarios of difficult functions have to do with tasks where there is **no easy closed-form expression connecting inputs to outputs.**

- **E.g., what is a good selling price for my apartment?**
- Guessing the selling price of an apartment based on its parameters (size, location, etc.) and previous sales.

**s\$1,680,000**

Negotiable

3  3  1184 sqft s\$ 1,418.92 psf

## Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016

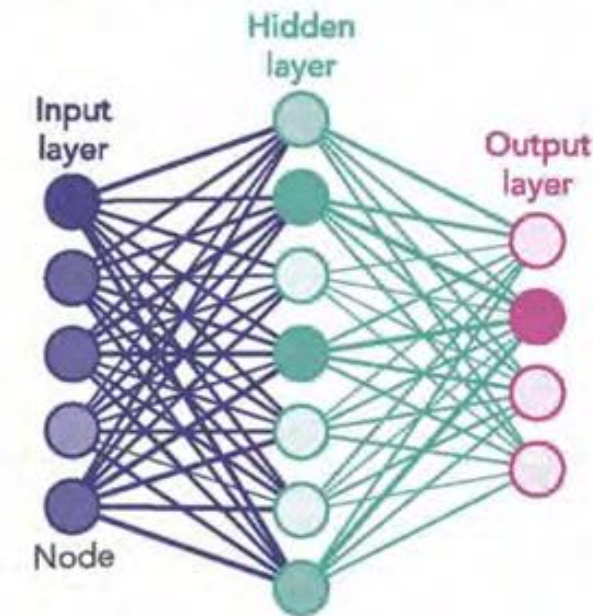
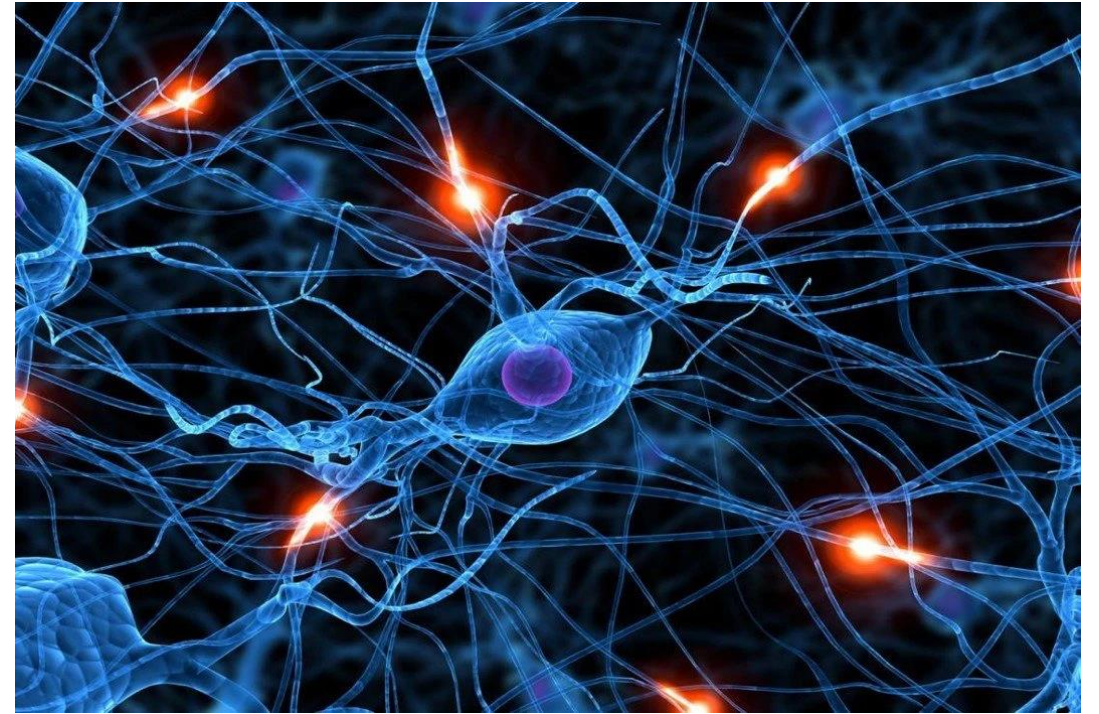


# What is AI/ML/NN/DL?

## Definition (**Neural Networks**):

**Neural Networks (NNs)** are computing systems inspired by the biological neural networks that constitute animal brains.

NNs are based on a **collection of connected units or nodes called artificial neurons**, which loosely model the neurons in a biological brain.



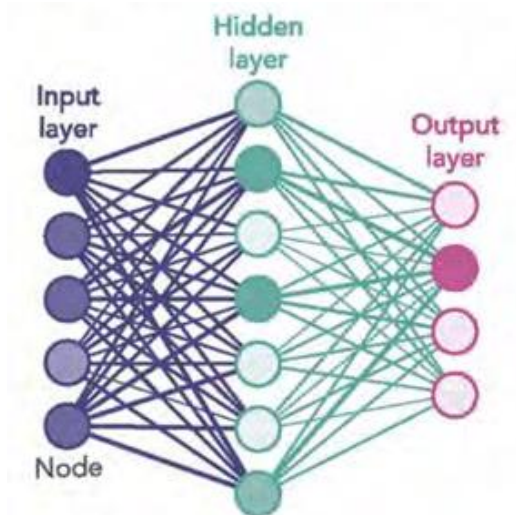
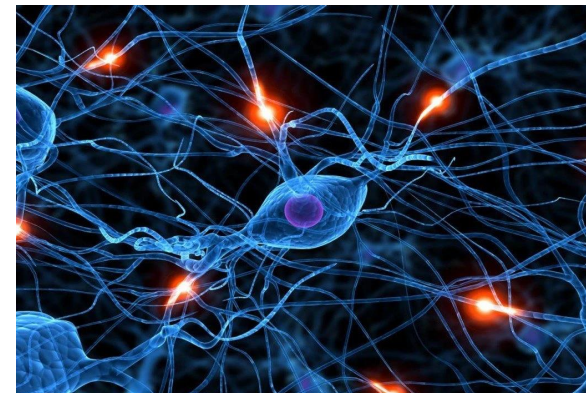
# What is AI/ML/NN/DL?

## Definition (**Neural Networks**):

**Neural Networks (NNs)** are computing systems inspired by the biological neural networks that constitute animal brains.

NNs are based on a **collection of connected units or nodes called artificial neurons**, which loosely model the neurons in a biological brain.

Each connection, like the synapses in a biological brain, can transmit a signal to other neurons, therefore **processing any information** given as inputs to produce a final signal as output.



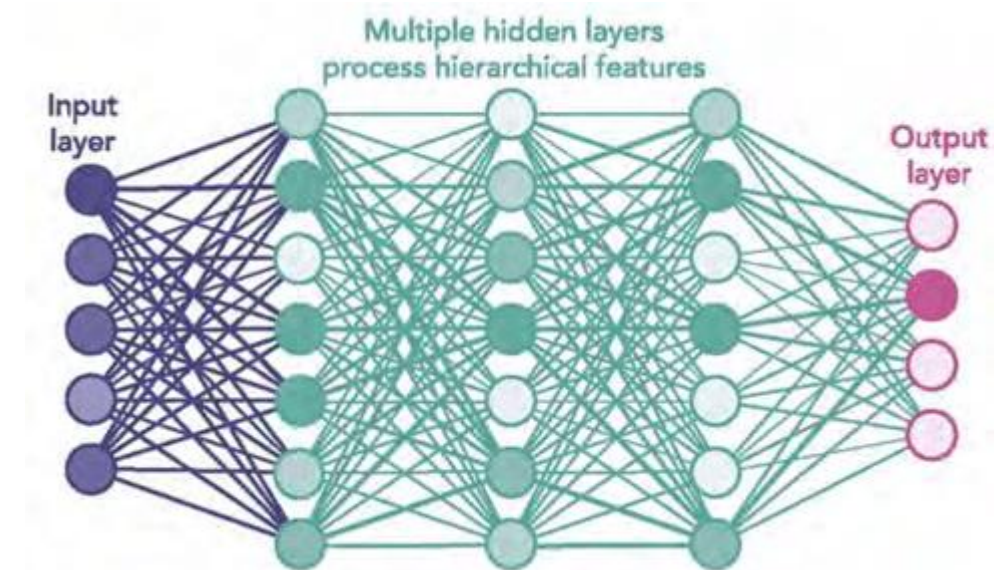
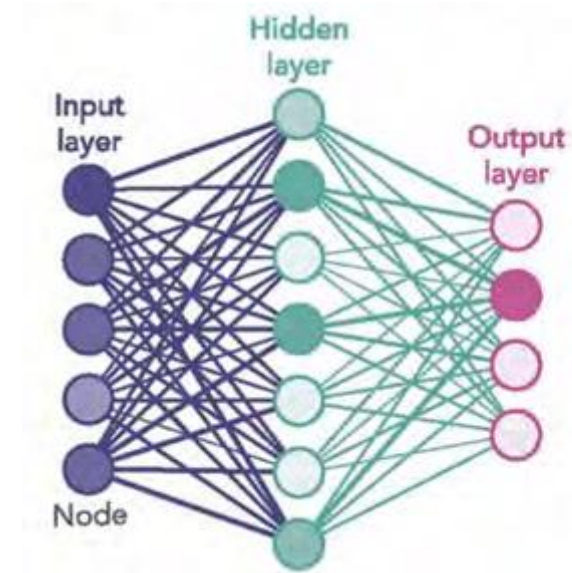


# What is AI/ML/NN/DL?

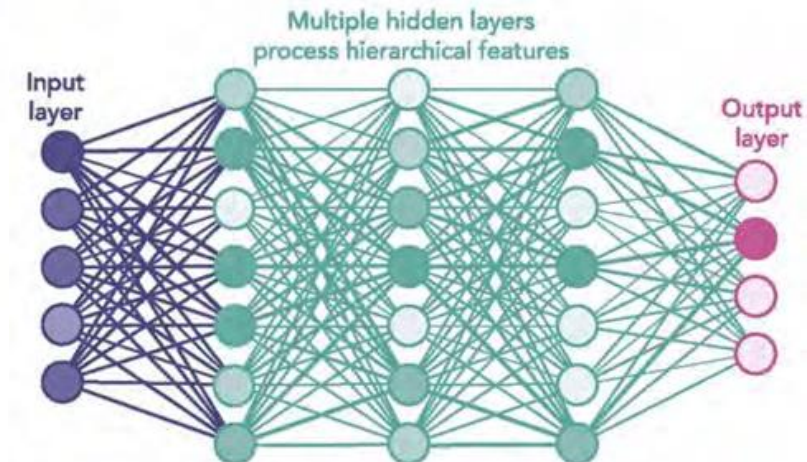
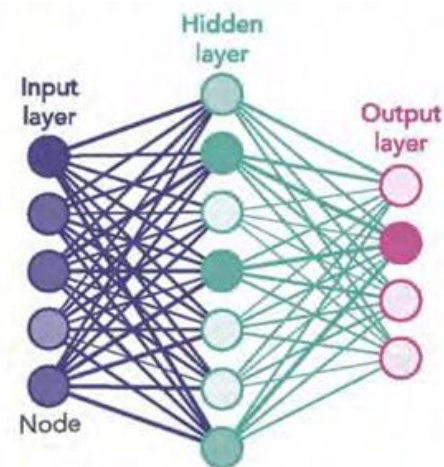
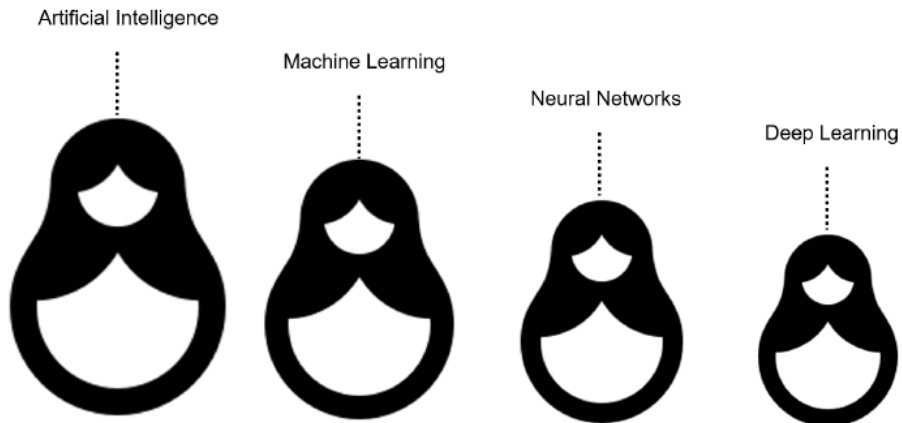
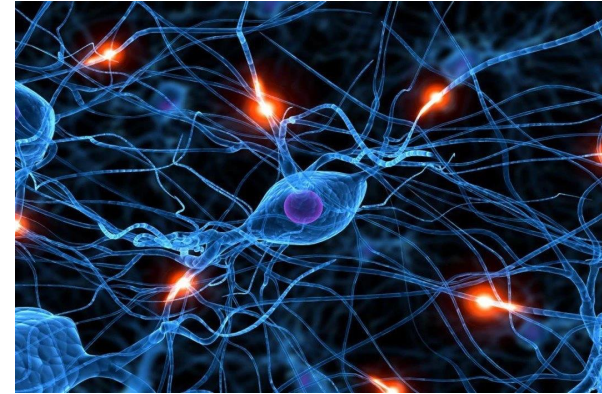
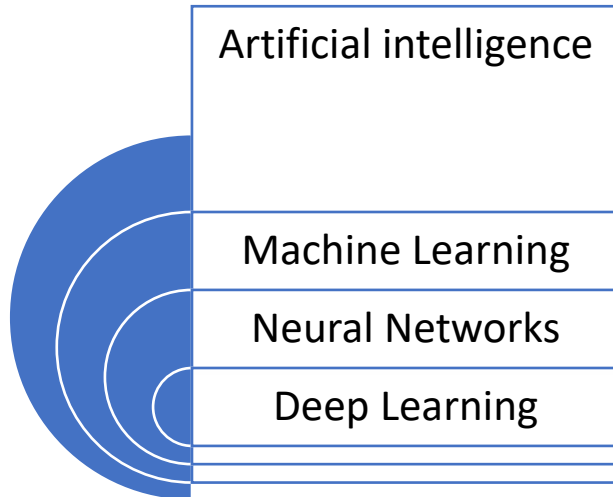
## Definition (**Deep Learning**):

**Deep Learning (DL)** is a subfield of machine learning, and deep neural networks make up the backbone of deep learning algorithms.

The **number of node layers**, or **depth**, of neural networks is what distinguishes a shallow neural network from a deep neural network, which must have more than three.



# What is AI/ML/NN/DL?



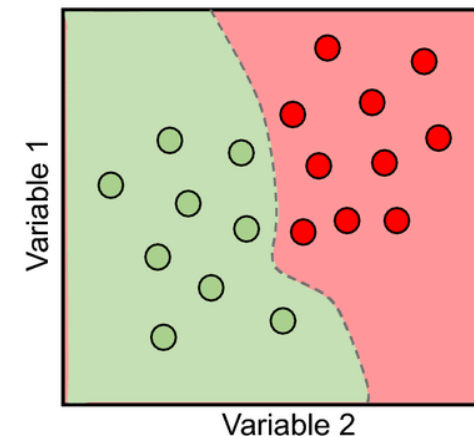
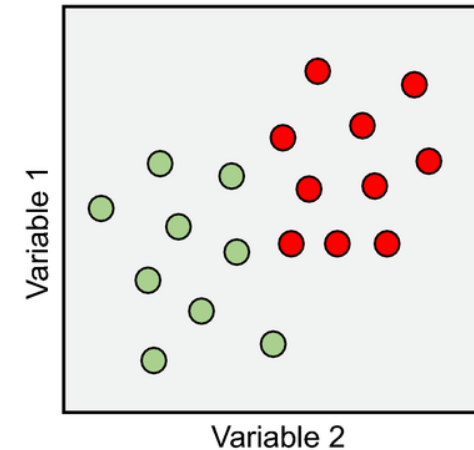
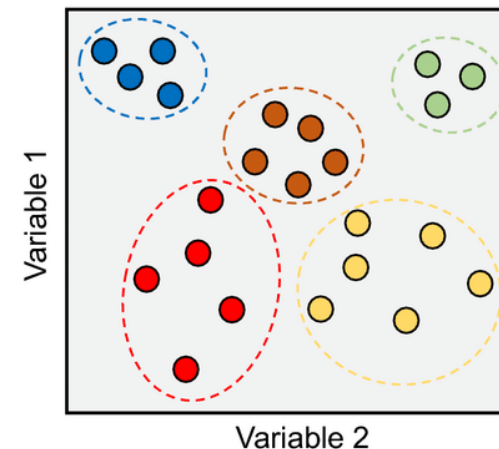
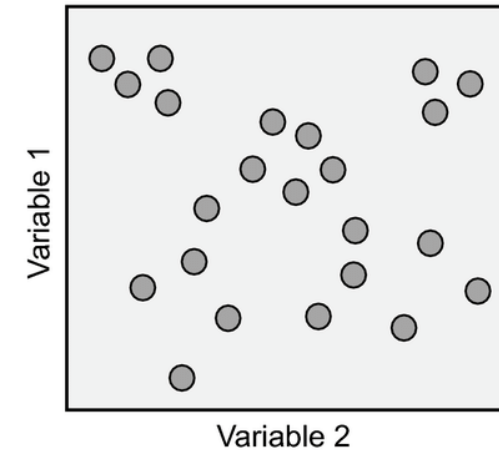
# Supervised vs Unsupervised Learning

## Definition (Supervised vs. Unsupervised Learning):

**Supervised** and **Unsupervised Learning** are the two techniques of machine learning.

The main difference is **the need for labelled training data**:

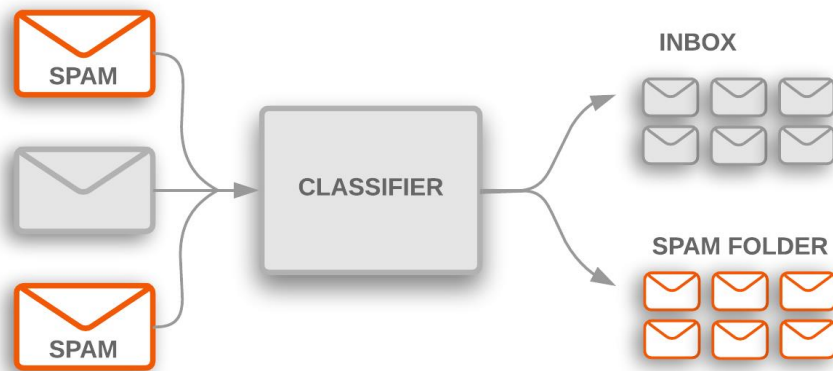
- Supervised machine learning relies on labelled input and output data to learn and make predictions,
- while **unsupervised learning does not require labelled data**.





# Supervised Learning Examples

- Examples of **supervised learning**: spam detection, text classification, predicting the stock market, etc.





# Supervised Learning Examples

- Examples of **supervised learning**: spam detection, text classification, predicting the stock market, etc.

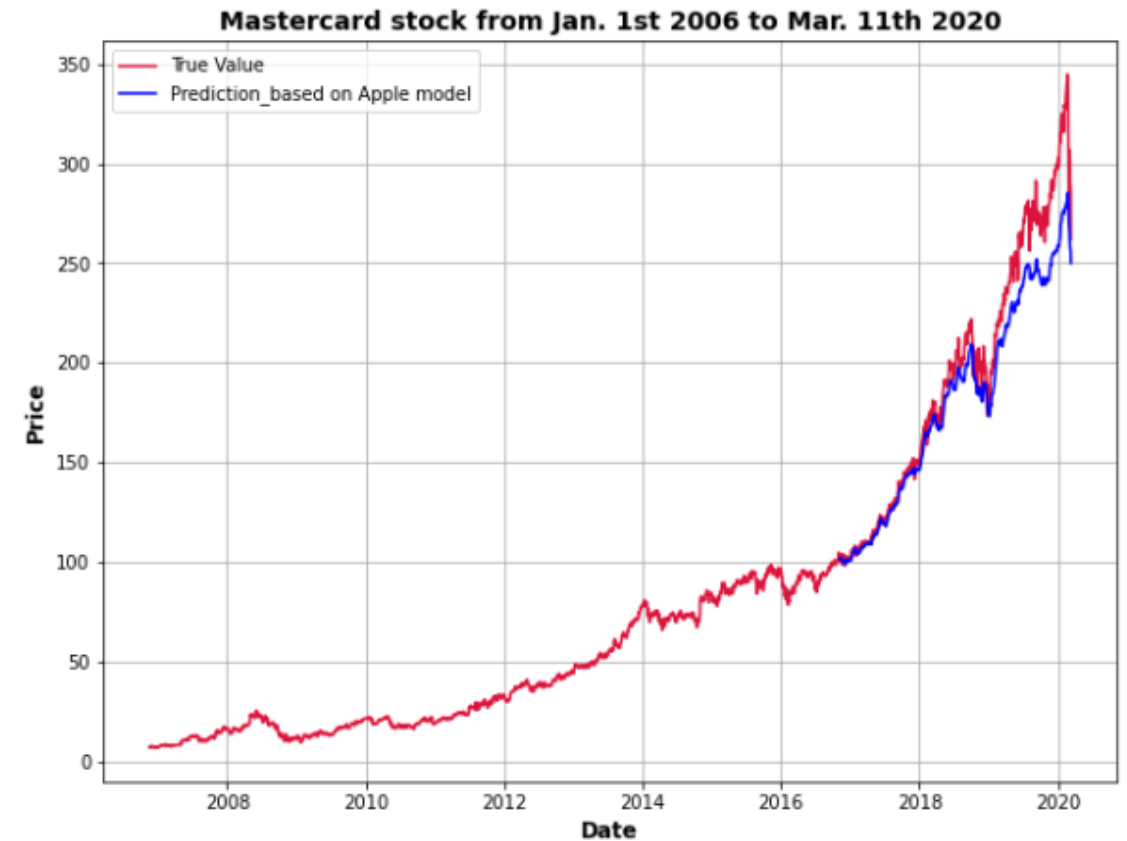
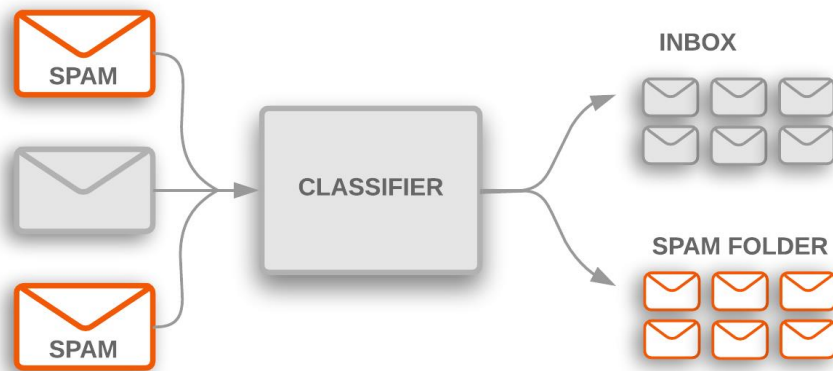


Figure 7. MAST stock price predictions using LSTM trained on AAPL

# Regression vs. Classification


## Definition (**Regression**):

Regression models are used to identify the relationships between the input and output variables.

Regression algorithms are used to **predict continuous values** such as price, salary, age, etc.

In regression, the outputs are often **continuous numerical values**.

**s\$1,680,000** Negotiable

3  3  1184 sqft s\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016



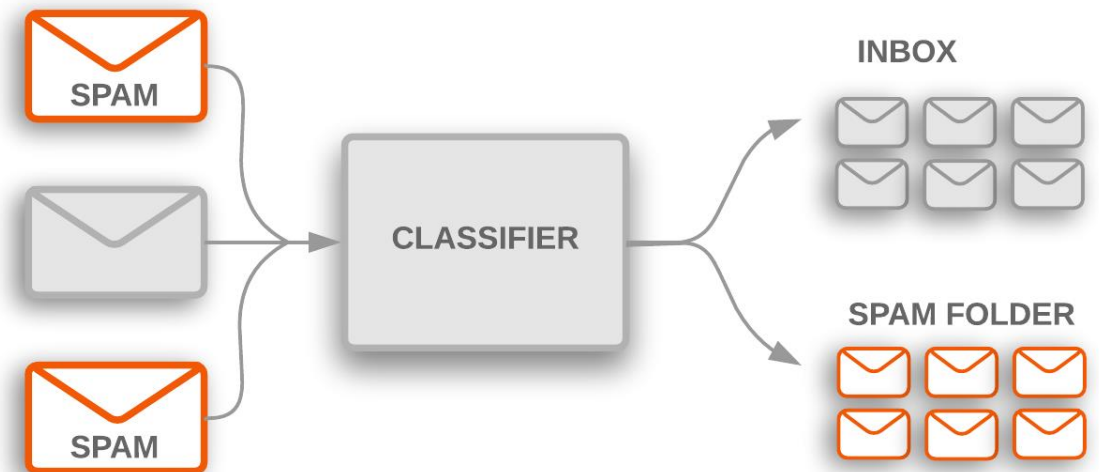
# Regression vs. Classification (Supervised)

## Definition (**Classification**):

Classification models are used to divide the samples in the dataset into different **classes**.

Classification algorithms are then used to predict/classify **discrete values**, such as Male or Female, True or False, Spam or Not Spam, etc.

In Classification, the outputs are **discrete** or **categorical values**.



# Supervised Learning Examples

**Example:** predicting the market.

**Question:** is it a **regression** or a **classification** task?

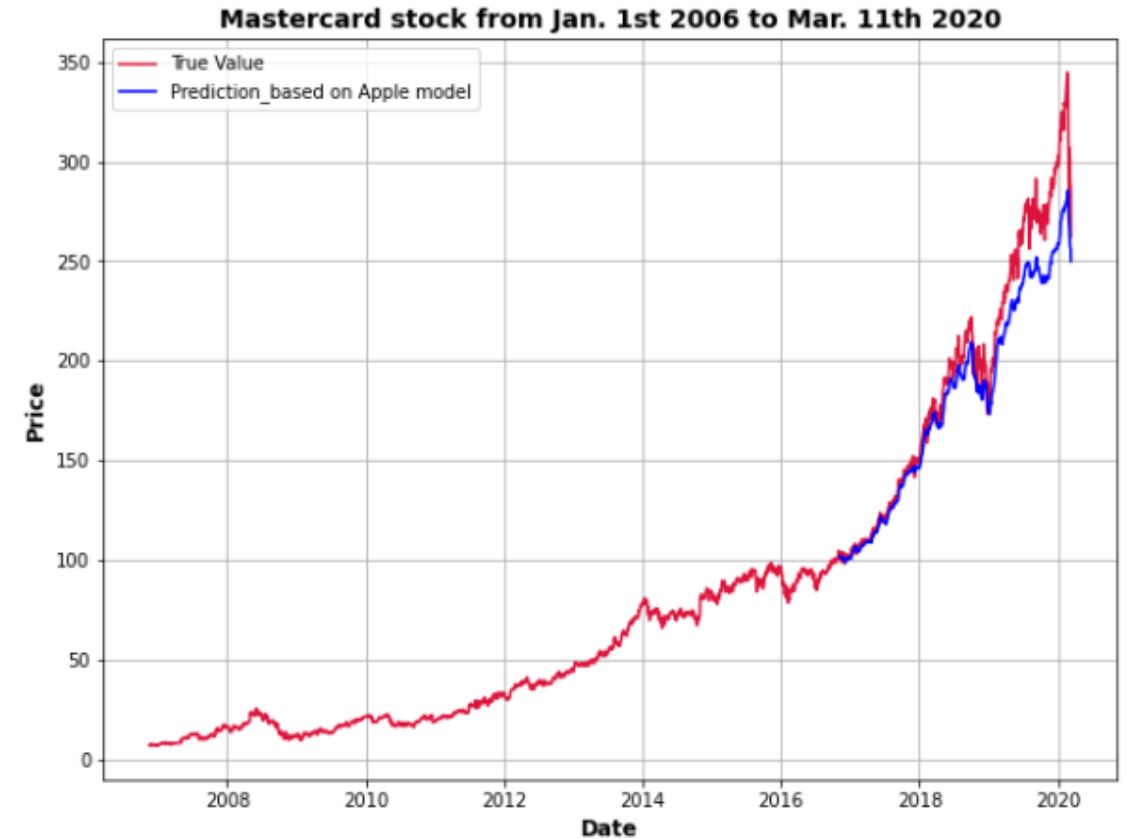


Figure 7. MAST stock price predictions using LSTM trained on AAPL

# Supervised Learning Examples

**Example:** predicting the market.

**Question:** is it a **regression** or a **classification** task?

**Depends.**

If the output we are **predicting** is **the value of the stock in the future**, then probably **regression**.



Figure 7. MAST stock price predictions using LSTM trained on AAPL

# Supervised Learning Examples

**Example:** predicting the market.

**Question:** is it a **regression** or a **classification** task?

**Depends.**

If the output we are **predicting is the value of the stock in the future**, then probably **regression**.

If the plan is to **predict whether we should buy, sell or wait**, then probably **classification**.

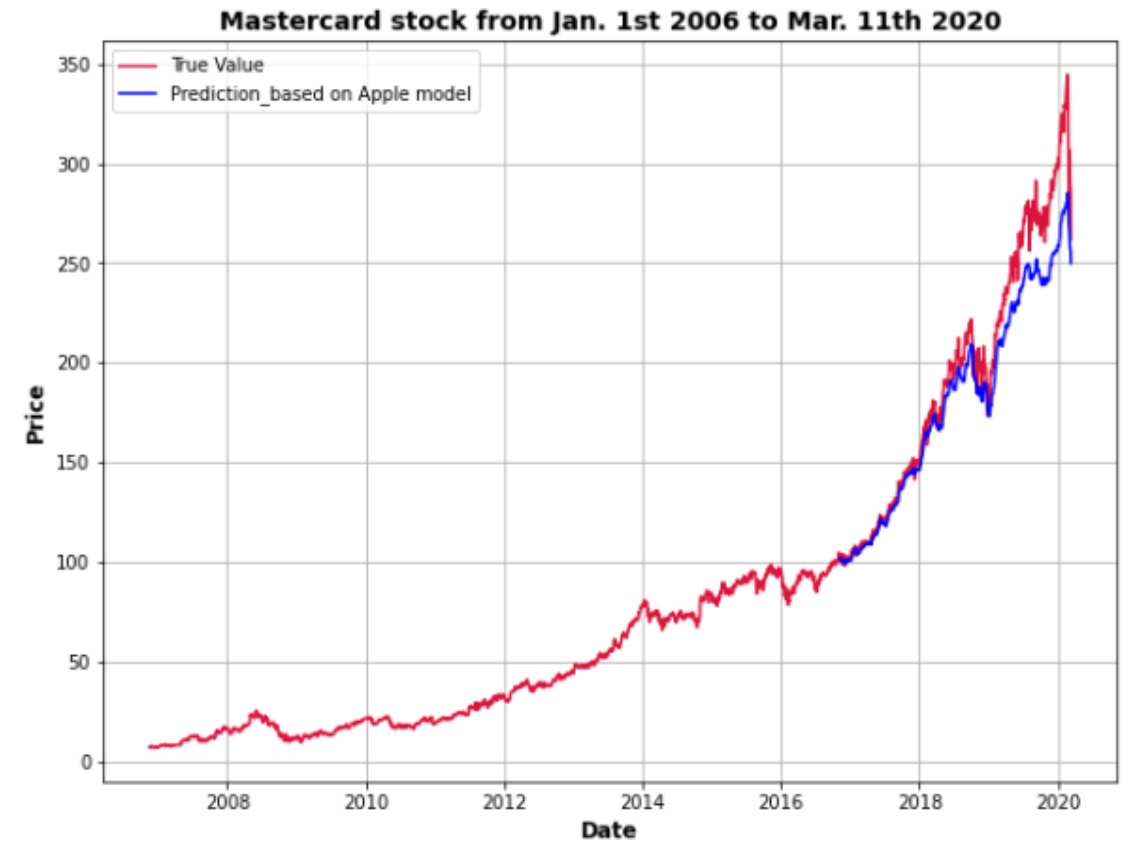
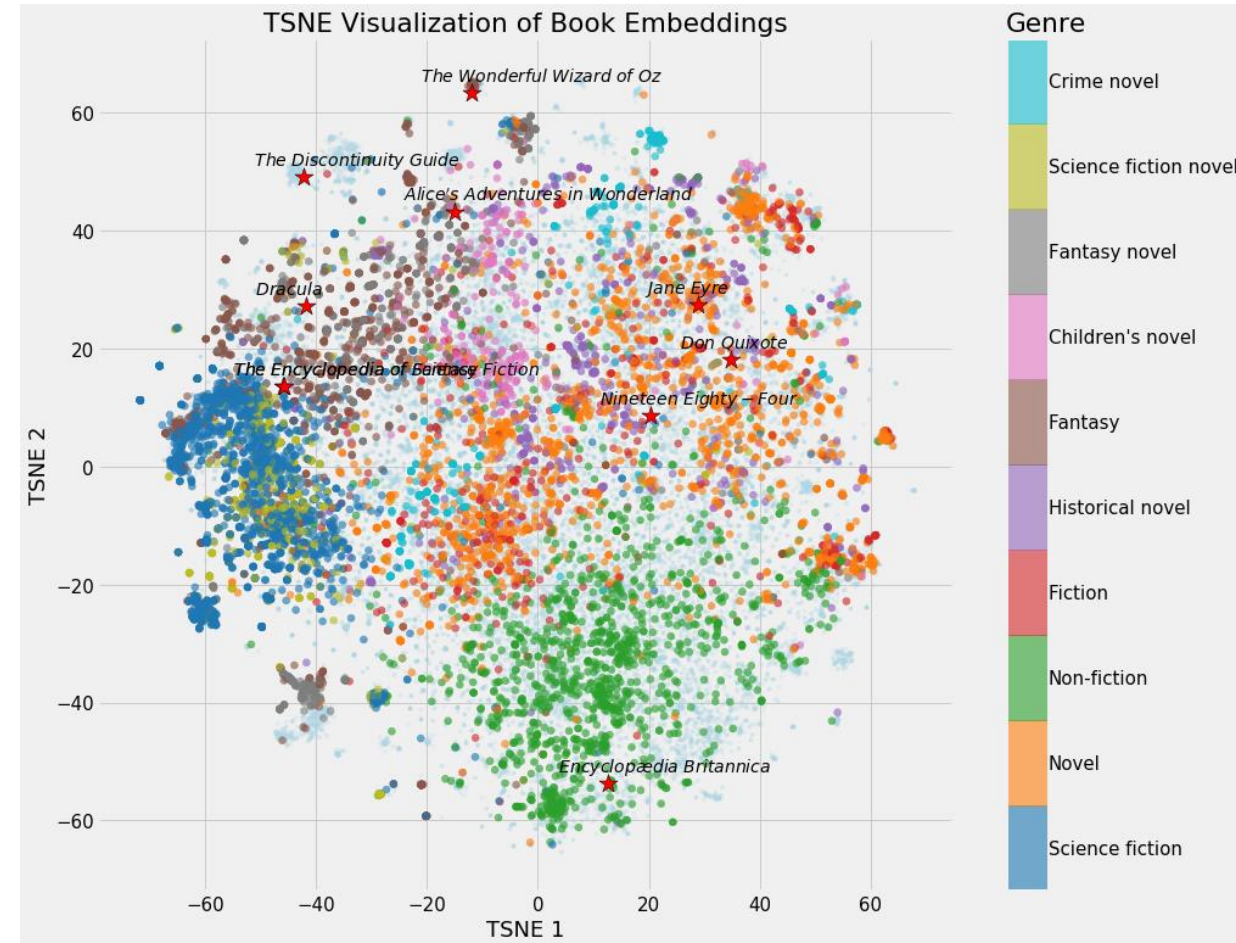


Figure 7. MAST stock price predictions using LSTM trained on AAPL

# Unsupervised Learning Examples

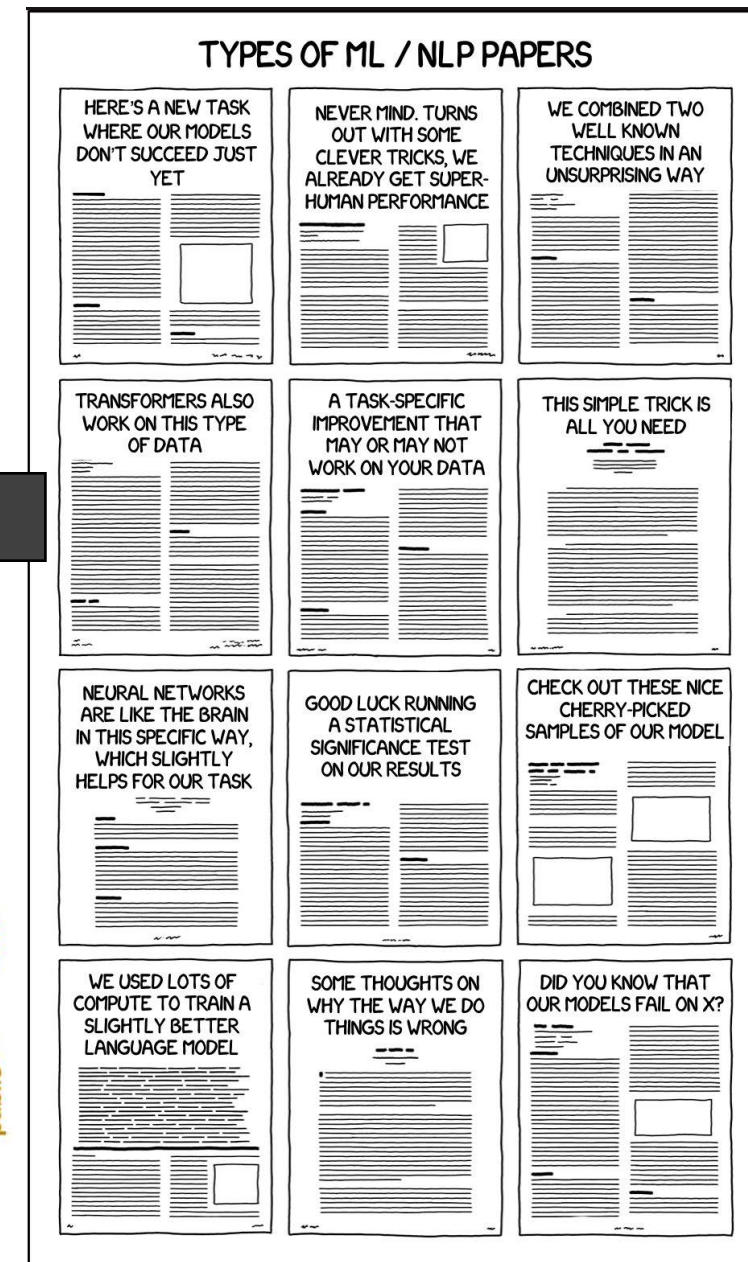
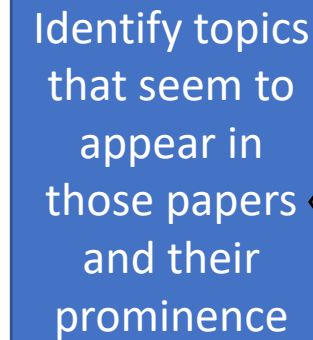
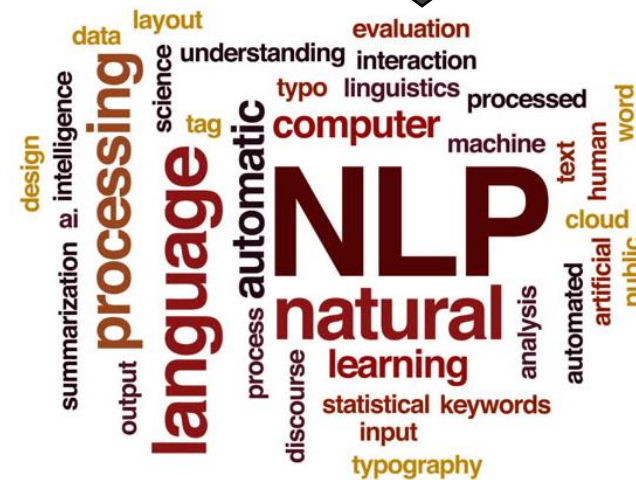
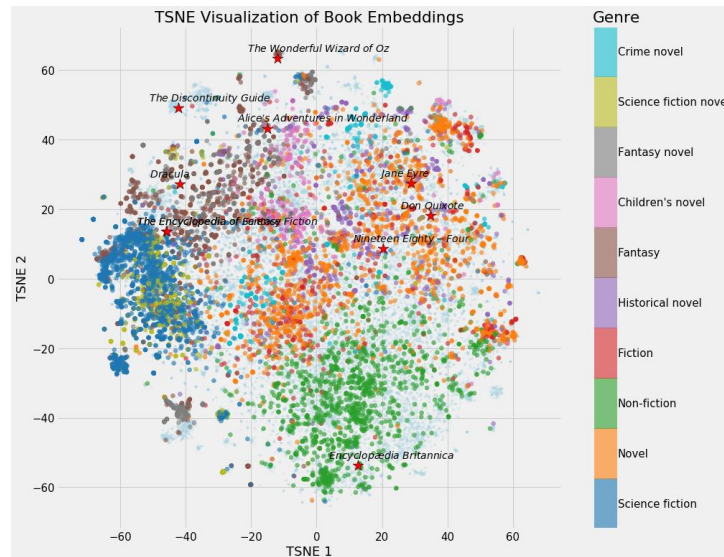
- Examples of **unsupervised learning**: clustering customers based on their behaviors, segmenting images, and identifying topics in a document.





- Examples of **unsupervised learning**: clustering customers based on their behaviors, segmenting images, and identifying topics in a document.

- Examples of **unsupervised learning**: clustering customers based on their behaviors, segmenting images, and identifying topics in a document.





# Clustering vs. Association (Unsupervised)

## Definition (**Clustering vs. Association**):

Unsupervised learning can be used for two types of problems: **Clustering** and **Association**.

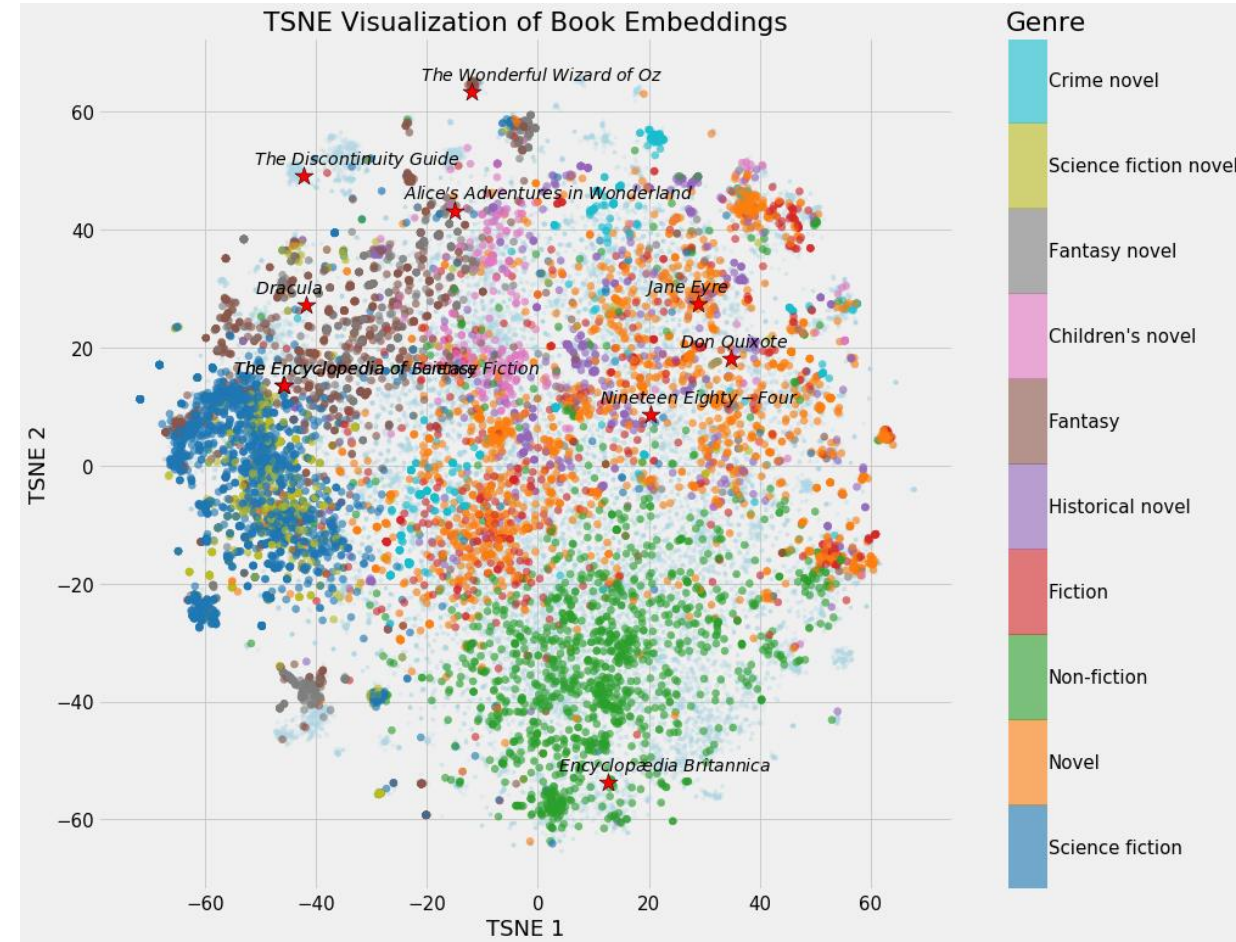
The main difference is that

- **clustering** groups data based on similarity,
- while **association** looks for relationships between variables.

# Clustering vs. Association (Unsupervised)

## Examples of clustering:

- Group customers into clusters based on the similarities in their purchasing behaviours, such as their spending habits or types of products they like.
- Group the catalogue of movies available on Netflix and identify what movies have in common, so you can later recommend a movie to a user based on what he/she has seen before.



# Clustering vs. Association (Unsupervised)

## Example of association:

- Finding the relationship between customer buying behaviors and customer satisfaction.
- Association can be used to understand how changes in customer behavior impacts customer satisfaction, such as if buying a certain product increases customer satisfaction (or the opposite).

Customer Satisfaction Survey on McDVoice.com

Español

Welcome to the McDonald's Customer Satisfaction Survey on McDVoice.com.

[Load Accessibility Friendly Version](#)

We value your candid feedback and appreciate you taking the time to complete our survey.

① To begin, please enter the 26-digit Survey Code located in the middle of your receipt.

-  -  -  -  -

**Start**

Survey code:  
99999-01010-61515-20238-00032-2  
2111 McDonald's Drive  
Oak Brook  
IL  
60513  
!!! THANK YOU !!!  
TEL# 312-555-5555 Store# 88710

KS# 1 Jun.15'15 (Mon) 20:23  
MFY SIDE 1 KVS Order 01  
QTY ITEM TOTAL  
1 Egg McMuffin 2.99  
Subtotal 2.99



# Deep Learning and Supervised/Unsupervised

While it is possible to use Deep Learning models and Neural Networks to perform both Supervised and Unsupervised tasks...

The vast majority of DL/NN tasks fall in the category of supervised learning.

(And we will therefore mostly focus on that)

# Elements of a ML problem

## Definition (**Elements of a Machine Learning problem**):

Machine learning problems should be well-defined, i.e. the following elements must be clearly identified:

1. **Task (T)** at hand,
2. **Inputs (I)** and **Outputs (O)**,
3. **Model (M)** definition and its **trainable parameters (P)**,
4. **Loss (L)** function to measure the performance on said task and dataset.

**s\$1,680,000**

Negotiable

3  3  1184 sqft s\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016



# Elements of a ML problem

In the example on the right...

## 1. Task (T) at hand:

- Predict price of an apartment based on the apartment features.
- Using a dataset consisting of previously seen appartments to learn the selling prices of apartments.
- It is therefore a supervised regression task.

**S\$1,680,000**

Negotiable

3  3  1184 sqft S\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016





# Elements of a ML problem


In the example on the right...

## 2. Inputs (I) and Outputs (O):

- Inputs will be a list of apartment features (e.g. number of bathrooms, number of bedrooms, surface size, etc.).
- Output will be the selling price in SGD or surface price in SGD per sqft.

**S\$1,680,000**

Negotiable

3  3  1184 sqft S\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016



# Elements of a ML problem



In the example on the right...

## 3. Model (M) definition and its trainable parameters (P):

- Could be a linear or polynomial regression model (to be discussed this week). Or a neural network of some sort. Or even something else (check your ML course!)
- ## 4. Loss (L) function to measure the performance on said task and dataset:
- To be discussed later, as this might depend on the model used.

**S\$1,680,000**

Negotiable

3  3  1184 sqft S\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016





# Our first toy example

**Our first toy example:** simplified version of the apartment price prediction.

## 1. Task (T) at hand:

- Predict price of an apartment based on the apartment features.
- We will generate a “fake” dataset consisting of apartments with features and selling prices.
- Supervised regression task.

s\$1,680,000

Negotiable

3  3  1184 sqft s\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016



# Our first toy example

**Our first toy example:** simplified version of the apartment price prediction.

## 2. Inputs (I) and Outputs (O):

- Inputs will be a list of apartment features, which here will only consist of the surface of the apartment, in square meters (sqm).
- Output will be the selling price in millions of SGD (mSGD).

s\$1,680,000

Negotiable

3 🏠 3 🚿 1184 sqft s\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016



# Our first toy example (mock dataset)

**Have a look at Week 1, Notebook 1.**

We will create a dataset, which uses surfaces as inputs, with values drawn randomly between 40 and 150 sqm.

We will assume that the price is randomly generated, using a uniform distribution, with

- average  $100000 + 14373$  times the surface in sqm,
- and a "variance" of  $\pm 10\%$  around the average value.

# Our first toy example (mock dataset)

```
1 # Two random generator functions to generate a mock dataset.
2 # 1. Surfaces randomly generated as uniform between min_surf and max_surf
3 def surface(min_surf, max_surf):
4     return round(np.random.uniform(min_surf, max_surf), 2)
```

```
1 # 2. Price is 100000 + 14373 times the surface in sqm, +/- 10%
2 # (randomly shifted to give the dataset some diversity).
3 def price(surface):
4     # Note: this will return the price in millions of SGD.
5     return round((100000 + 14373*surface)*(1 + np.random.uniform(-0.1, 0.1)))/1000000
```

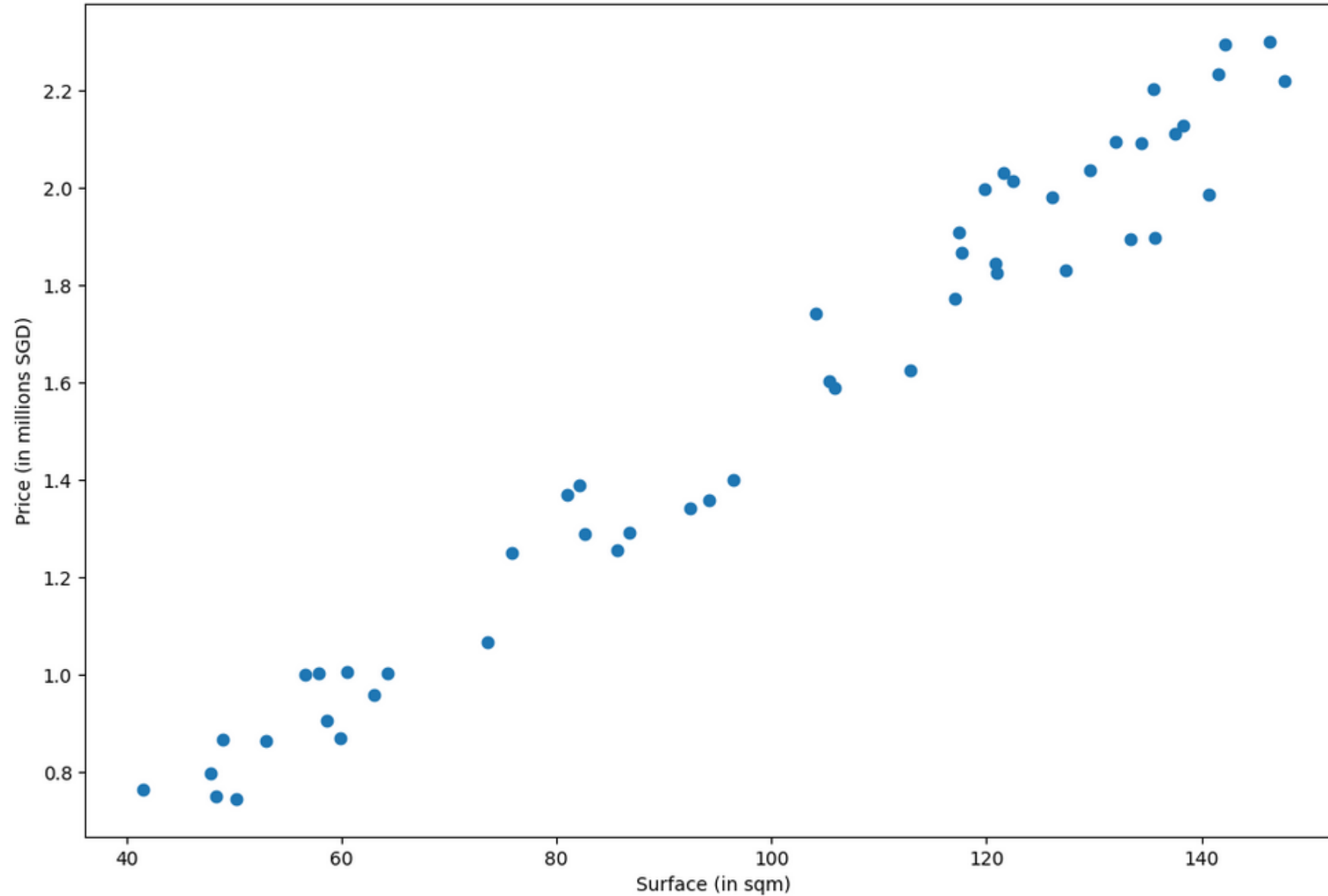
```
1 # 3. Generate dataset function
2 def generate_datasets(n_points, min_surf, max_surf):
3     x = np.array([surface(min_surf, max_surf) for _ in range(n_points)])
4     y = np.array([price(i) for i in x])
5     return x, y
```

# Our first toy example (mock dataset)

```
1 # 4. Dataset generation (n_points points will be generated).
2 # Surfaces randomly generated as uniform between 40sqm and 150sqm.
3 # We will use a seed for reproducibility.
4 min_surf = 40
5 max_surf = 150
6 np.random.seed(27)
7 n_points = 50
8 inputs, outputs = generate_datasets(n_points, min_surf, max_surf)
9 print(inputs)
10 print(outputs)
```

```
[ 86.83 129.6  120.89 135.48  82.17 147.74 138.25  63.07 121.6  112.95
 137.55 134.38 122.42 135.72  60.54  75.81  81.02 127.31  56.62  58.69
   48.93  73.57 126.16  57.92  47.77 117.12  59.91 105.88  85.68  96.49
   64.27 119.81 133.44 142.18 120.95  92.42  94.22 105.4  48.36  52.92
 146.31 104.17  50.17  41.5  132.06 140.63 117.5  82.57 117.63 141.56]
[1.290893 2.034977 1.84501  2.201767 1.389632 2.218678 2.127228 0.959054
 2.029469 1.623609 2.111638 2.09194  2.012386 1.89553  1.004256 1.250228
 1.368325 1.830127 1.000719 0.906513 0.867629 1.065907 1.979544 1.001403
 0.796199 1.771816 0.867878 1.587176 1.25434  1.40047  1.002361 1.9972
 1.894479 2.293443 1.823577 1.340533 1.358613 1.602167 0.750759 0.863093
 2.30035  1.741468 0.7448  0.763732 2.093772 1.986868 1.90702  1.289541
 1.86578  2.231851]
```

```
1 # Display dataset and see that there is a rather clear linear trend.  
2 plt.figure(figsize = (12, 8))  
3 plt.scatter(inputs, outputs)  
4 plt.xlabel("Surface (in sqm)")  
5 plt.ylabel("Price (in millions SGD)")  
6 plt.show()
```



# Our first toy example

**Our first toy example:** simplified version of the apartment price prediction.

## 3. Model (M) definition and its trainable parameters (P):

- Here, our model will consist of a **linear regression** model.

s\$1,680,000

Negotiable

3  3  1184 sqft s\$ 1,418.92 psf

### Details

Property Type	Floor Size
Executive Condominium For Sale	1184 sqft
Developer	PSF
Tampines EC Pte Ltd	S\$ 1,418.92 psf
Furnishing	Floor Level
Partially Furnished	High Floor
Tenure	TOP
99-year Leasehold	January, 2016



# Our first model, the linear regression

## Definition (Linear Regression):

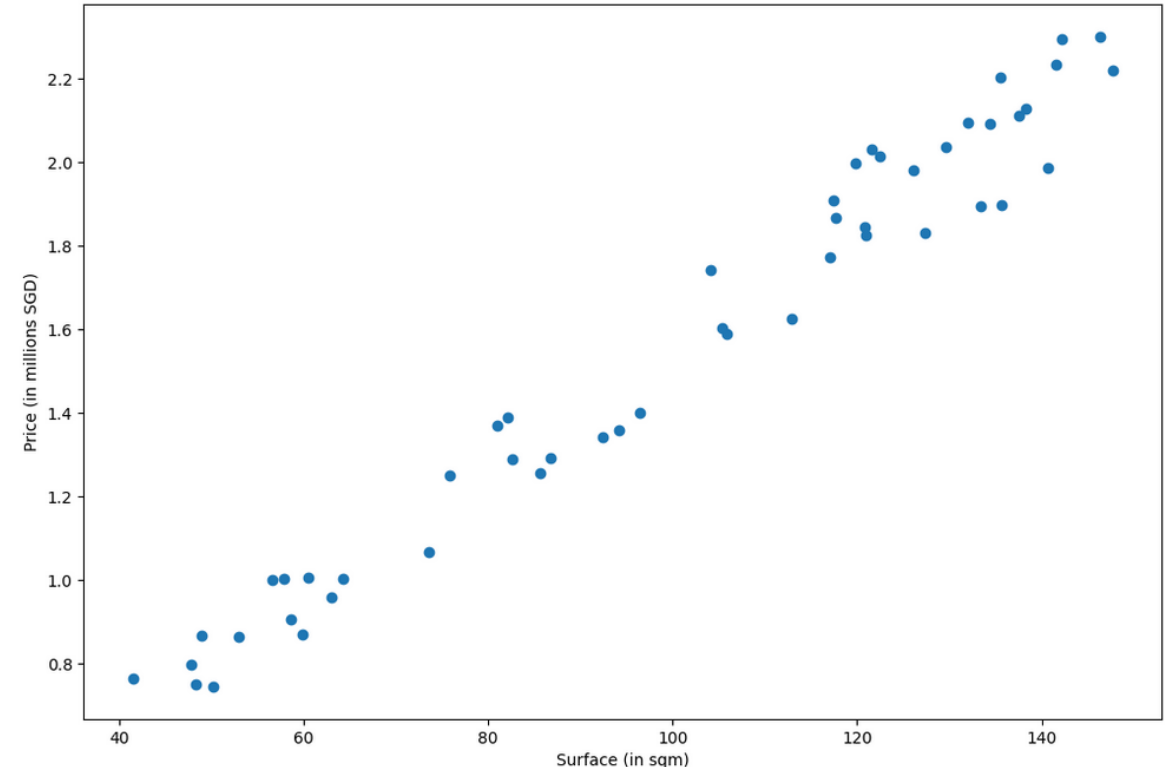
**Linear Regression** is a model, which **assumes that there is a linear relationship between inputs and outputs.**

It therefore consists of two trainable parameters  $(a, b)$ , to be freely chosen.

These will connect any input  $x_i$  to its respective output  $y_i$ , with the following equation:

$$y_i \approx a x_i + b$$

```
1 # Display dataset and see that there is a rather clear linear trend.
2 plt.figure(figsize = (12, 8))
3 plt.scatter(inputs, outputs)
4 plt.xlabel("Surface (in sqm)")
5 plt.ylabel("Price (in millions SGD)")
6 plt.show()
```





# Our first model, the linear regression

## Definition (**Linear Regression**):

**Linear Regression** is a model, which **assumes that there is a linear relationship between inputs and outputs.**

It therefore consists of two trainable parameters  $(a, b)$ , to be freely chosen.

These will connect any input  $x_i$  to its respective output  $y_i$ , with the following equation:

$$y_i \approx a x_i + b$$

```
1 # Linear regression has two trainable parameters (a and b).
2 # Other parameters, like min_surf, max_surf, n_points will
3 # help get points for the upcoming matplotlib displays.
4 def linreg_matplotlib(a, b, min_surf, max_surf, n_points = 50):
5     x = np.linspace(min_surf, max_surf, n_points)
6     y = a*x + b
7     return x, y
```

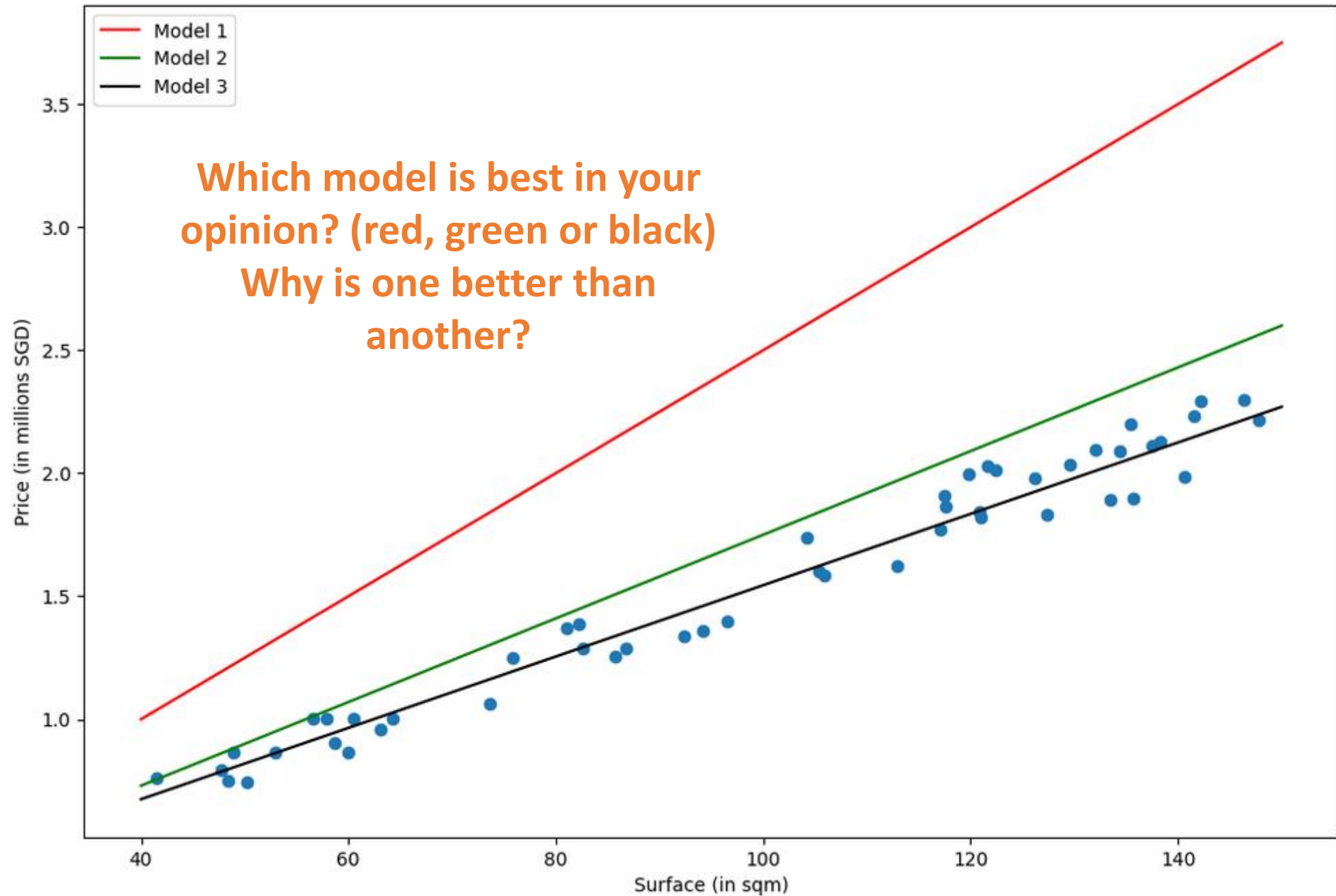
# Let us try different values of (a, b)

Model 1

Model 2

Model 3

```
1 # Display dataset
2 plt.figure(figsize = (12, 8))
3 plt.scatter(inputs, outputs)
4 plt.xlabel("Surface (in sqm)")
5 plt.ylabel("Price (in millions SGD)")
6
7 # Add some Linreg (model 1)
8 a1 = 25000/1000000
9 b1 = 0
10 linreg_dataset1_inputs, linreg_dataset1_outputs = linreg_matplotlib(a1, b1, min_surf, max_surf, n_points)
11 plt.plot(linreg_dataset1_inputs, linreg_dataset1_outputs, 'r', label = "Model 1")
12
13 # Another Linreg (model 2)
14 a2 = 17000/1000000
15 b2 = 50000/1000000
16 linreg_dataset2_inputs, linreg_dataset2_outputs = linreg_matplotlib(a2, b2, min_surf, max_surf, n_points)
17 plt.plot(linreg_dataset2_inputs, linreg_dataset2_outputs, 'g', label = "Model 2")
18
19 # A final Linreg (model 3)
20 a3 = 14500/1000000
21 b3 = 95000/1000000
22 linreg_dataset3_inputs, linreg_dataset3_outputs = linreg_matplotlib(a3, b3, min_surf, max_surf, n_points)
23 plt.plot(linreg_dataset3_inputs, linreg_dataset3_outputs, 'k', label = "Model 3")
24
25 # Display
26 plt.legend(loc = 'best')
27 plt.show()
```



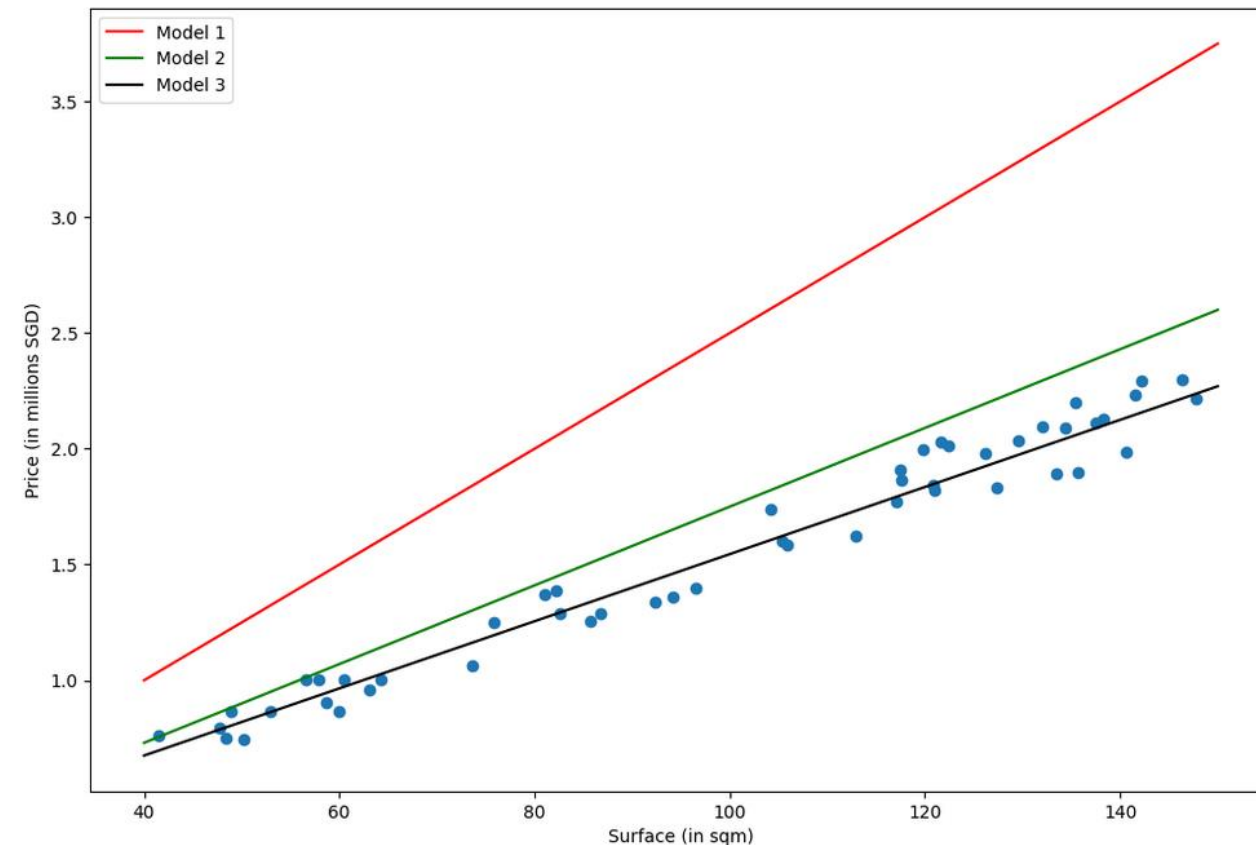
# The need for a loss function

Which model is best in your opinion? (red, green or black)  
Why is one better than another?

Telling which model is best can prove difficult, especially for close values of  $(a, b)$ .

Good news, there is one last element we have not used.

4. **Loss (L)** function to measure the performance of the model on said task and dataset.



# The need for a loss function

## Definition (**Loss** function):

A **loss function** (also known as a **cost function**) is a mathematical function that measures the difference between the predicted output of a model and the true output we should be matching.

It is used to

- **evaluate the performance of the model,**
- And more specifically **evaluate how good our choice of trainable parameters** (here, the parameters are  $a$  and  $b$ ) are at fitting the data.

It is used to guide the optimization process during **training**.

# The need for a loss function

## Definition (**The Mean Square Error loss function**):

In our linear model, a good loss function we could use consists of the **Mean Square Error (MSE) loss function**.

The MSE is calculated by calculating the square difference between:

- The **prediction**  $p_i = ax_i + b$  formed by the model for some given inputs  $x_i$ ,
- The **true value**  $y_i$  that we should be matching.

We then repeat this operation for every sample  $i$ , and average those errors together, i.e.

$$L(a, b, x, y) = \frac{1}{N} \sum_{i=1}^N (ax_i + b - y_i)^2$$

# The need for a loss function

Recall the definition of the MSE loss function

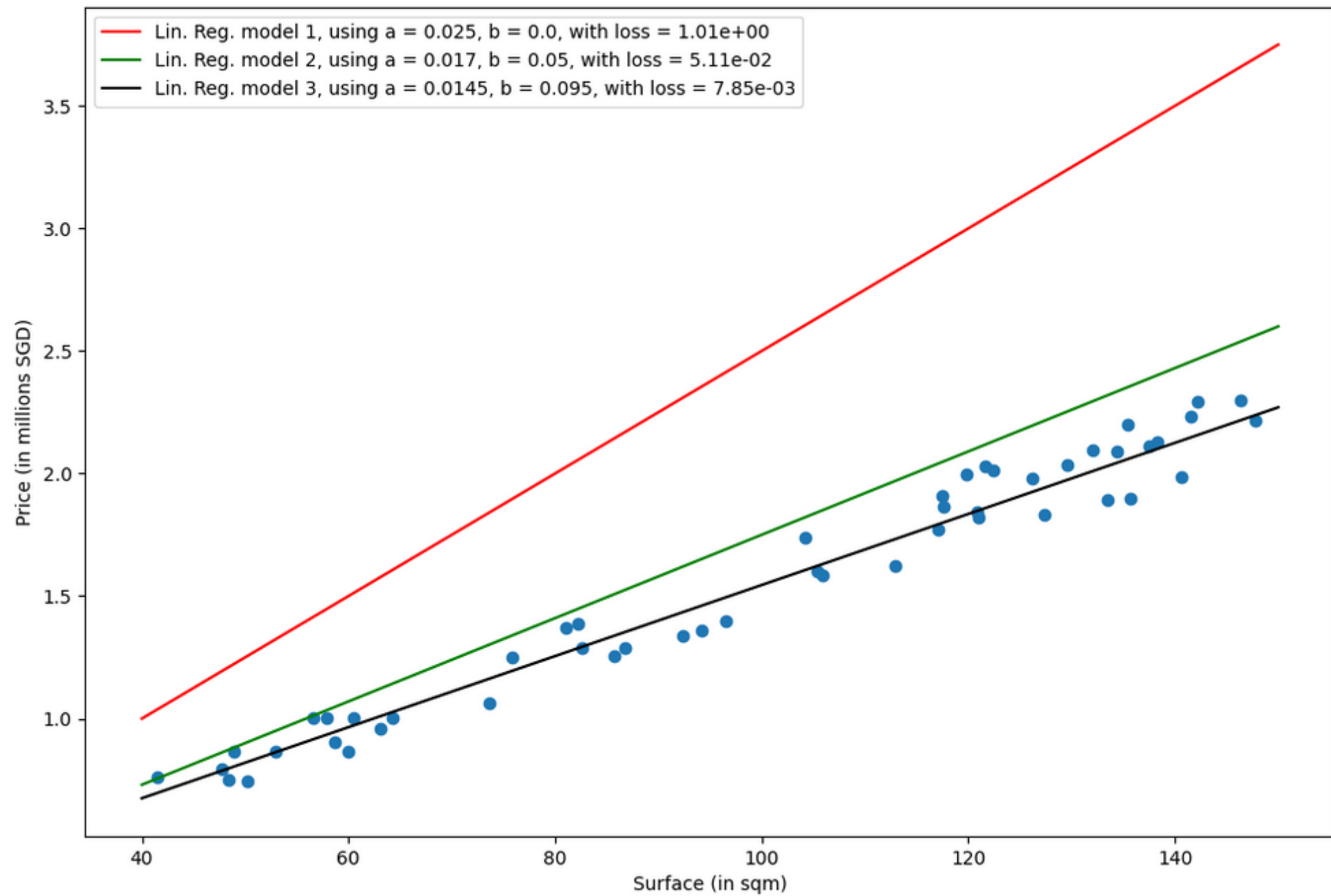
$$L(a, b, x, y) = \frac{1}{N} \sum_{i=1}^N (ax_i + b - y_i)^2$$

```
In [9]: 1 # Mean square error as a loss function
        2 # Displaying loss using exponential notation (XXXe-YYY)
        3 def loss_mse(a, b, x, y):
        4     val = np.sum((y - (a*x + b))**2)/x.shape[0]
        5     return '{:.2e}'.format(val)
```

```
In [10]: 1 # The lower the loss function, the better the linear regression values (a, b) fit the dataset.
        2 loss1 = loss_mse(a1, b1, inputs, outputs)
        3 loss2 = loss_mse(a2, b2, inputs, outputs)
        4 loss3 = loss_mse(a3, b3, inputs, outputs)
        5 print(loss1, loss2, loss3)
```

```
1.01e+00 5.11e-02 7.85e-03
```

Restricted



Restricted



# Training a model

## Definition (**Training a model**):

**Training a model** consists of finding the best values for the trainable parameters of the chosen model, for the given task and dataset.

The goal of training is to find the set of model parameters that minimize the loss function. Once a minimal value is obtained on the loss function, then we can claim that the “optimal” trainable parameters have been found (or in other word, the model has been trained).

# Training a linear regression model

In the case of linear regression, it consists of solving the following optimization problem.

$$(a^*, b^*) = \operatorname{argmin}_{a, b} [L(a, b, x, y)]$$

That is

$$(a^*, b^*) = \operatorname{argmin}_{a, b} \left[ L(a, b, x, y) = \frac{1}{N} \sum_{i=1}^N (ax_i + b - y_i)^2 \right]$$

# Training a linear regression model

## Definition (The **normal equation** for linear regression):

Solving this optimization problem can be done analytically, as it can be proven that the optimal choice of parameters  $W^* = \begin{pmatrix} b^* \\ a^* \end{pmatrix}$  follows the **normal equation**, below.

With

$$X = \begin{pmatrix} 1 & x_1 \\ \dots & \dots \\ 1 & x_N \end{pmatrix}$$

And

$$Y = \begin{pmatrix} y_1 \\ \dots \\ y_N \end{pmatrix}$$

$$W^* = (X^T X)^{-1} X^T Y$$

Finally,  $X^T$  denotes the transposed matrix of  $X$  and  $(X)^{-1}$  its inverse.

# Proof of the Normal Equation

Using the notations

$$X = \begin{pmatrix} 1 & x_1 \\ \dots & \dots \\ 1 & x_N \end{pmatrix},$$

$$Y = \begin{pmatrix} y_1 \\ \dots \\ y_N \end{pmatrix},$$

$$W^* = \begin{pmatrix} b^* \\ a^* \end{pmatrix}$$

We have, in matrix form

$$XW^* = \begin{pmatrix} a^*x_1 + b^* \\ \dots \\ a^*x_N + b^* \end{pmatrix}$$

And, also

$$XW^* - Y = \begin{pmatrix} a^*x_1 + b^* - y_1 \\ \dots \\ a^*x_N + b^* - y_N \end{pmatrix}$$

# Proof of the Normal Equation

Our loss function, the MSE  $L$ , is therefore defined, in matrix format as:

$$L = \frac{1}{N} (XW^* - Y)^T (XW^* - Y)$$

This is equivalent to

$$L = \frac{1}{N} ((XW^*)^T - Y^T)(XW^* - Y)$$

We then get

$$L = \frac{1}{N} (W^{*T} X^T - Y^T)(XW^* - Y)$$

Expanding

$$L = \frac{1}{N} (W^{*T} X^T X W^* - Y^T X W^* - W^{*T} X^T Y + Y^T Y)$$

# Proof of the Normal Equation

The expression

$$L = \frac{1}{N} (W^{*T} X^T X W^* - Y^T X W^* - W^{*T} X^T Y + Y^T Y)$$

Recall that we are trying to find the solution to optimization problem  
 $(a^*, b^*) = \operatorname{argmin}_{a,b} [L(a, b, x, y)]$

Recognizing that  $Y^T X W^* = W^{*T} X^T Y$ , differentiating our function  $L$  with respect to  $W$  and equating it to zero gives:

$$\frac{\partial L}{\partial W} = \frac{1}{N} (2X^T X W - 2X^T Y) = 0$$

# Proof of the Normal Equation

This is equivalent to

$$(X^T X W^* - X^T Y) = 0$$

Or,

$$X^T X W^* = X^T Y$$

**Assuming  $X^T X$  is invertible** (and that might not always be the case based on your dataset!), this finally gives:

$$W^* = (X^T X)^{-1} X^T Y$$



# Training a linear regression model

- Normal Equation formula:  $W^* = (X^T X)^{-1} X^T Y$

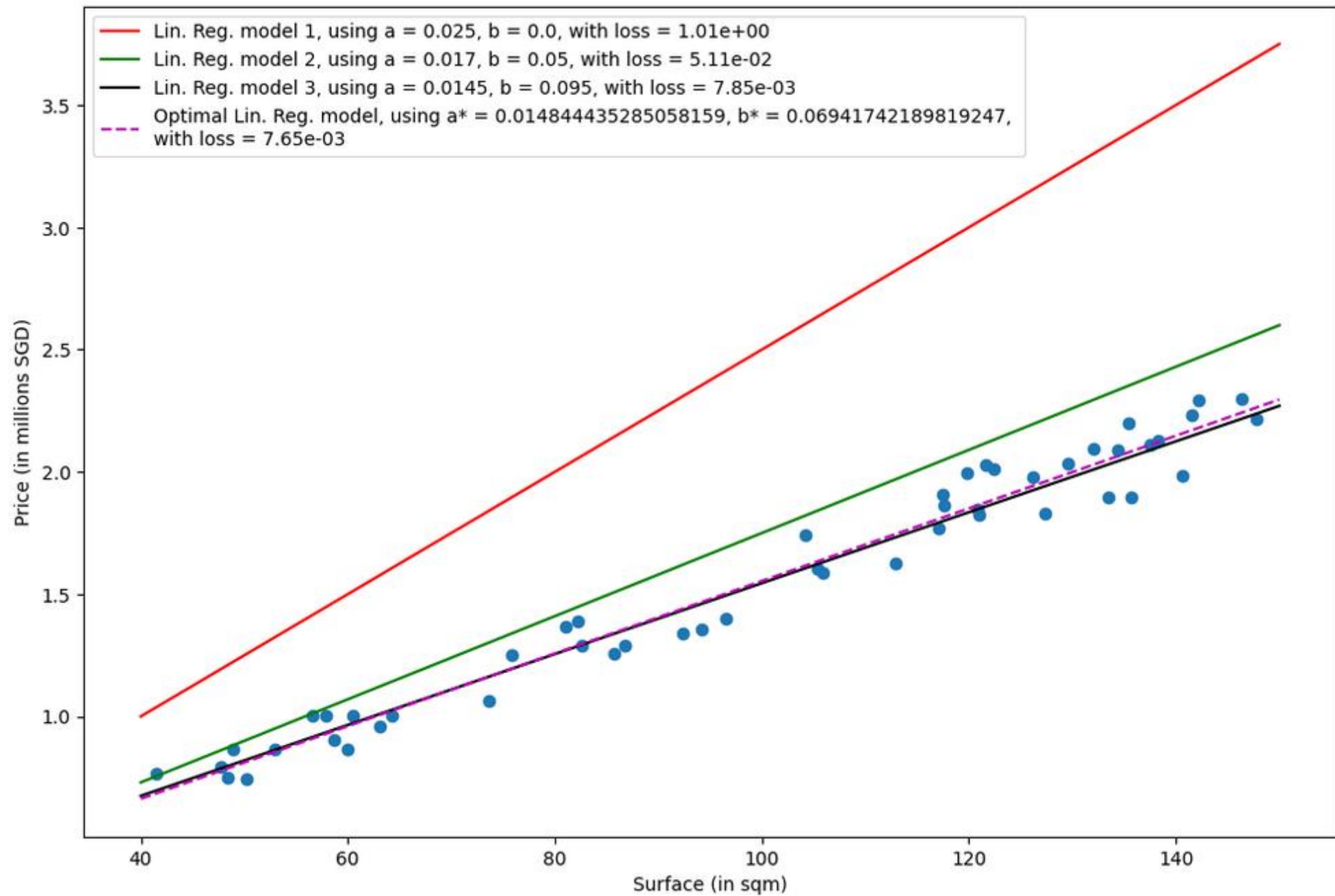
```
1 # Defining the X matrix, following the notation above, as a numpy array.  
2 X = np.array([[1, x_i] for x_i in inputs])  
3 print(X)
```

```
1 # While we are at it, let us define the transposed version of X.  
2 XT = np.transpose(X)  
3 print(XT)
```

```
1 # Defining the Y matrix, following the notation above, as a numpy array.  
2 Y = np.array([y_i for y_i in outputs])  
3 print(Y)
```

```
1 # Defining W_star according to our formula.  
2 W_star = np.matmul(np.linalg.inv(np.matmul(XT,X)), np.matmul(XT,Y))  
3 print(W_star)  
4 # Extracting a_star and b_star values from W_star.  
5 b_star, a_star = W_star[0, 0], W_star[1, 0]  
6 print("Optimal a_star value: ", a_star)  
7 print("The value we used for a in the mock dataset generation: ", 14373/1000000)  
8 print("Optimal b_star value: ", b_star)  
9 print("The value we used for b in the mock dataset generation: ", 100000/1000000)
```

Restricted



Restricted

# A problem with the normal equation

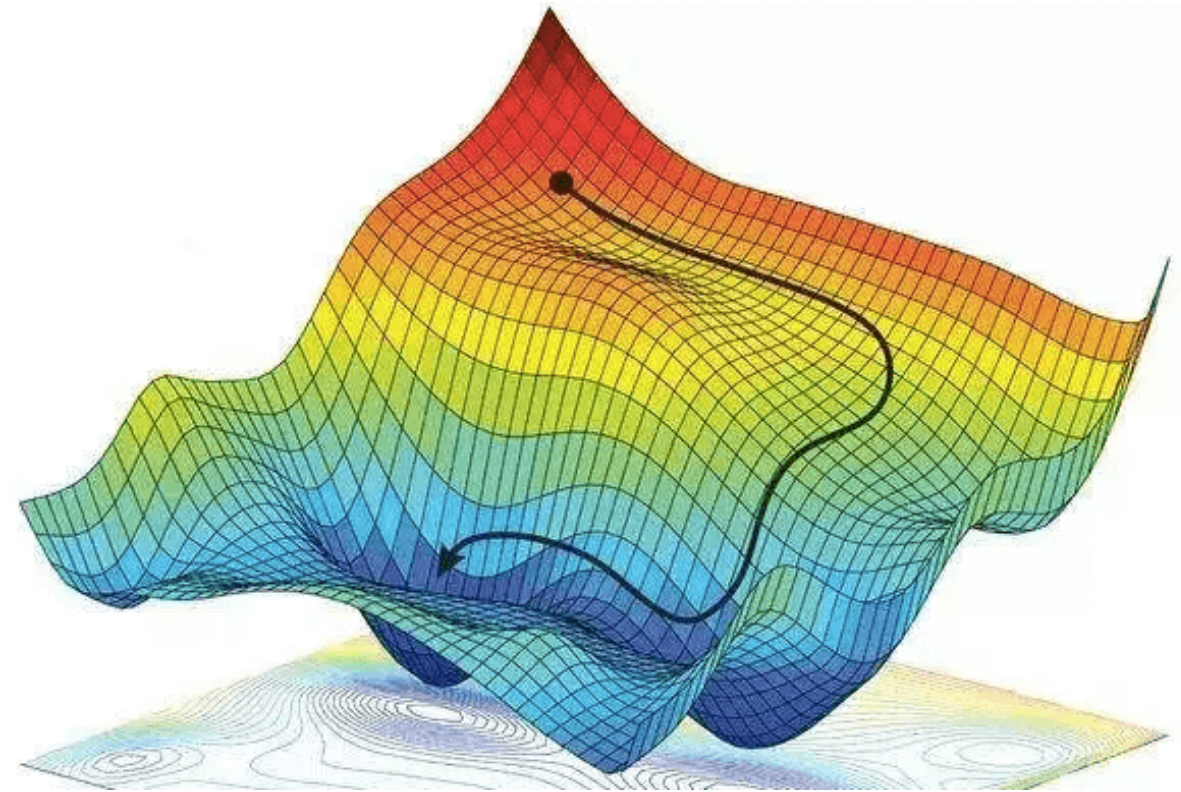
- The normal equation  $W^* = (X^T X)^{-1} X^T Y$  immediately gives the optimal set of parameters  $(a^*, b^*)$  to use for linear regression.
- However, it can become **computationally expensive when the number of features is very large.**
- It might even be **impossible to use when the matrix of feature variables  $(X^T X)^{-1}$  is not invertible.**
- **Problem: In these cases, the normal equation can be slow to compute, or it may not even have a solution.**
- **Another problem: More sophisticated models than linear regression will not have a normal equation, anyway.**

# Gradient Descent, to the rescue!

## Definition (**Gradient Descent**):

**Gradient Descent (GD)** is an iterative algorithm used to solve optimization problems.

- It starts with an initial set of parameters.
- It then repeatedly updates the parameters in the direction of the negative gradient of the given cost function, until it converges to a local minimum.



*Forgot about GD?  
Have a look at your notes from 10.013 M&A!*

# Gradient Descent, to the rescue!

## Definition (**Gradient Descent**):

**Gradient Descent (GD)** is an iterative algorithm used to solve optimization problems.

- It starts with an initial set of parameters.
- It then repeatedly updates the parameters in the direction of the negative gradient of the given cost function, until it converges to a local minimum.
- The normal equation has many problems...
- But GD, on the other hand can be used to find the optimal solution even when the normal equation is not applicable.
- The main advantage of gradient descent is that it can handle very large datasets and it can also be used for non-linear models.

# GD in Linear Regression

- In the case of Linear Regression, with the MSE loss defined earlier, we have

$$(a^*, b^*) = \operatorname{argmin}_{a, b} \left[ L(a, b, x, y) = \frac{1}{N} \sum_{i=1}^N (ax_i + b - y_i)^2 \right]$$

# GD in Linear Regression

We have the following derivatives with respect to  $a$  and  $b$ :

$$D_a = \frac{\partial L}{\partial a} = \frac{-2}{N} \sum_{i=1}^N x_i (y_i - (a x_i + b))$$

$$D_b = \frac{\partial L}{\partial b} = \frac{-2}{N} \sum_i y_i - (a x_i + b)$$



# GD in Linear Regression

The gradient descent **update rules** are therefore defined as

$$a \leftarrow a - \alpha D_a$$
$$a \leftarrow a + \frac{2\alpha}{N} \sum_{i=1}^N x_i (y_i - (a x_i + b))$$

And

$$b \leftarrow b - \alpha D_b$$
$$b \leftarrow b + \frac{2\alpha}{N} \sum_i y_i - (a x_i + b)$$

With parameter  $\alpha$  being the **learning rate** for the gradient descent, a parameter to be decided manually, later.

# GD in Linear Regression

## Gradient Descent Linear Regression procedure:

- We will then initialize  $a$  and  $b$  with some values (could be zero, random, or something else).
- For a given number of iterations, we will apply the two update rules defined earlier.
- (Optionally, we might decide to stop iterating, if we realize that the values of  $a$  and  $b$  are no longer changing. This is called **early stopping**, and is typically implemented by tracking the changes on each iteration and breaking if the changes are less than a threshold  $\delta$ .)

```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

```

Initialize a and b as you please.

```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

```

Number of samples in dataset.

```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

```

Two possible stopping conditions,  $\text{change} < \text{delta}$  or  $\text{counter} > \text{max\_count}$

```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

```

Update using our GD update rules from earlier

```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

```

Compute change on this iteration (to decide on early stopping or not)



```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

```

Calculate new MSE value using the new parameters a and b. We also keep track of these losses for display later

```

1 def gradient_descent_linreg(a_0, b_0, x, y, alpha = 1e-5, delta = 1e-5, max_count = 1000):
2     # Define the initial values of a and b as a_0 and b_0
3     a, b = a_0, b_0
4     # Define N as the number of elements in the dataset
5     N = len(x)
6     # Keep track of how much a and b changed on each iteration
7     change = float("Inf")
8     # Counter as safety to prevent infinite looping
9     counter = 0
10    # List of losses, to be used for display later
11    losses = []
12    while change > delta:
13        # Helper to visualize iterations of while loop
14        print("-----")
15        # Use gradient descent update rules for a and b
16        D_a = -2/N*(sum([x_i*(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
17        D_b = -2/N*(sum([(y_i - (a*x_i + b)) for x_i, y_i in zip(x, y)]))
18        a = a - alpha*D_a
19        b = b - alpha*D_b
20        print("Gradients: ", D_a, D_b)
21        print("New values for (a, b): ", a, b)
22        # Compute change
23        change = max(abs(alpha*D_a), abs(alpha*D_b))
24        print("Change: ", change)
25        # Compute and display current loss
26        loss = loss_mse(a, b, x, y)
27        losses.append(float(loss))
28        print("Loss: ", loss)
29        # Counter update, will break if iterations number exceeds max_count,
30        # to prevent gradient descent from going indefinitely.
31        # (Just a safety measure, for good practice, we would definitely prefer to see
32        # the while loop break "naturally", because change eventually fell under the threshold stop.)
33        counter += 1
34        if(counter > max_count):
35            print("Maximal number of iterations reached.")
36            break
37    return a, b, losses

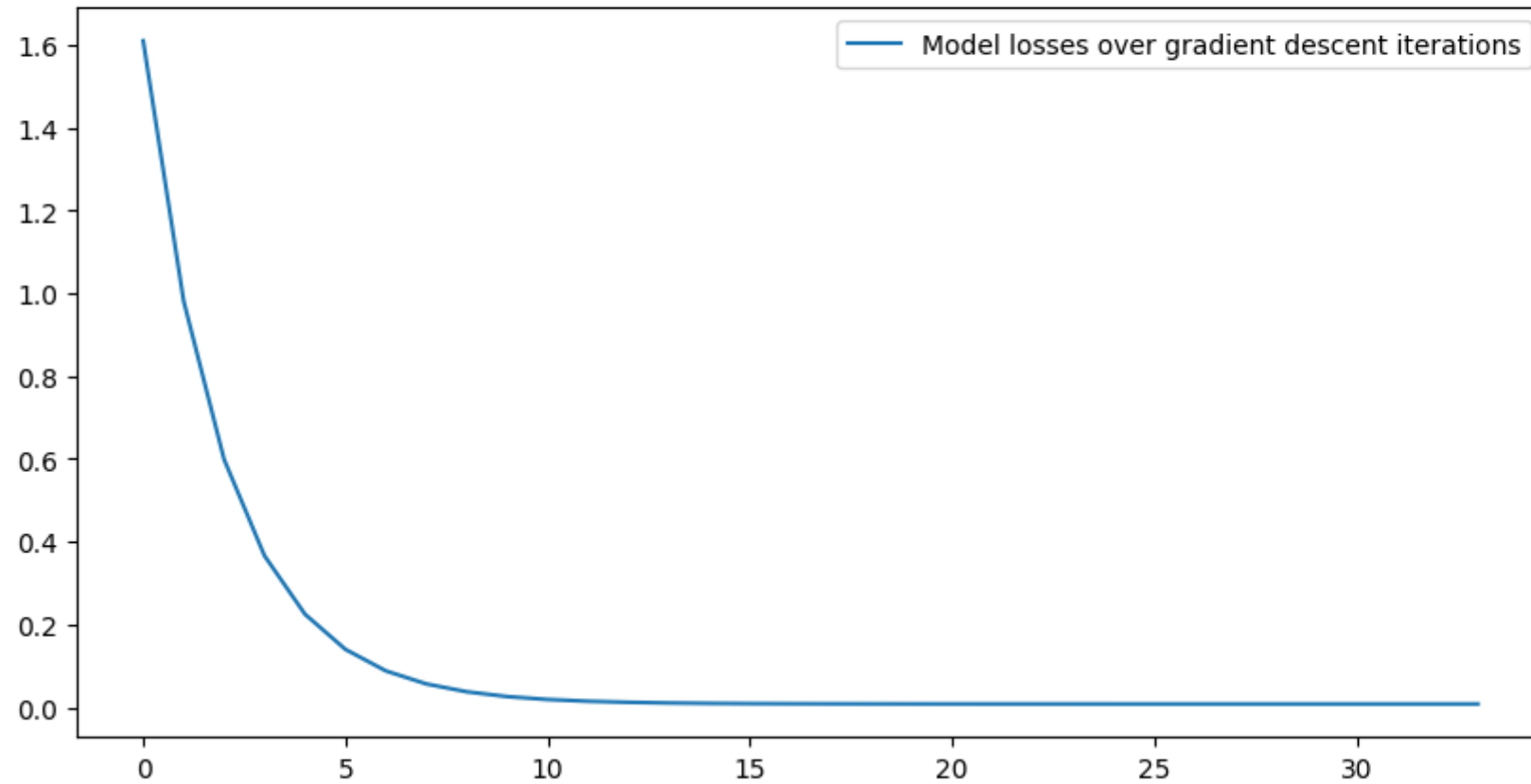
```

Return trained parameters and losses  
evolution on each iteration

```
: 1 a_gd, b_gd, losses = gradient_descent_linreg(a_0 = 0, b_0 = 0, x = inputs, y = outputs, alpha = 1e-5, delta = 1e-6)
```

```
-----  
Gradients: -342.3996226384001 3.105429920000001  
New values for (a, b): 0.0034239962263840013 -3.105429920000001e-05  
Change: 0.0034239962263840013  
Loss: 1.61e+00  
-----  
Gradients: -266.62829357559235 2.4212214483389536  
New values for (a, b): 0.006090279162139925 -5.526651368338955e-05  
Change: 0.0026662829357559236  
Loss: 9.82e-01  
-----  
Gradients: -207.62478780001163 1.8884249597020164  
New values for (a, b): 0.008166527040140042 -7.415076328040972e-05  
Change: 0.0020762478780001164  
Loss: 5.99e-01  
-----  
Gradients: -161.6784683033283 1.4735337252735503  
New values for (a, b): 0.009783311723173324 -8.888610053314522e-05  
Change: 0.001616784683033283
```

```
1 # Display dataset
2 plt.figure(figsize = (10, 5))
3 plt.plot(losses, label = "Model losses over gradient descent iterations")
4
5 # Display
6 plt.legend(loc = 'best')
7 plt.show()
```



# Checking for optimal parameters

- We have generated the dataset ourselves, so we know what should be the values for a and b!

```
1 print("Optimal a_star value: ", a_star)
2 print("Value for a_star, found by gradient descent: ", a_gd)
3 print("We used 14373/1000000 in the mock dataset generation, which is: ", 14373/1000000)
4 print("Optimal b_star value: ", b_star)
5 print("Value for b_star, found by gradient descent: ", b_gd)
6 print("The value we used in the mock dataset generation: ", 100000/1000000)
```

Optimal a\_star value: 0.014844435285058159

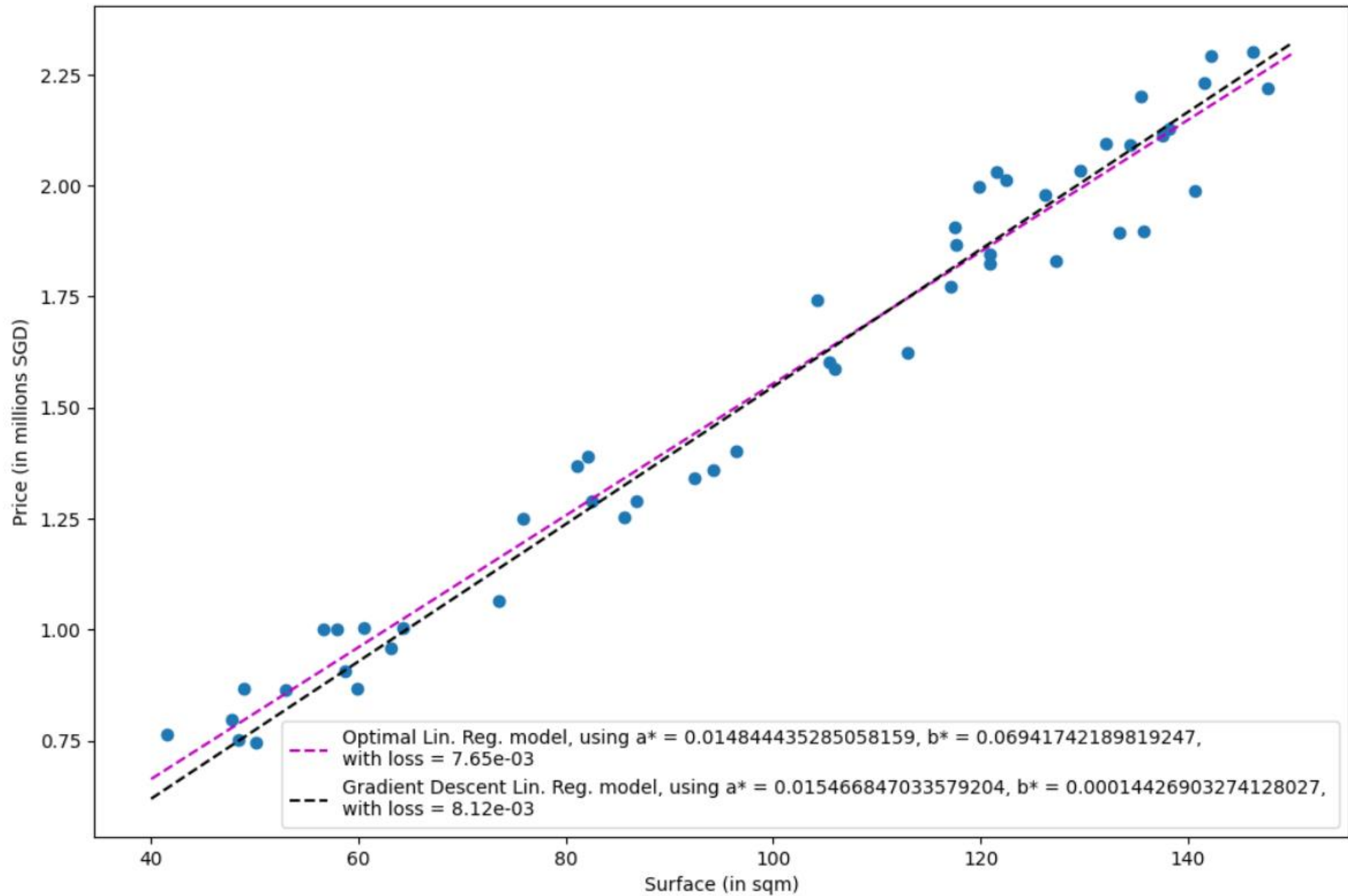
Value for a\_star, found by gradient descent: 0.015466847033579204

We used 14373/1000000 in the mock dataset generation, which is: 0.014373

Optimal b\_star value: 0.06941742189819247

Value for b\_star, found by gradient descent: 0.00014426903274128027

The value we used in the mock dataset generation: 0.1



# Using Sklearn for Linear regression

- In practice, we never implement the linear regression model ourselves (but it is a good practice to try it at least once!)
- It is often faster to rely on the **sklearn** library and use the **LinearRegression** object!

```
1 # Creating a sklearn Linear Regression model.
2 # It uses the same analytical formula from earlier, i.e.  $W^* = (X^T X)^{-1} X^T Y$ .
3 reg = LinearRegression().fit(sk_inputs, sk_outputs)
4 # The coefficients for  $a^*$  and  $b^*$  are found using coeff_ and intercept_ respectively.
5 a_sk = reg.coef_[0]
6 b_sk = reg.intercept_
7 print("Optimal a_star value: ", a_star)
8 print("Value for a_star, found by sklearn: ", a_sk)
9 print("Optimal b_star value: ", b_star)
10 print("Value for b_star, found by sklearn: ", b_sk)
```

Optimal a\_star value: 0.014844435285058159

Value for a\_star, found by sklearn: 0.014844435285058166

Optimal b\_star value: 0.06941742189819247

Value for b\_star, found by sklearn: 0.06941742189818956



# Predicting using Sklearn Linear regression

- After training, it is also good practice to check if the predictor makes sense, by asking it to predict the price of an apartment it has never seen before.
- Confirm the value manually, if possible.

```
1 # We can later use this Linear Regression model, to predict the price of
2 # a new apartment with surface 105 sqm (price in millions SGD).
3 new_surf = 105
4 pred_price = reg.predict(np.array([[new_surf]]))[0]
5 print(pred_price)
```

1.628083126829297

```
1 avg_price = 14373*new_surf + 100000
2 min_val = 0.9*avg_price
3 max_val = 1.1*avg_price
4 print("Min, max, avg prices: ", [min_val, max_val, avg_price])
```

Min, max, avg prices: [1448248.5, 1770081.5000000002, 1609165]