Data Science

Dr. Muhammad Adeel Nisar

Assistant Professor – Department of IT,
Faculty of Computing and Information Technology,
University of the Punjab, Lahore

Classes and Meeting Hours

Schedule	
	Tuesdays and Thursdays
Classes	11:30 to 13:00
Meeting hours	13:30 to 14:30

Office: Room 12, Ground Floor, Graduate Block, PUCIT

Email: adeel.nisar@pucit.edu.pk

Web: http://faculty.pucit.edu.pk/adeelnisar

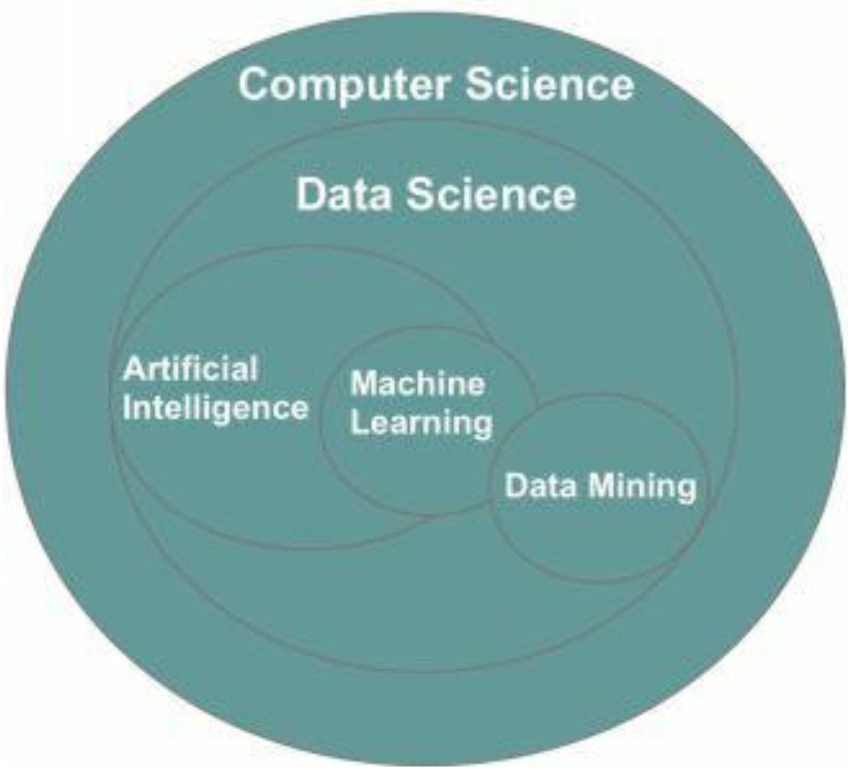
Contents

- Introduction to the Course
- What is Data Science?
- Exploratory Data Analysis
- Applications
- What is Machine Learning?
- Machine Learning Life-Cycle
- Types of Machine Learning
- Current and Proposed Final Year Projects
- Linear Regression
- Introduction to Python
- Revision of Related Concepts of Linear Algebra

What is Data Science?

- Data science is a subdomain of computer science that expertise to uncover actionable insights hidden in (an organization’s) data. These insights can be used to guide decision making and strategic planning.

- Data science uses the tools of artificial intelligence, machine learning and data mining.



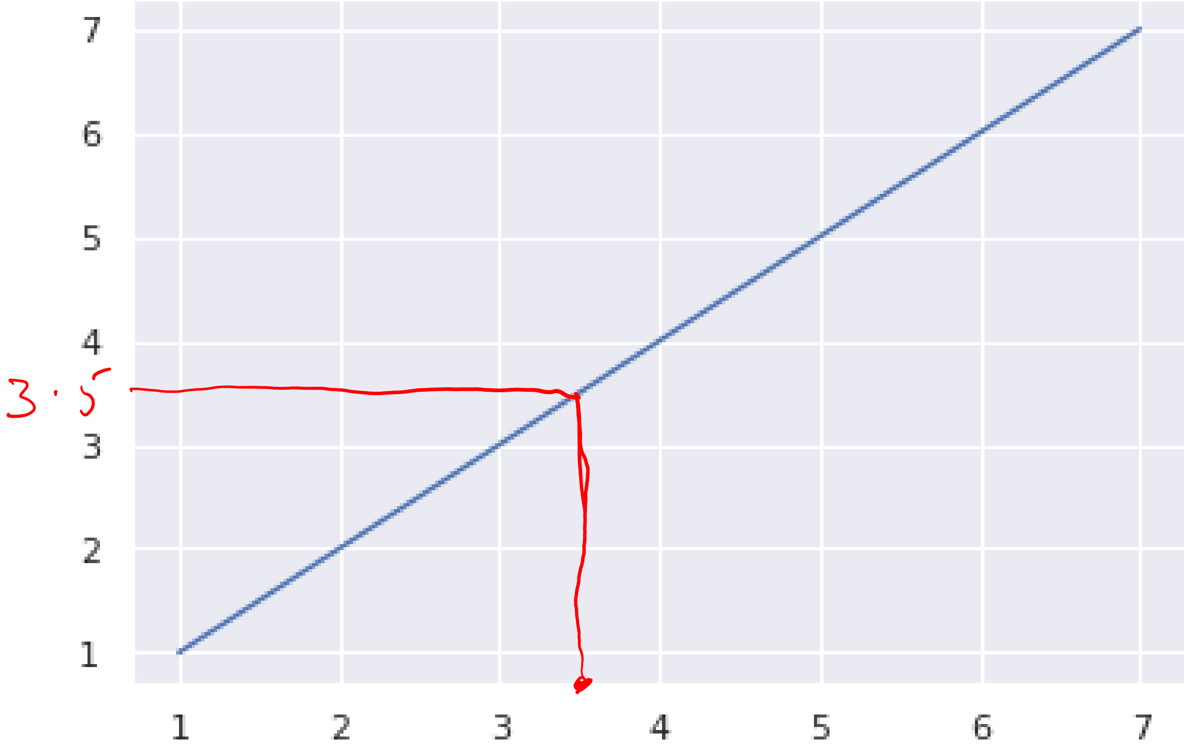
Exploratory Data Analysis

- `x = [1, 2, 3, 4, 5, 6, 7]`
- `y = [1, 2, 3, 4, 5, 6, 7]`

Exploratory Data Analysis

- **x** = [1, 2, 3, 4, 5, 6, 7]
- **y** = [1, 2, 3, 4, 5, 6, 7]

$x = 3.5$



Exploratory Data Analysis

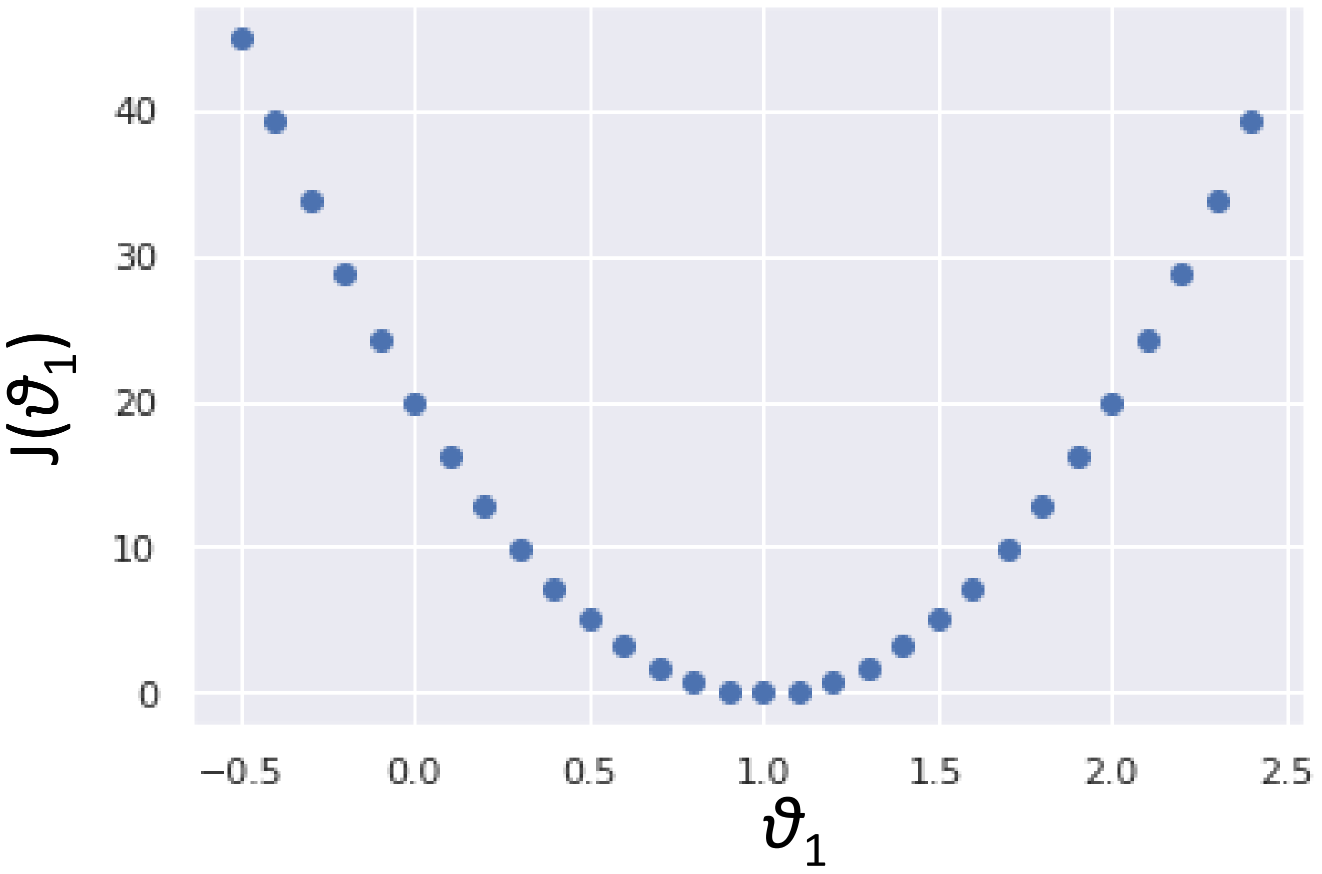
- **x** = [1, 2, 3, 4, 5, 6, 7]
- **y** = ~~[2, 4, 6, 8, 10, 12, 14]~~

y = [3, 5, 7, 9, 11, 13, 15]

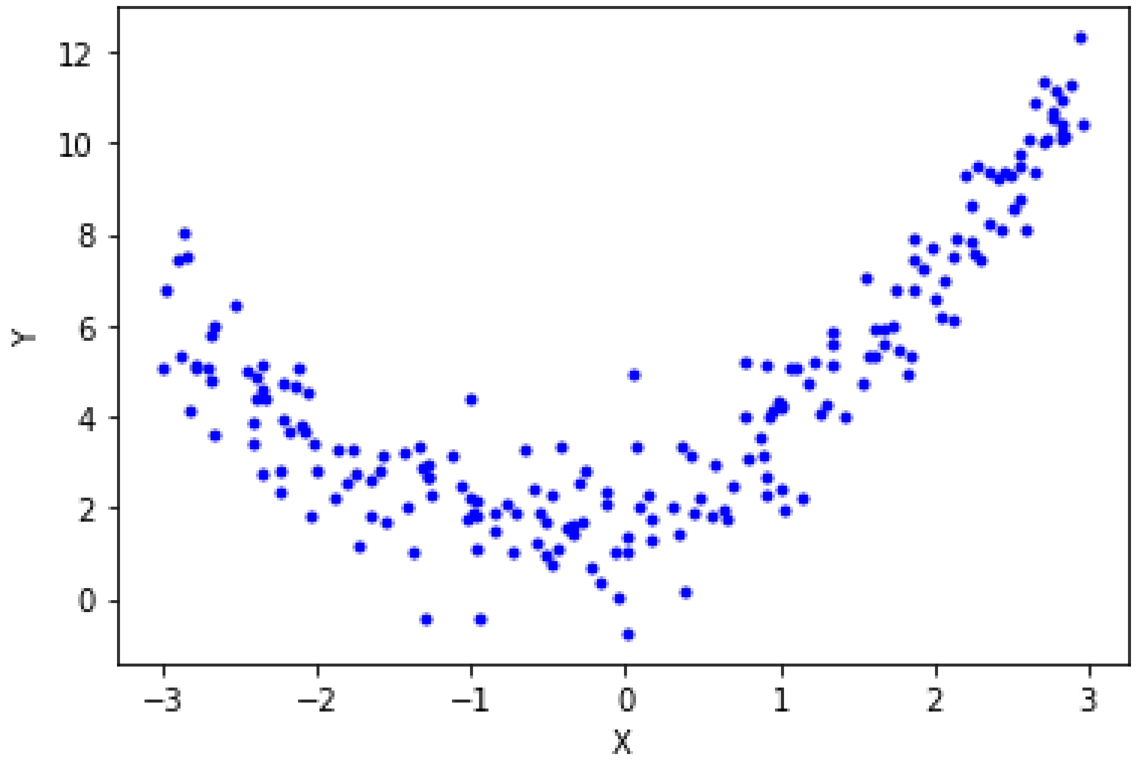
$$y = 2x + 1$$

Exploratory Data Analysis

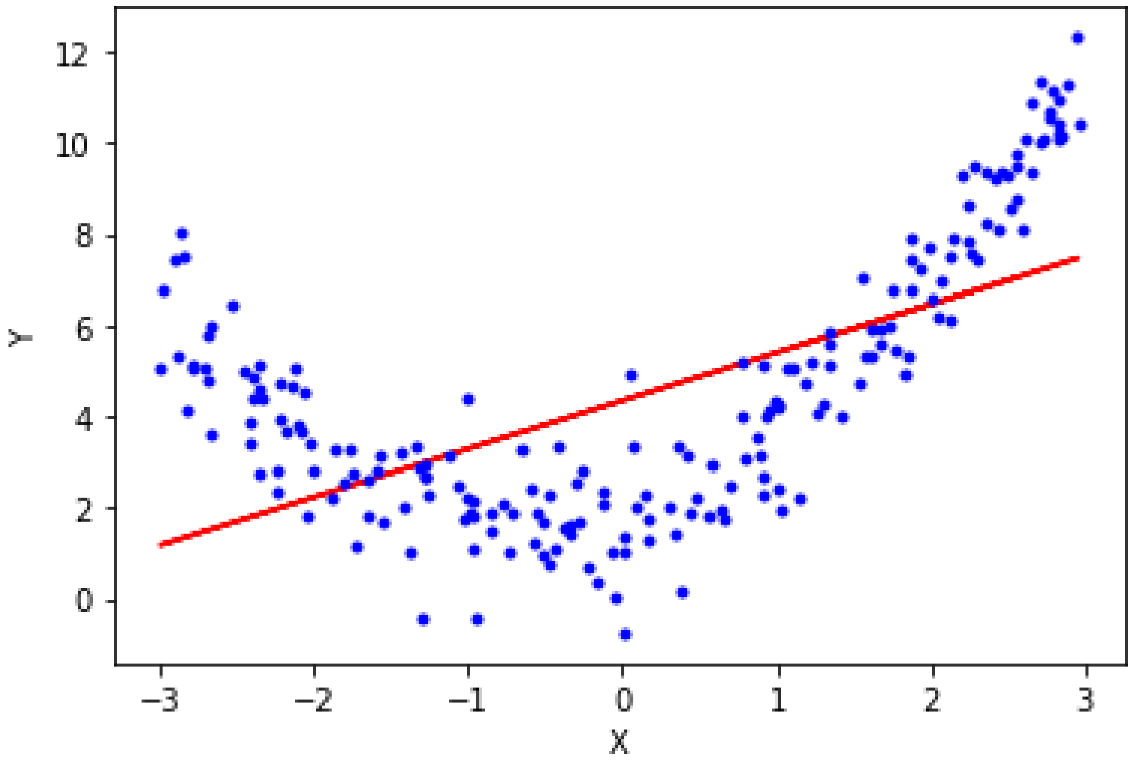
- θ_1 = [-0.5, -0.25, 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25]
- $J(\theta_1)$ =[45.0, 31.25, 20.0, 11.25, 5.0, 1.25, 0.0, 1.25, 5.0, 11.25, 20.0, 31.25]



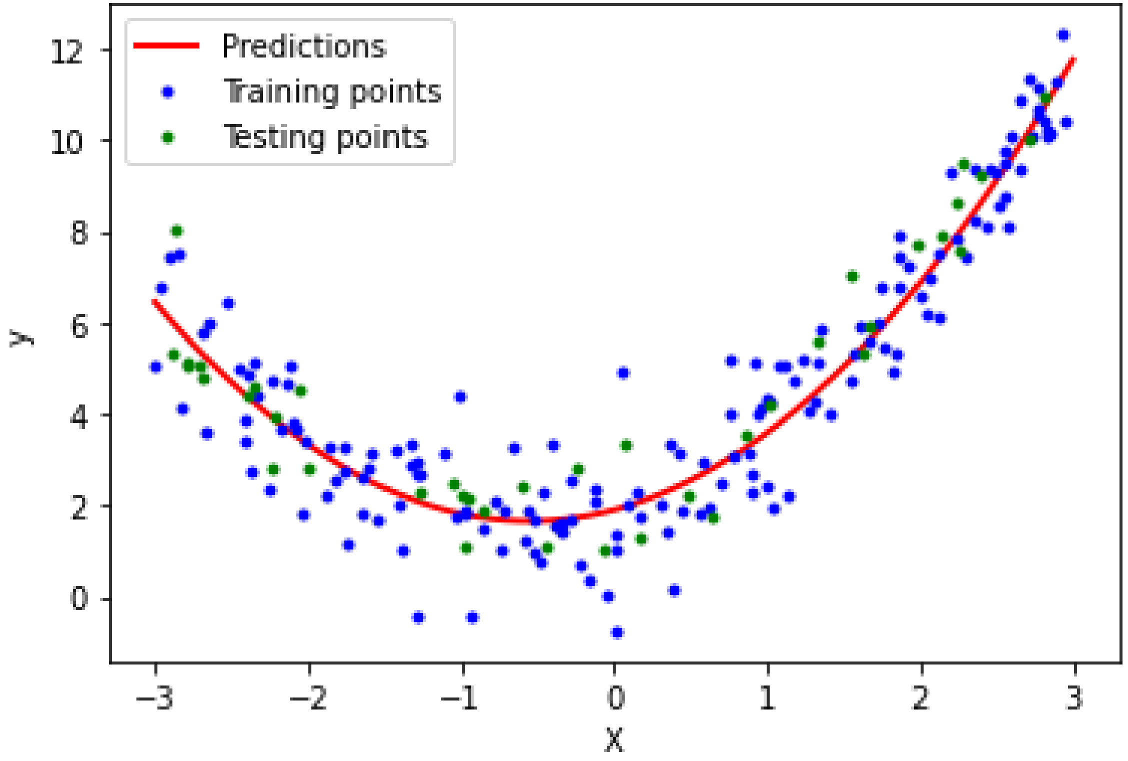
Exploratory Data Analysis



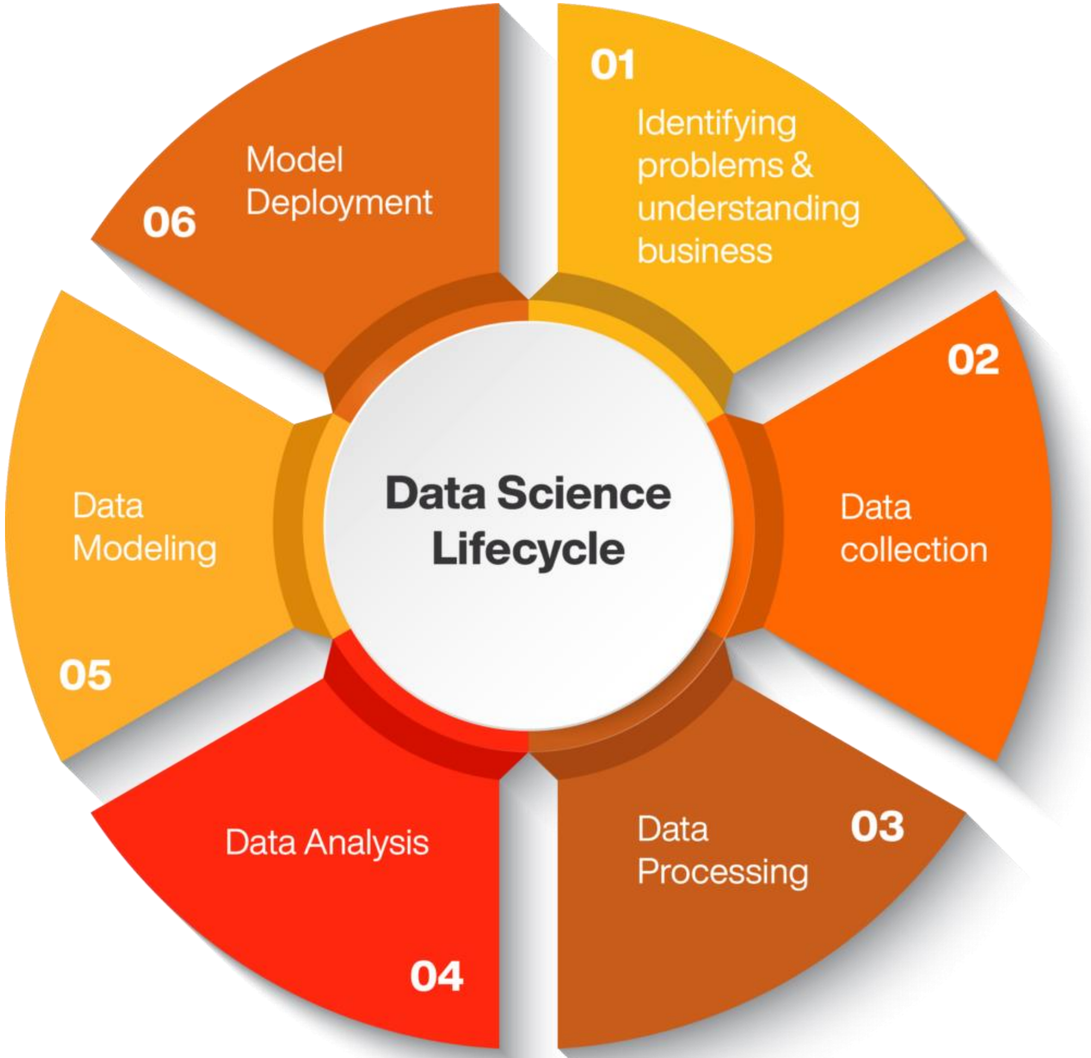
Exploratory Data Analysis



Exploratory Data Analysis



Data Science Life Cycle



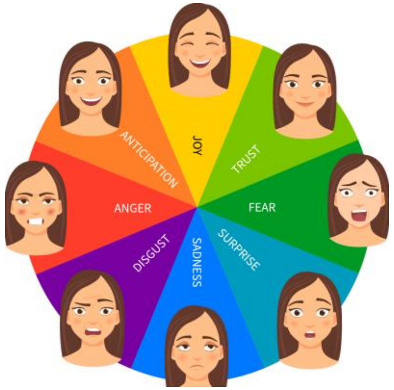
Data Science Life Cycle

- **Identifying problems and understanding business:** Discovering the answers for basic questions including requirements, priorities and budget of the project.
- **Data Collection:** Collecting data from relevant sources either in structured or unstructured form.
- **Data processing:** Processing and fine-tuning the raw data, critical for the goodness of the overall project.
- **Data analysis:** Capturing ideas about solutions and factors that influence the data life cycle.
- **Data modelling:** Preparing the appropriate model to achieve desired performance.
- **Model deployment:** Executing the analysed model in desired format and channel.

Applications/Use cases of Data Science

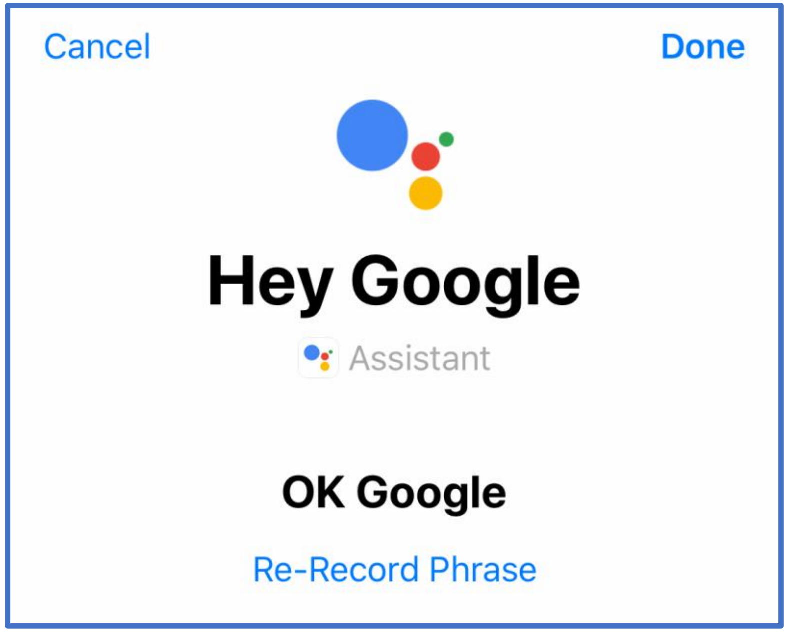
- Fraud and Risk Detection: An international bank delivers faster loan services with a mobile app using machine learning-powered credit risk models.
- Health Care: An activity recognition system may detect the level of independent living of an elderly person.
- Internet Search: All search engines (like Google, Yahoo, Bing, Ask, AOL, and so on) make use of data science algorithms to deliver the best result for our searched query in a fraction of seconds.
- Targeted Advertising, and many others…

Applications



Applications

- Virtual Personal Assistant



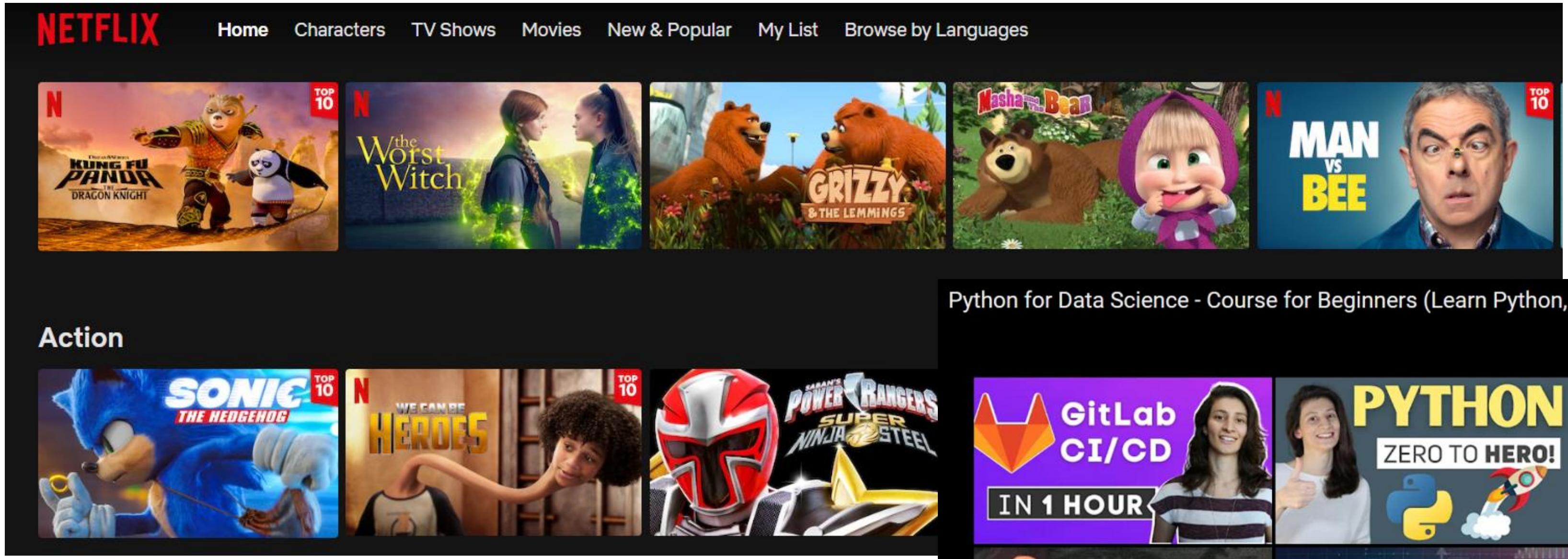
SILVIA
Intelligence On Command

amazon alexa

Bixby

Applications

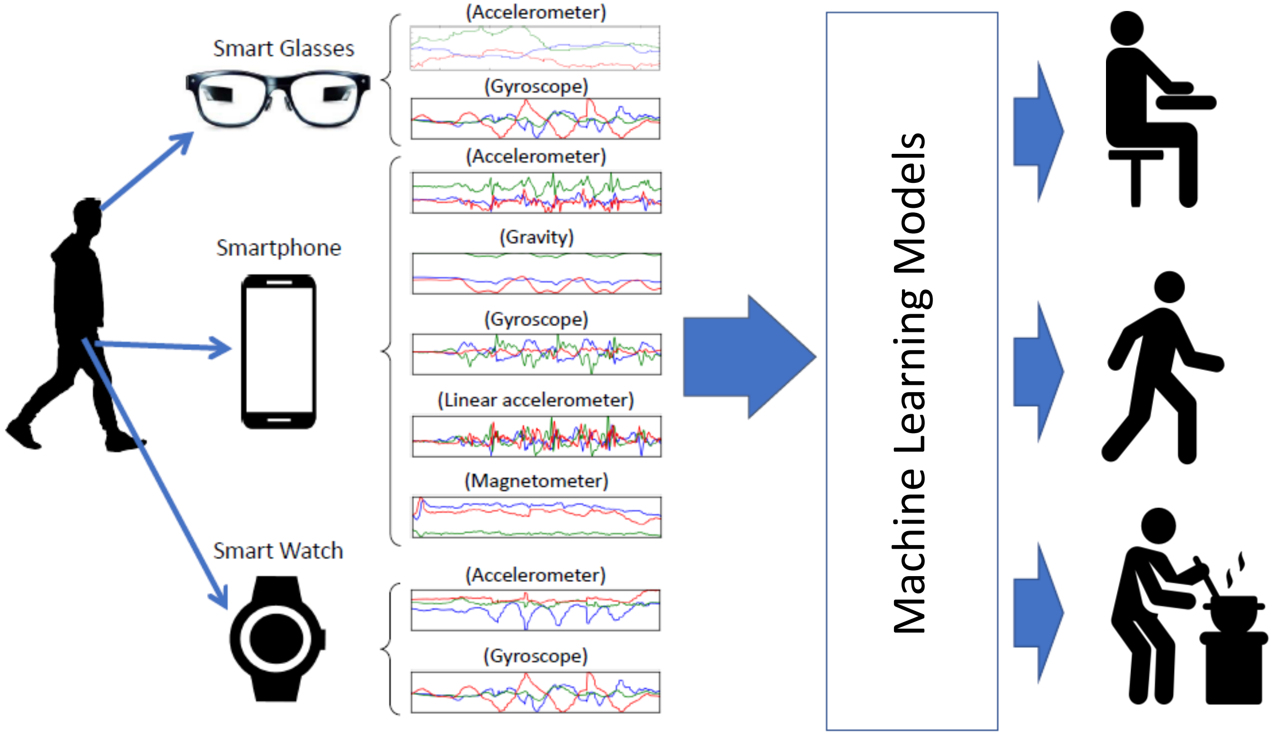
- Recommendation System



Applications

Activity Recognition / Sensors Data Understanding

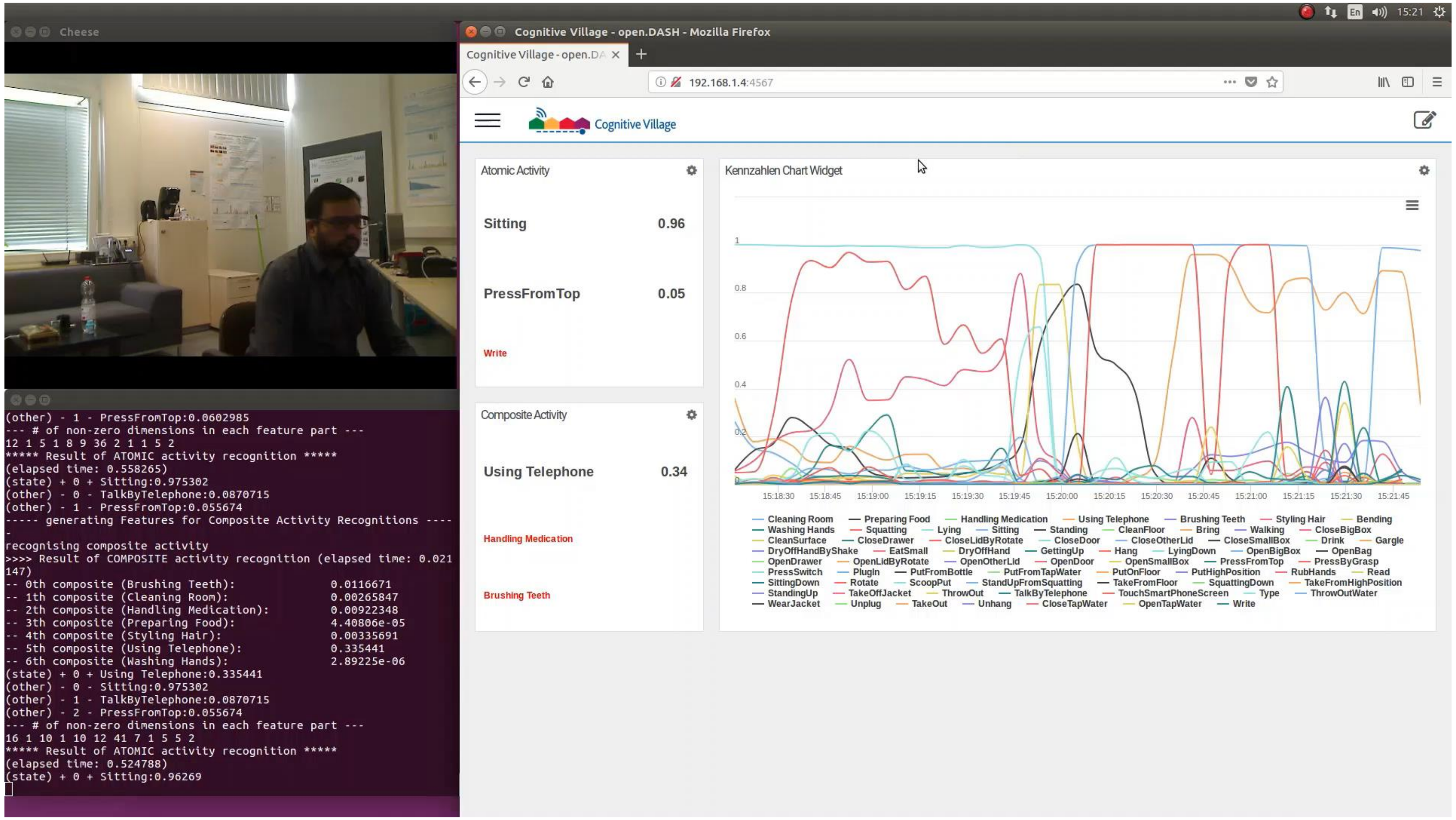
- Inertial Sensors



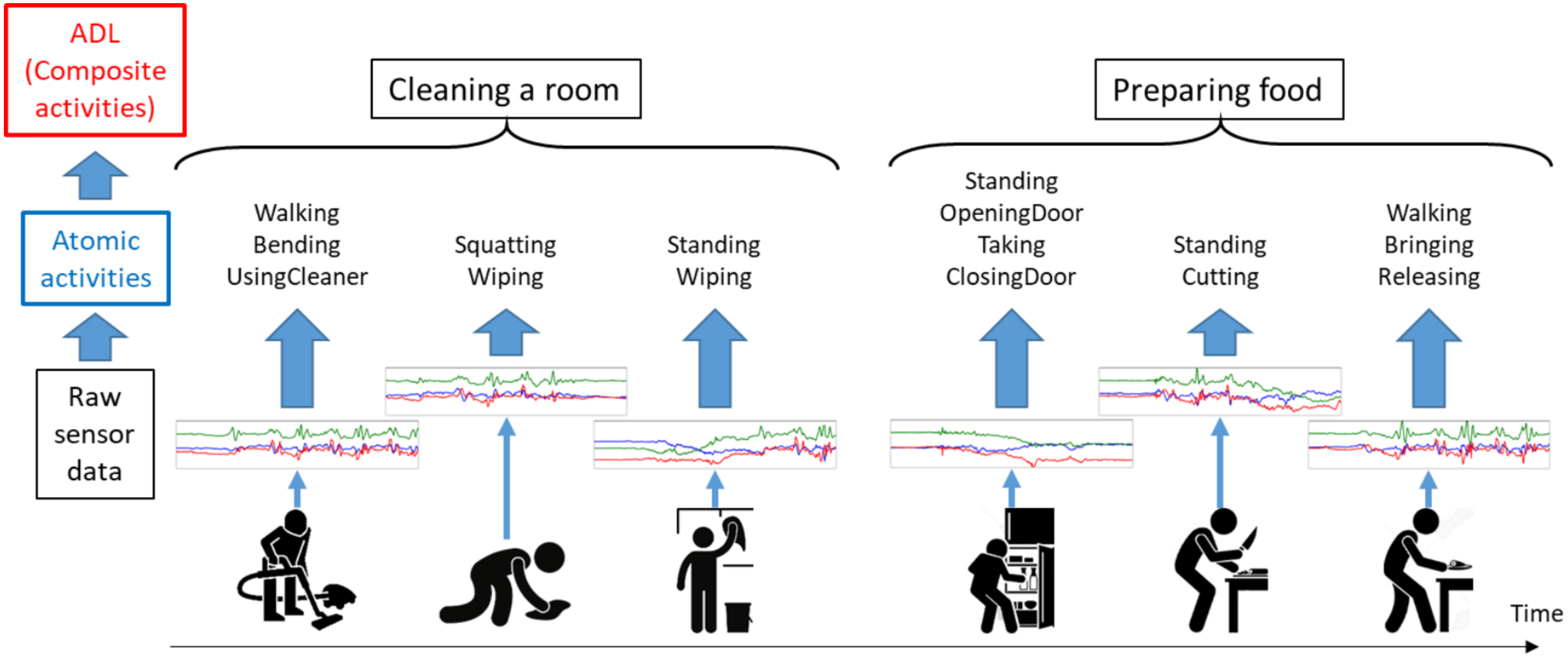
Applications

Recognition of Daily Activities

https://youtu.be/s5wZP4ArZtU



Application

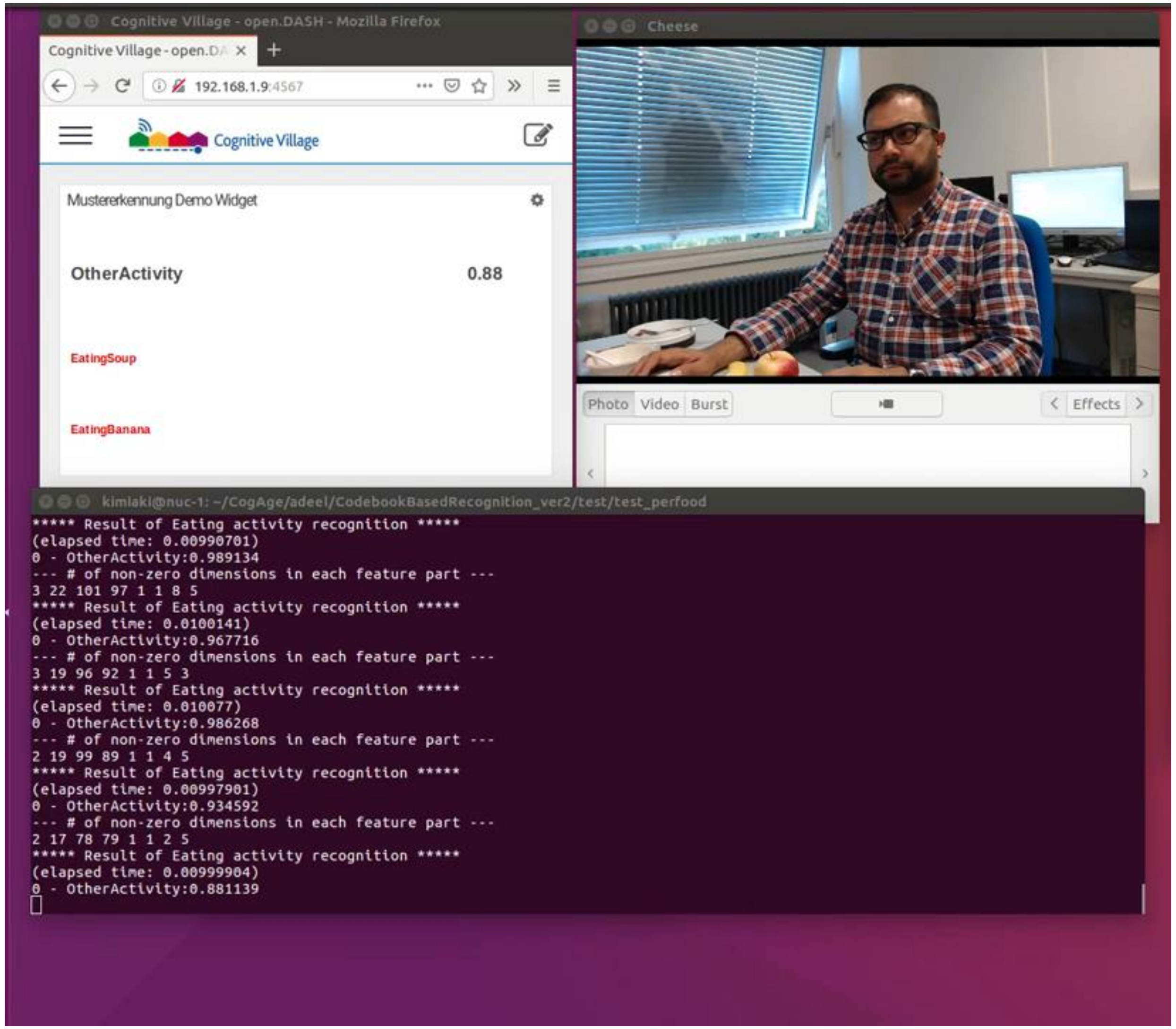


Recognition of Daily Activities

Applications

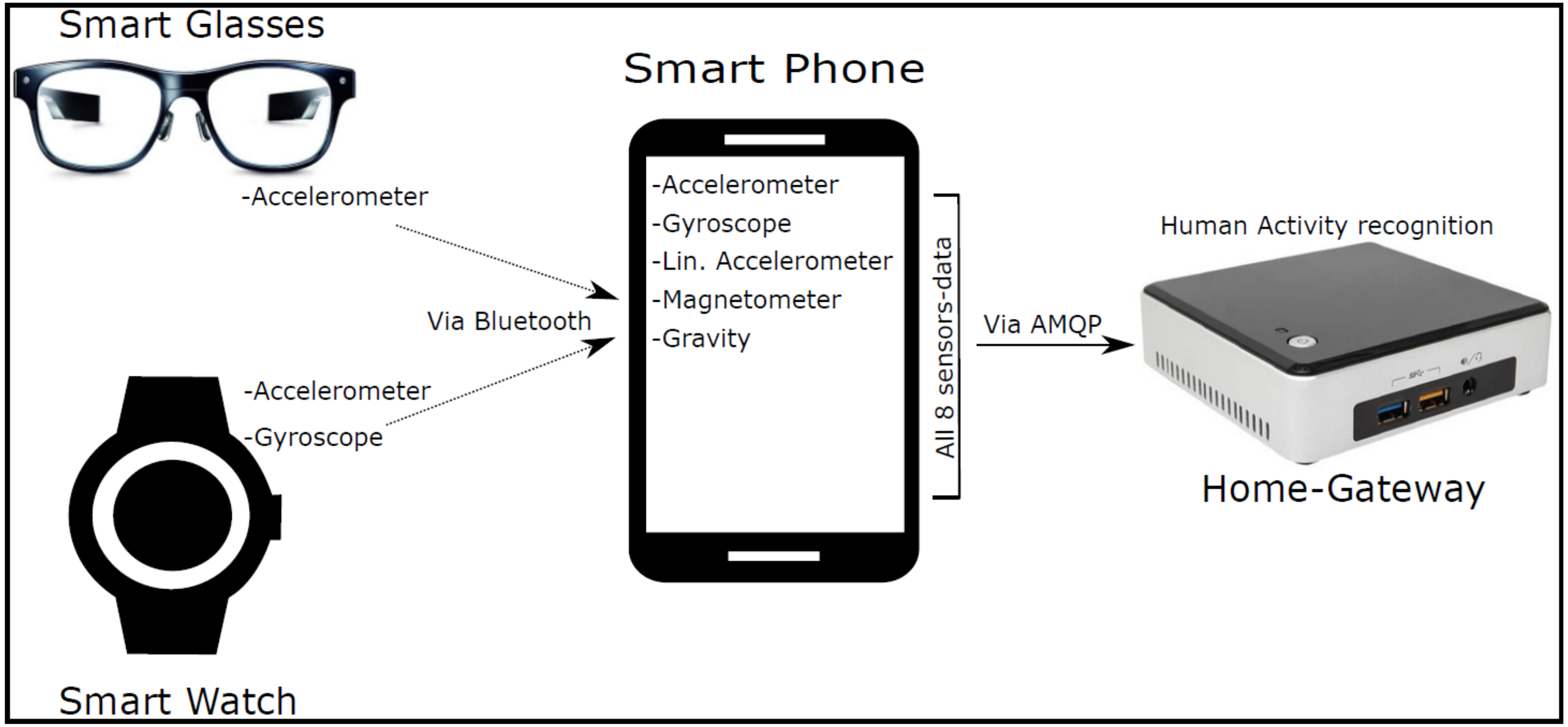
Recognition of Eating Activities

https://youtu.be/J4QLzRRmCY8



Current Final Year Projects

1. Sensor Data Acquisition System



2. Eating Activities Detection

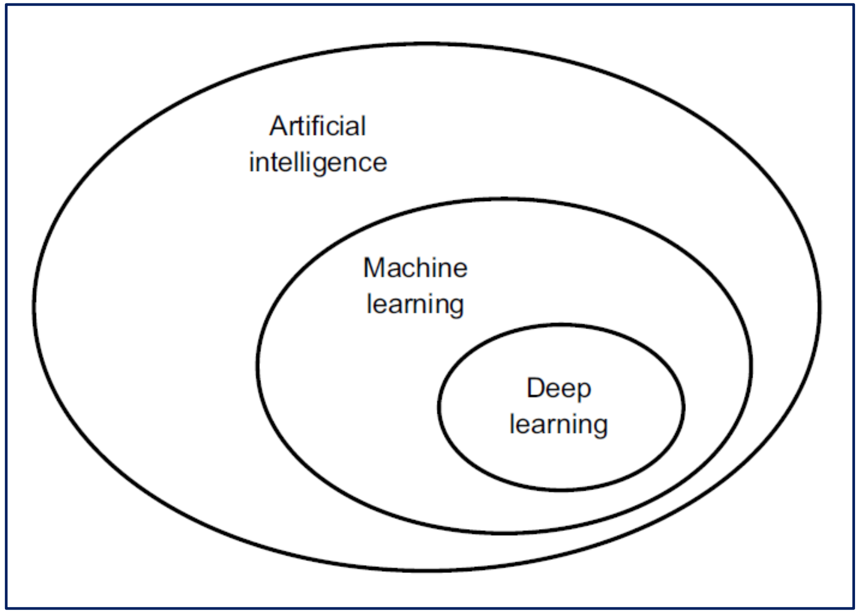
A Proposal for Final Year Project

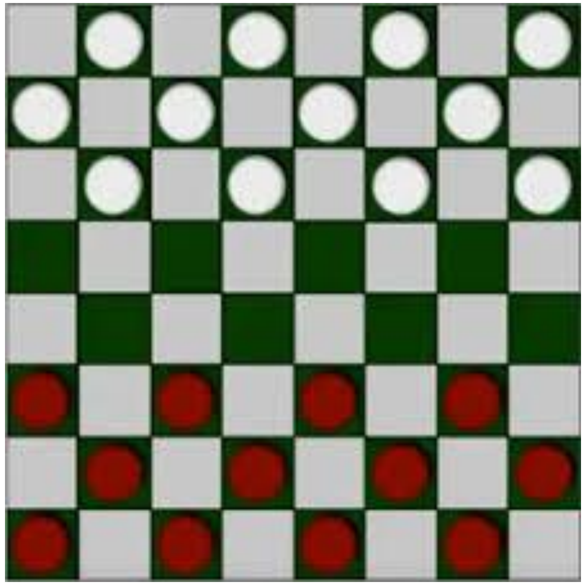
3. Detecting Genotype of Parkinson Patients

- Predicting Genotype based on Phenotype of Parkinson patients using Machine Learning
- Implementation and Comparison of Tabular Data Augmentation Techniques

What is Machine Learning?

- Machine learning (ML) is a branch of Artificial Intelligence (AI).
- Arthur Samuel (1959) defined ML as "Field of study that gives computers the ability to learn without being explicitly programmed"
  - Samuels wrote a checkers playing program
    - Had the program play 10000 games against itself
    - Work out which board positions were good and bad depending on wins/losses



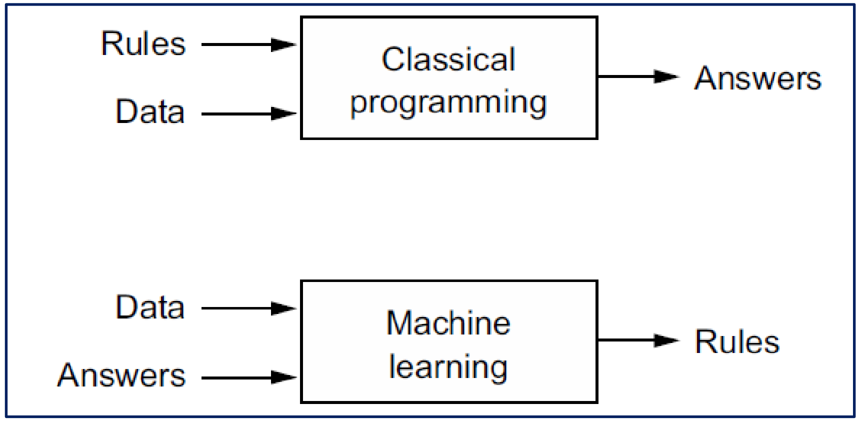


What is Machine Learning?

- Tom Michel (1999) defines ML as a well posed learning problem: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

The checkers example,

- E = 10000s games
- T is playing checkers
- P is probability of winning



Different Phases of Life-Cycle

1. • Data Acquisition → Tabular data, Image/video, Sensor data
2. • Data Preparation
3. • Feature Extraction → (1) Feature Engineering (2) Feature learning
4. • Train Model → LR, SVM, RF, ANN
5. • Test Model → For Evaluation and assessment
- • Evaluate and Improve

Types of Machine Learning

1
- **Supervised Machine Learning** assumes that a set of labelled training data is available and the classifier is designed by exploiting this a-priori known information.
- Two further types
  1
  - Regression
    - Linear Regression
    - Nonlinear Regression

  2
  - Classification
    - Logistic Regression
    - Naïve Bayes
    - Support Vector Machines etc

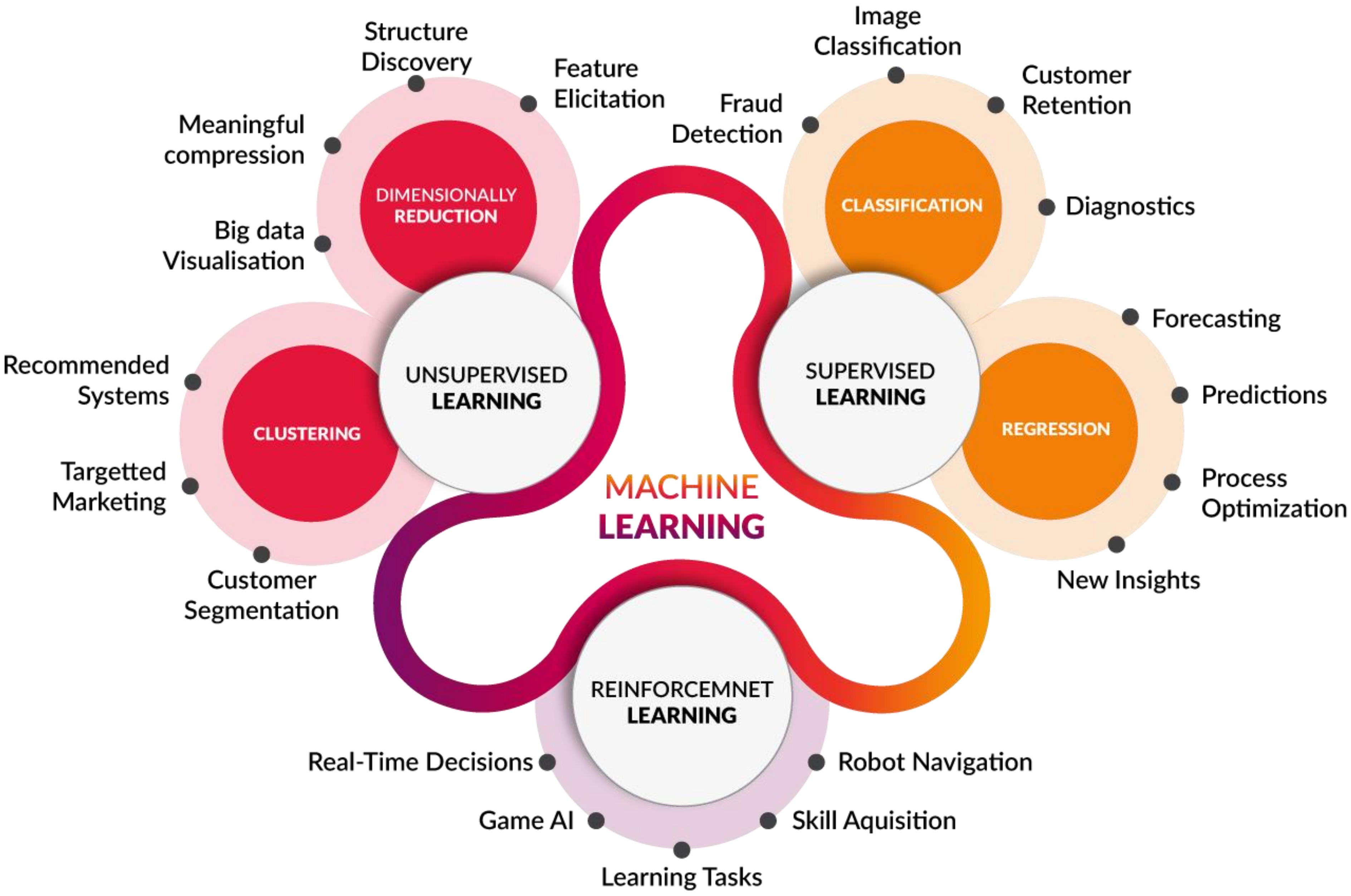
D.T / R.F

Types of Machine Learning

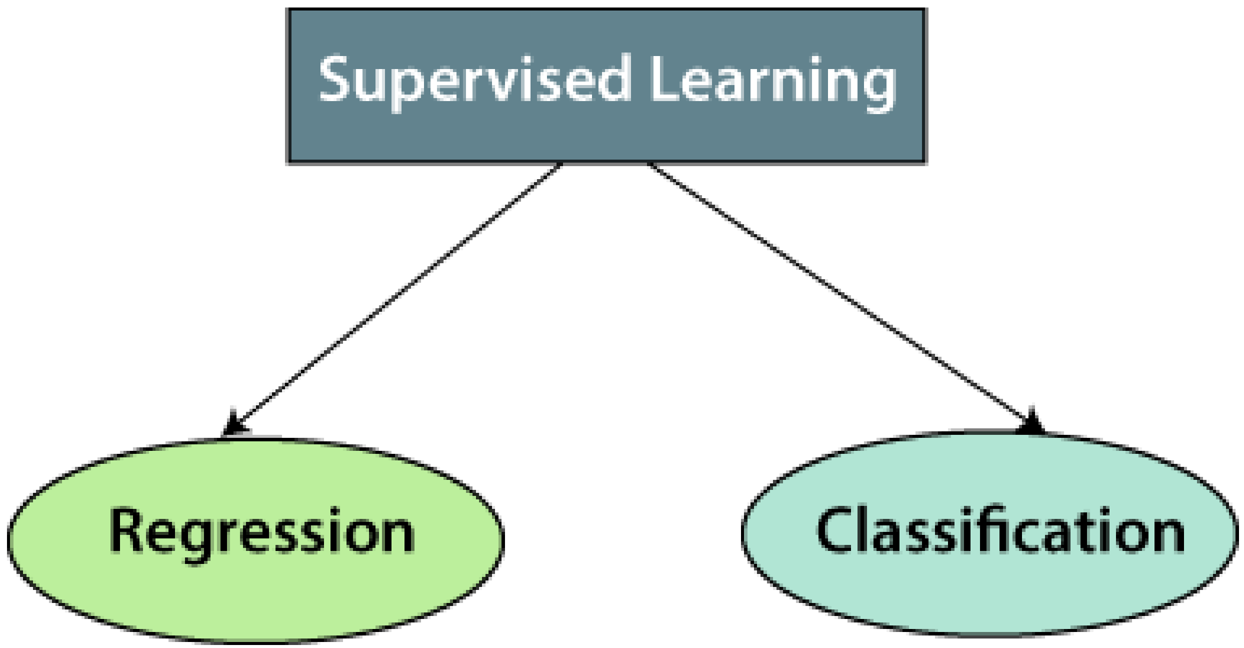
- **Unsupervised Machine Learning** clusters unlabeled training data described by feature vectors into similar groups
  - Clustering
    - K-Means Clustering
  - Dimensionality Reduction
    - Principal Component Analysis
    - Autoencoders

Types of Machine Learning

- In **Semi-supervised Machine Learning** the dataset contains both labeled and unlabeled examples. Usually, the quantity of unlabeled examples is much higher than the number of labeled examples. The goal of a semi-supervised learning algorithm is the same as the goal of the supervised learning algorithm.
- **Reinforcement Learning** solves a particular kind of problems where decision making is sequential, and the goal is long-term, such as game playing, robotics, resource management, or logistics.



Structure Discovery
Feature Elicitation
Meaningful compression
DIMENSIONALLY REDUCTION
Big data Visualisation
Image Classification
Customer Retention
Fraud Detection
CLASSIFICATION
Diagnostics
Recommended Systems
CLUSTERING
UNSUPERVISED LEARNING
Targetted Marketing
Customer Segmentation
SUPERVISED LEARNING
Forecasting
Predictions
REGRESSION
Process Optimization
New Insights
MACHINE LEARNING
REINFORCEMNET LEARNING
Real-Time Decisions
Robot Navigation
Game AI
Skill Aquisition
Learning Tasks



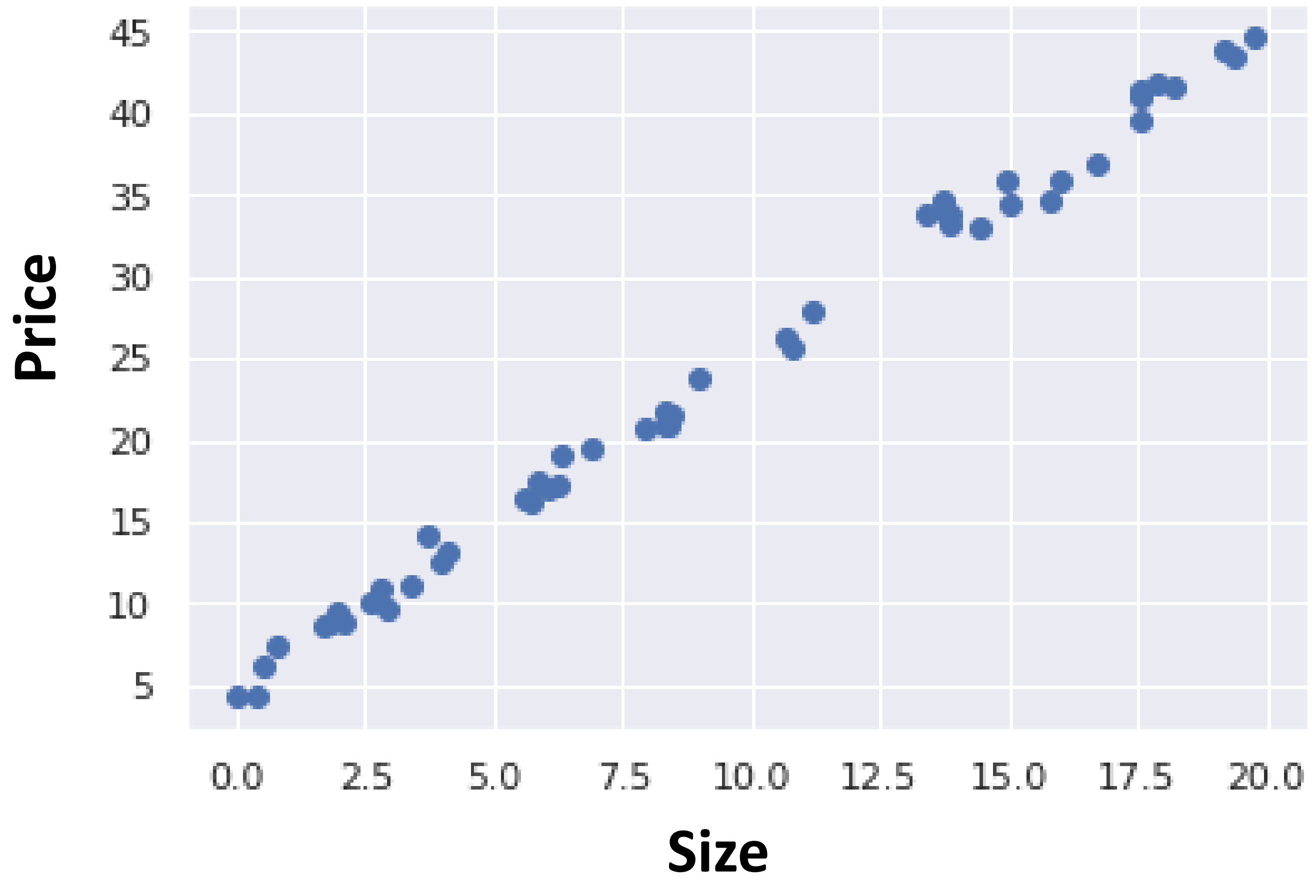
Supervised Learning
Regression
Classification

Regression

- A Supervised learning algorithm
- Taking input variables and trying to fit the output onto a continuous values.
- Linear regression with one variable is also known as "Univariate linear regression".
- Univariate linear regression is used when you want to predict a single output value y from a single input value x.
- The Hypothesis Function $y' = h\theta(x) = \theta_0 + \theta_1 x$
- Given the training data with right answers, predict the real-valued output for the test data.

Dataset and Plotted Graph

Input Data (x)	Correct Answer (y)
8.3	20.99
14.4	32.89
6.05	17.08
..	..



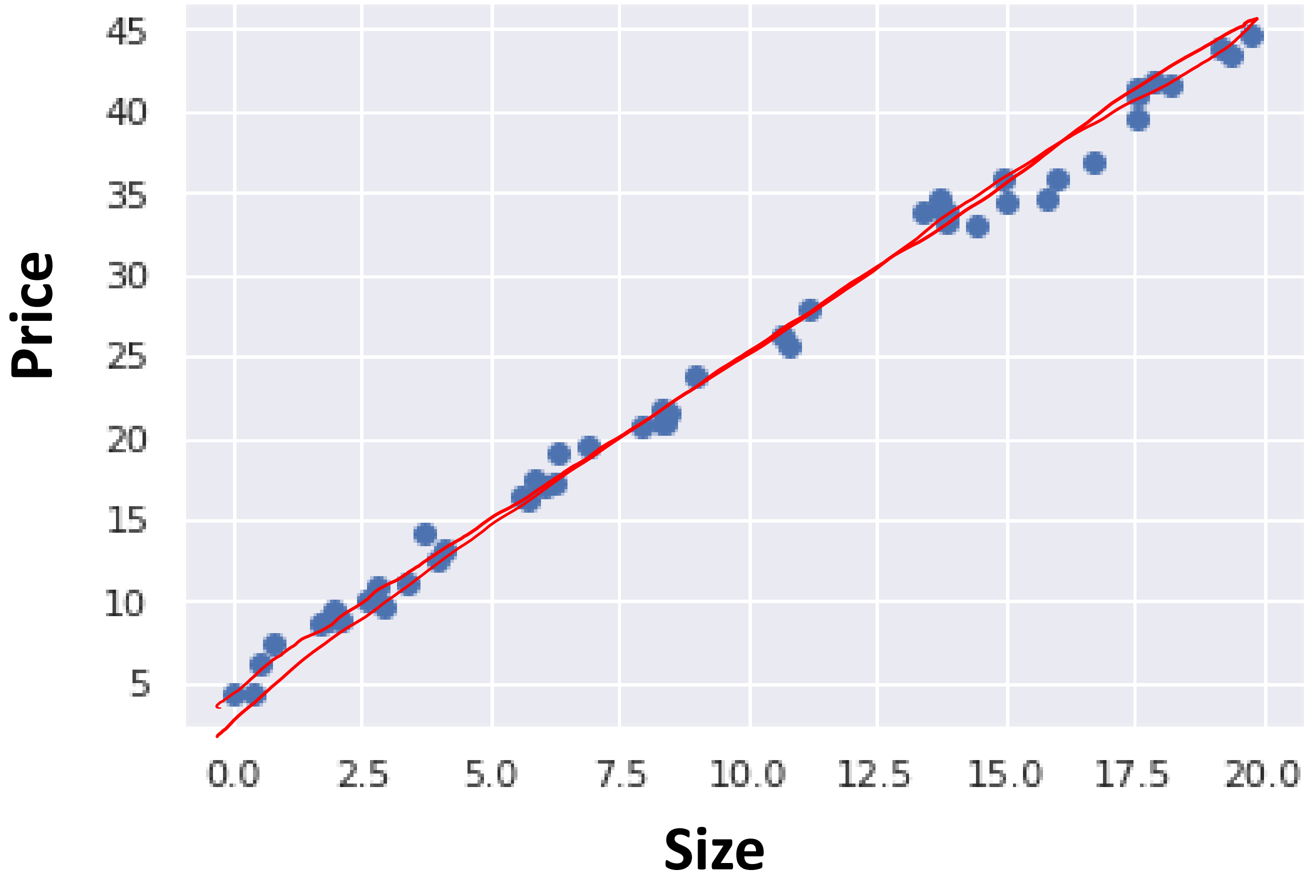
Dataset and Notations

- Notations
- m = Number of Training Examples
- **x** = Input Variable/ Features
- **y** = Output Variable / Target Value
- **(x,y)** is one training example
- **($x^{(i)}$,$y^{(i)}$)** is i^{th} training example
- **($x^{(1)}$,$y^{(1)}$) = (8.3, 20.99)**

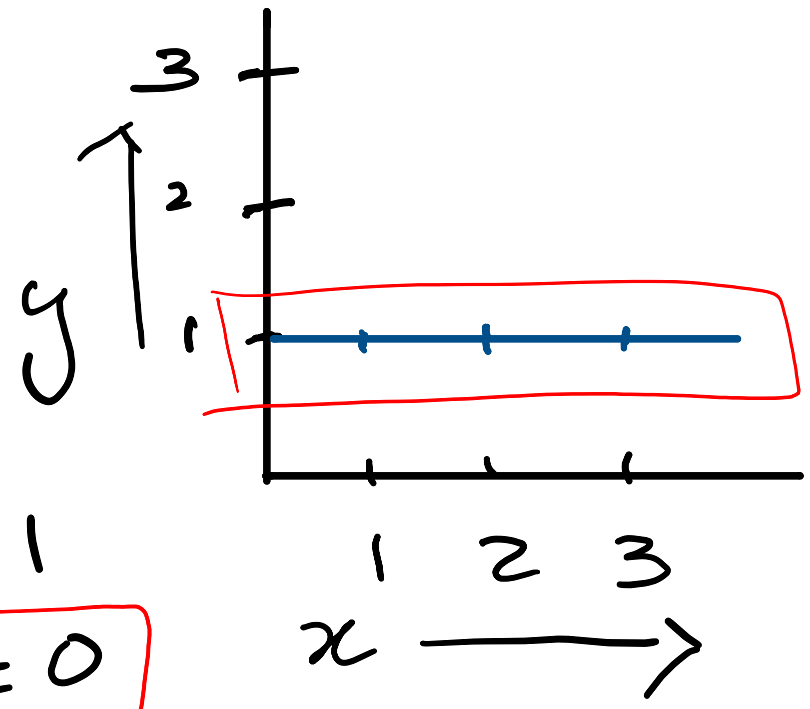
Input Data (x)	Correct Answer (y)
8.3	20.99
14.4	32.89
6.05	17.1
..	..

Linear Regression with One Variable

- This is like the equation of a straight line.
- We give $h\theta(x)$ values for θ_0 and θ_1 to get our estimated output y'.
- We are trying to create a function that will map out input data to our output data.

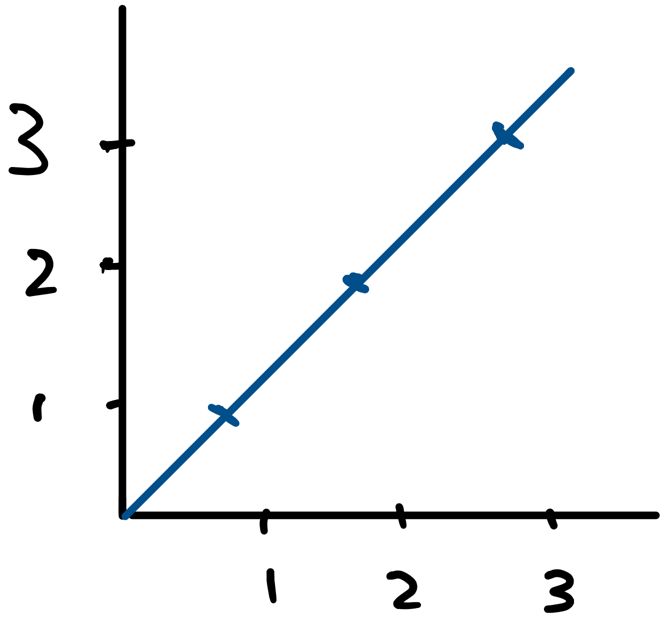


Linear Functions With Varying Values of ϴ



$\theta_0 = 1$

$\boxed{\theta_1 = 0}$



$\boxed{\theta_0 = 0}$

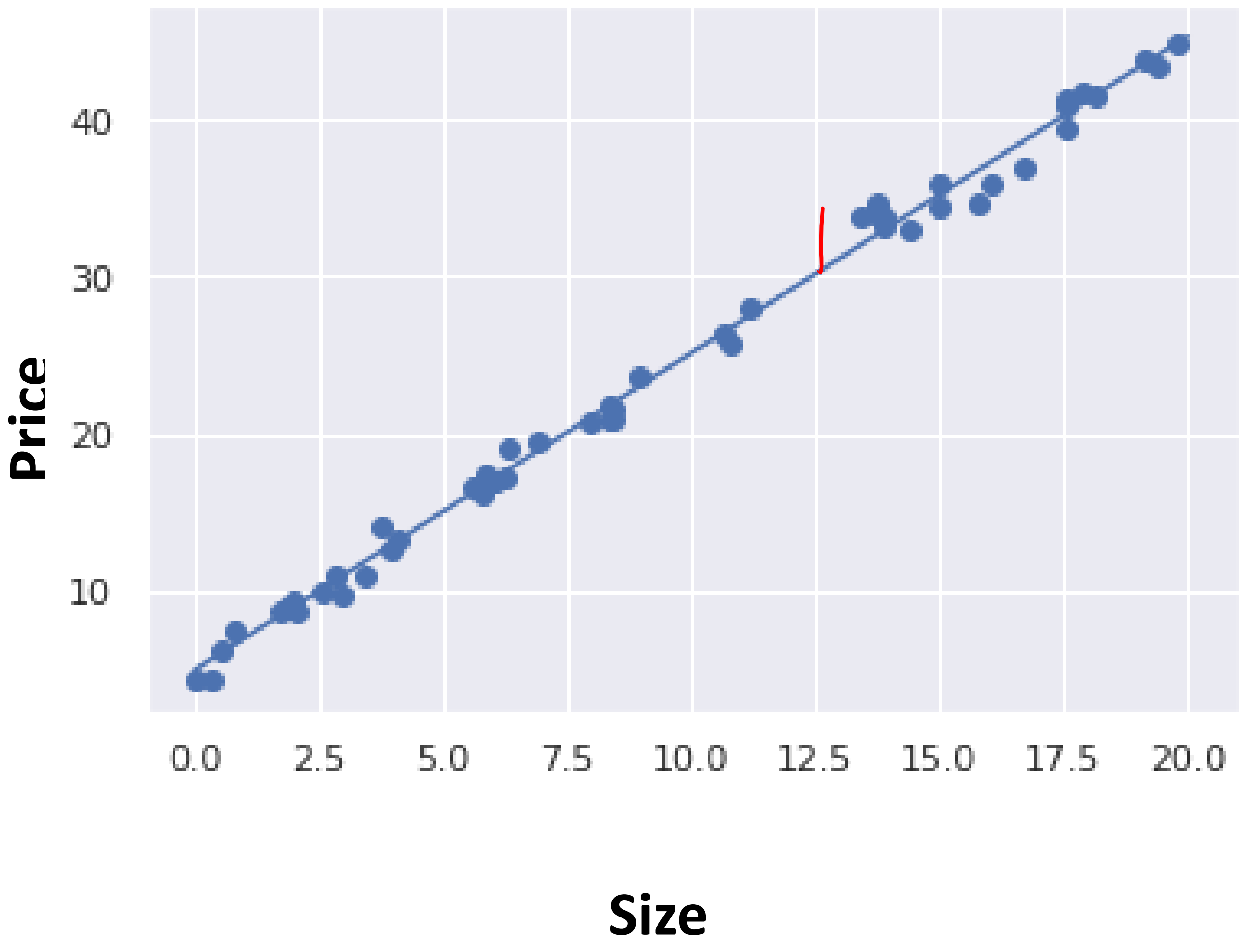
$\theta_1 = 1$

Linear Regression with One Variable

$$\overset{}{y'} = h\theta(x) = \overset{5}{\theta_0} + \overset{2\times 5}{\theta_1 x}$$

- Intercept: $\theta_0 = 5$
- Slope: $\theta_1 = 2$

Input Data (x)	Correct Answer (y)
5	15
10	25
15	35
..	..



Cost Function

- **Hypothesis Function**

$$y' = h\vartheta(x) = \vartheta_0 + \vartheta_1 x$$

- **Cost function** (to measure the performance of hypothesis function)

$$J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^{m} (y'^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^{m} (h\vartheta(x^{(i)}) - y^{(i)})^2$$

Cost Function

Input Data (x)	Correct Answer (y)	Estimated Answer	Error
8.3	20.99	21.6	-0.61
14.4	32.89	33.8	-0.81
6.05	17.1	17.08	0.02
..	..	..	..

$$Mean\ Square\ Error\ (MSE) = J(\vartheta_0, \vartheta_1) = \frac{1}{2m} \sum_{i=1}^{n} (y^{(i)} - y'^{(i)})^2$$

Cost Function

- **Hypothesis Function**

$$y' = h\vartheta(x) = \vartheta_0 + \vartheta_1 x$$

- **Cost function** (to measure the performance of hypothesis function)

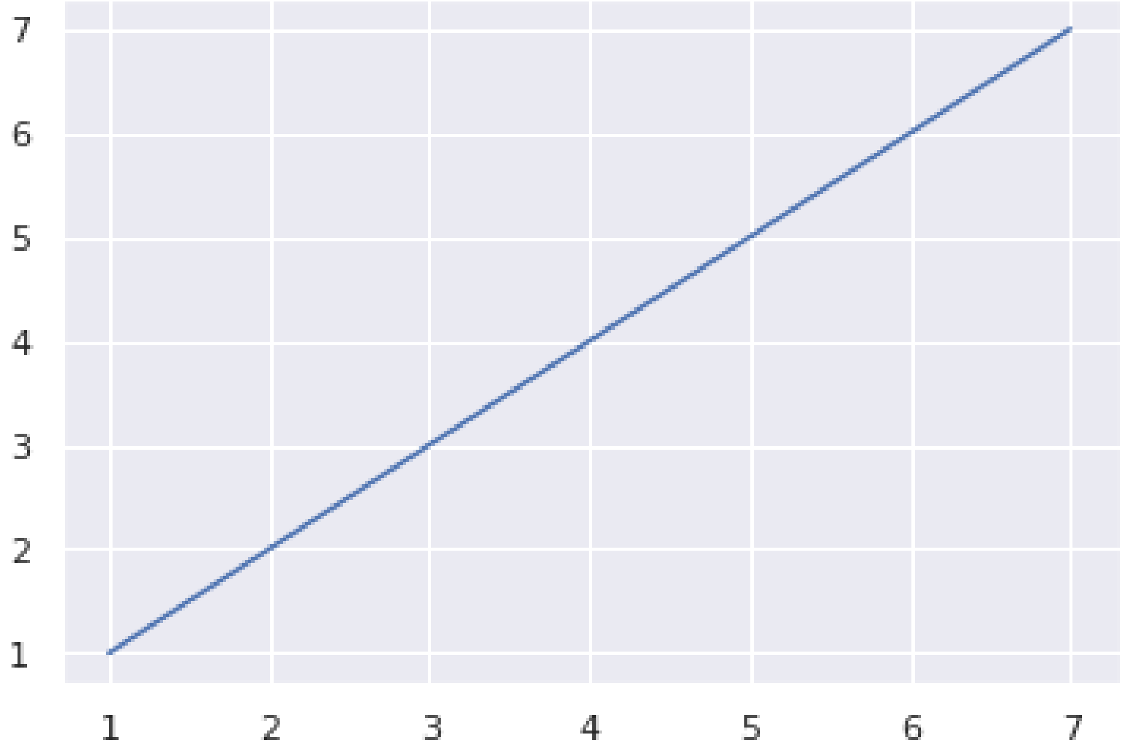
$$J(\vartheta_{0,}\vartheta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\ y'^{\,(i)}\ -\ y^{\,(i)}\ )^2 = \frac{1}{2m}\sum_{i=1}^{m}(\ h\vartheta(x^{\,(i)})\ -\ y^{\,(i)}\ )^2$$

- **Objective:**

$$\min_{\vartheta_{0,}\vartheta_1} \mathrm{J}(\vartheta_{0,}\vartheta_1)$$

Cost Function - Example

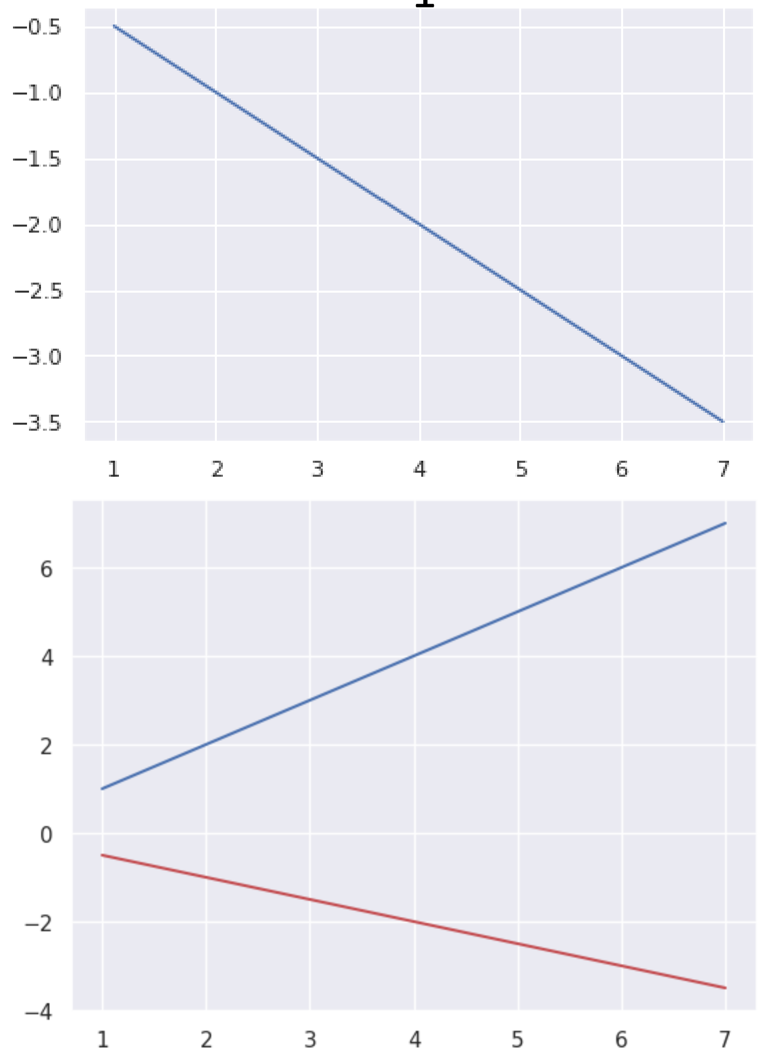
- **x** = [1, 2, 3, 4, 5, 6, 7]
- **y** = [1, 2, 3, 4, 5, 6, 7]
- $y' = h\theta(x) = \theta_0 + \theta_1 x$
- Assume $\theta_0 = 0$
- So, $y' = h\theta(x) = \theta_1 x$



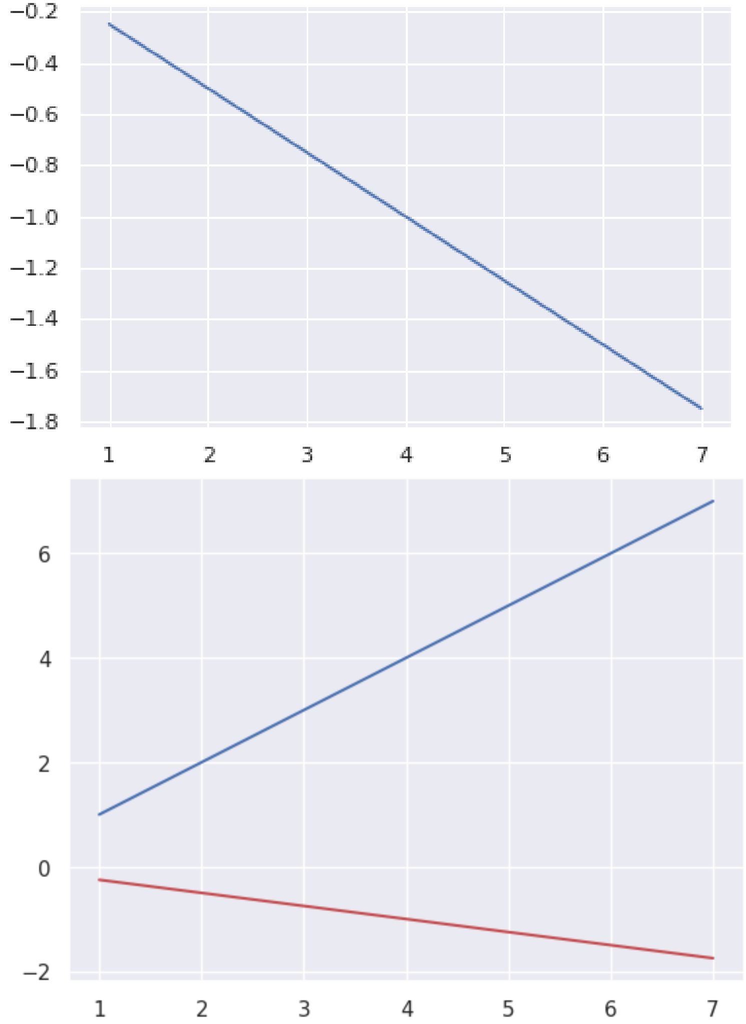
θ_1 = [-0.5, -0.25, 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25]

Cost Function - Example

- $\vartheta_1 = -0.5$, $J(\vartheta_1) = 45.0$

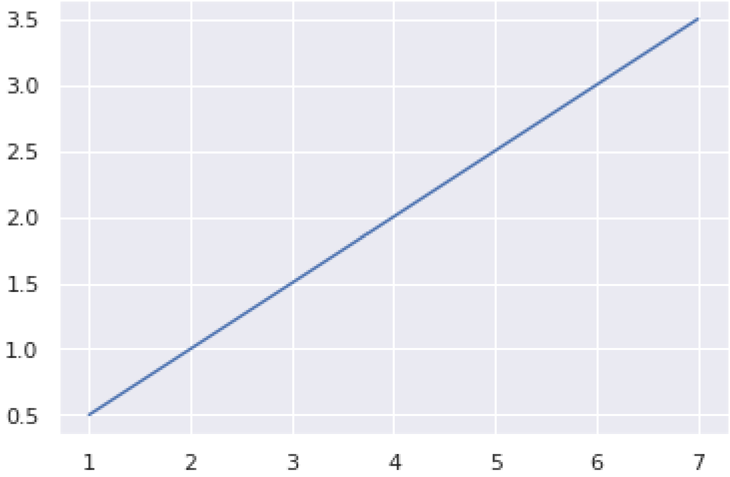


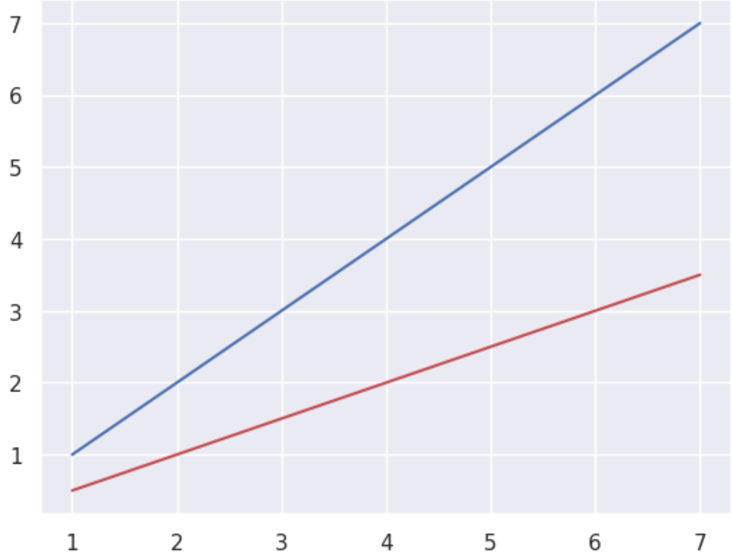
- $\vartheta_1 = -0.25$, $J(\vartheta_1) = 31.25$



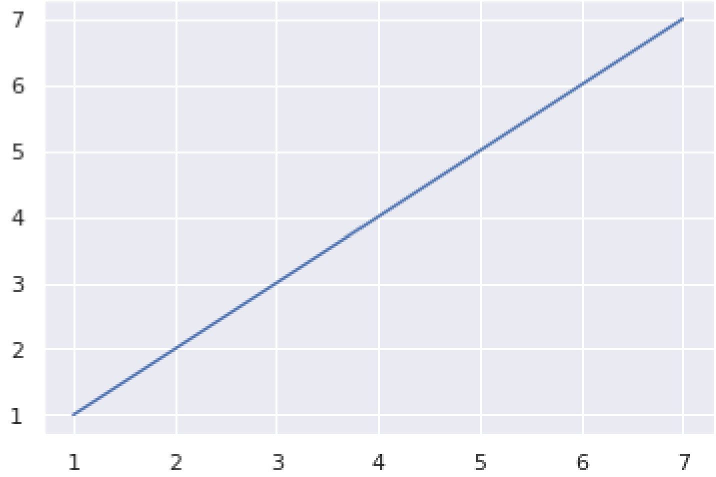
Cost Function - Example

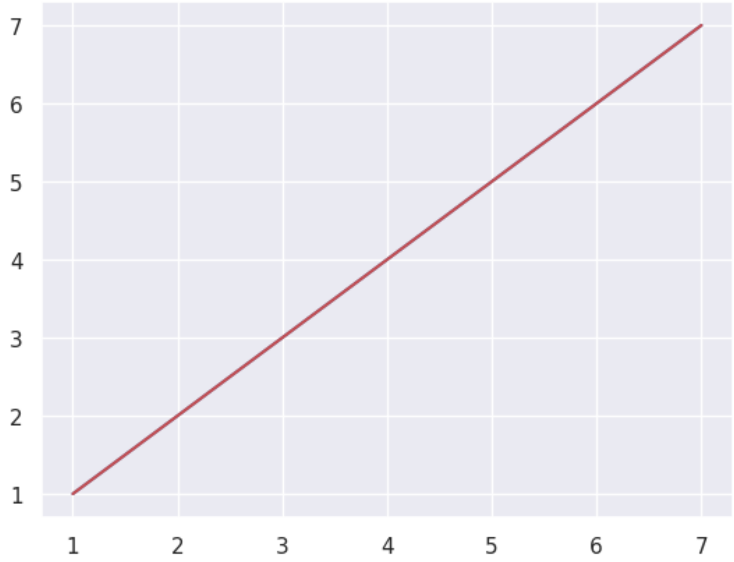
- ϑ_1 `= 0.5,` J(ϑ_1)= `5.0`





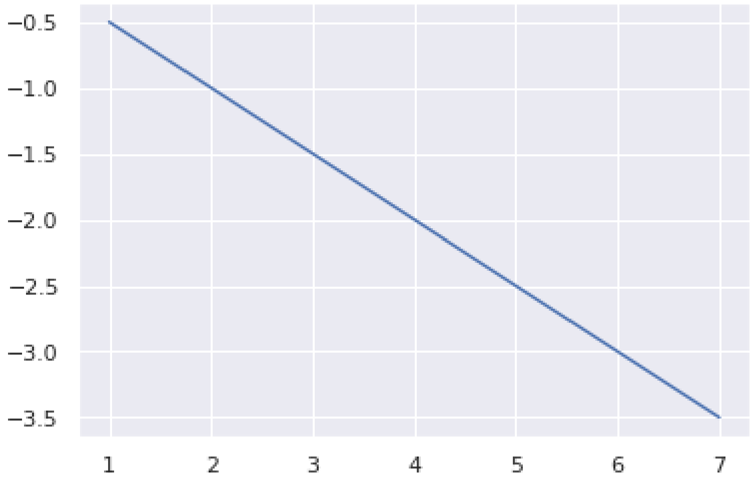
- ϑ_1 `= 1.0,` J(ϑ_1)= `0.0`



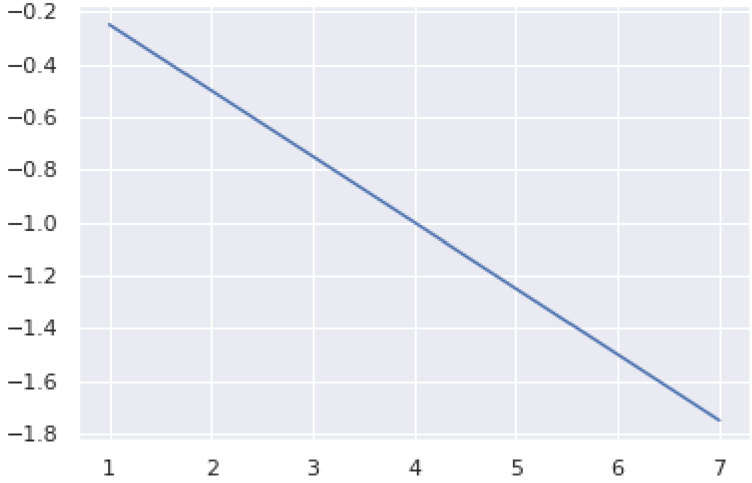


Cost Function - Example

- ϑ_1 = 1.5, J(ϑ_1)= 5.0

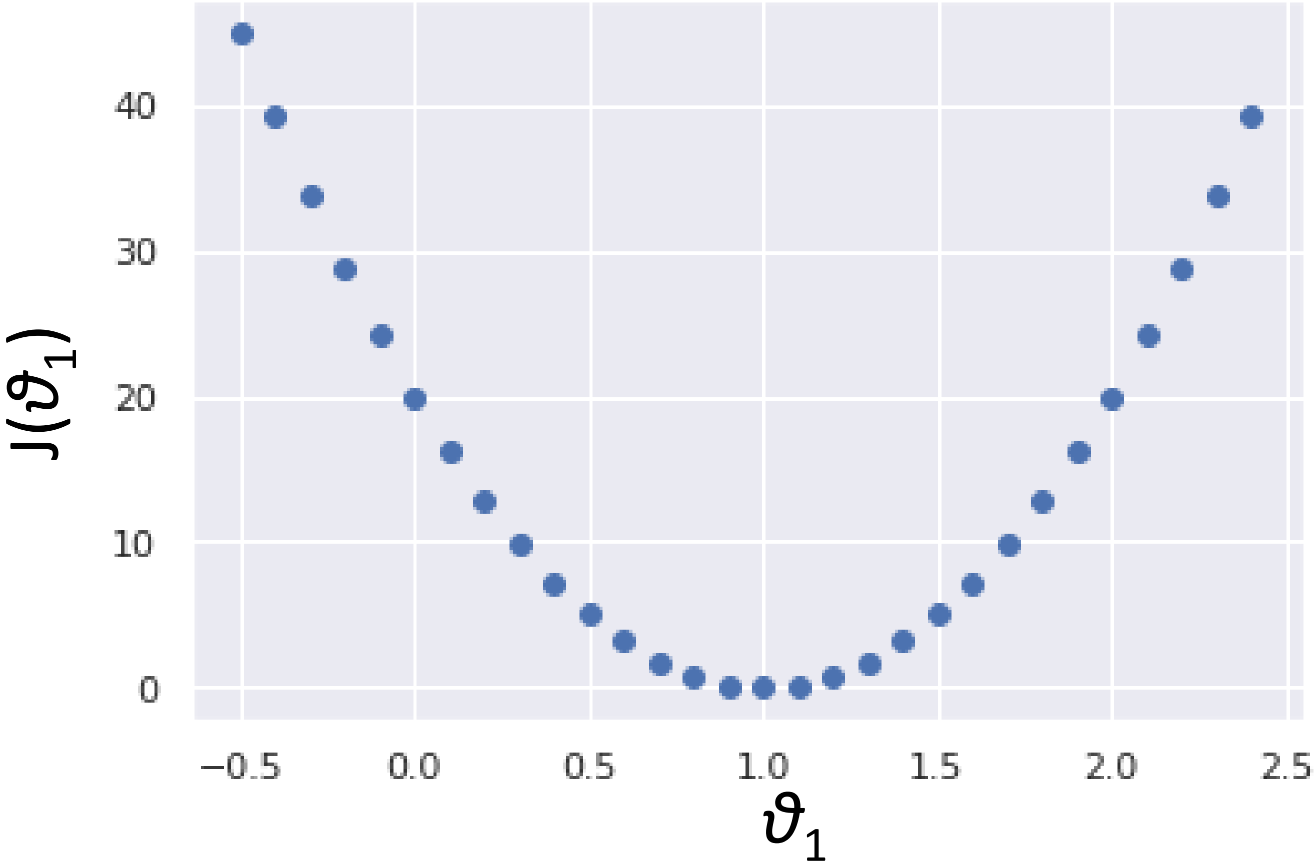


- ϑ_1 = 2.0, J(ϑ_1)= 20.0

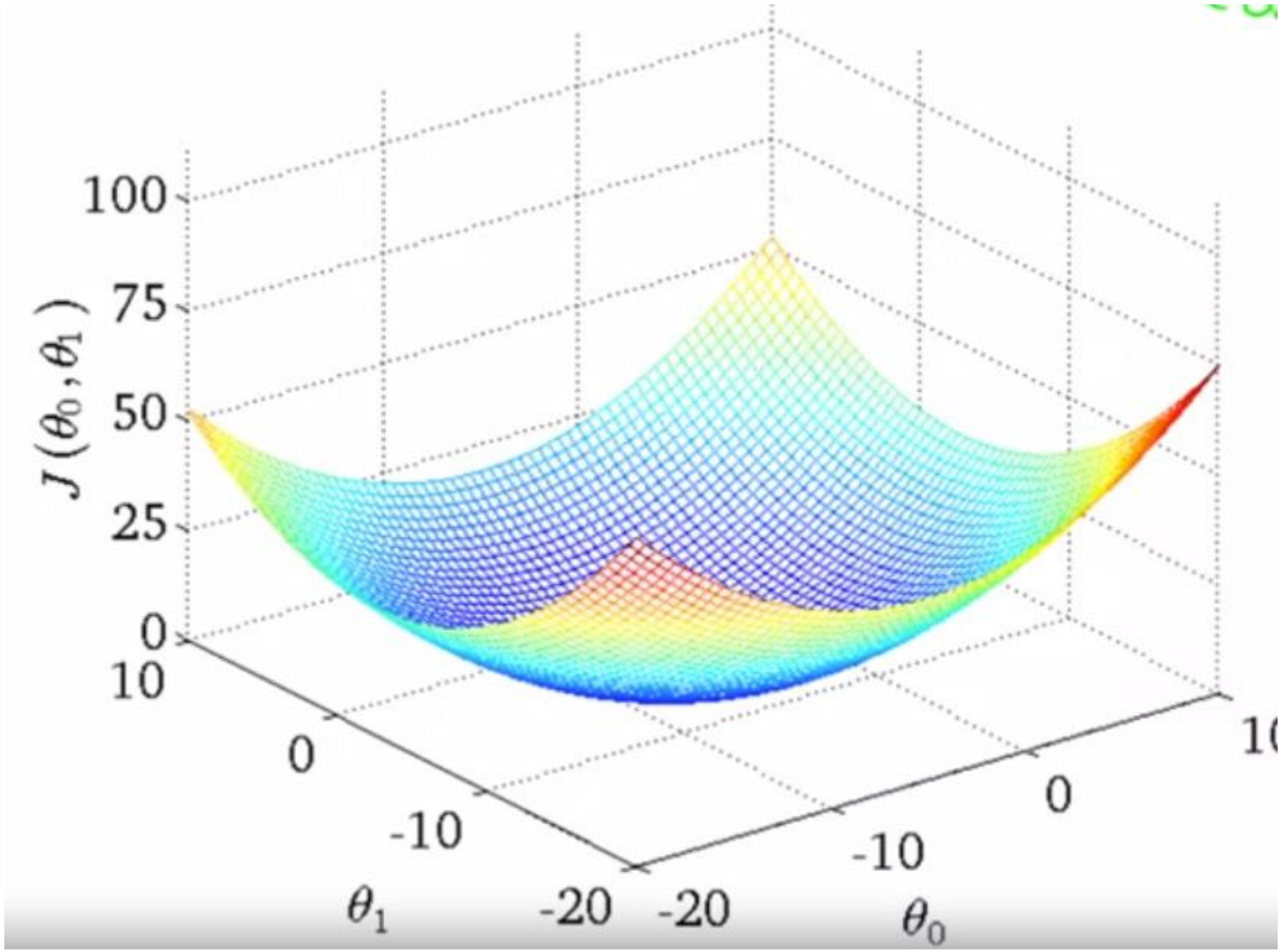


Graph of J(ϑ_1)

- θ_1 = [-0.5, -0.25, 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25]
- $J(\theta_1)$ =[45.0, 31.25, 20.0, 11.25, 5.0, 1.25, 0.0, 1.25, 5.0, 11.25, 20.0, 31.25]

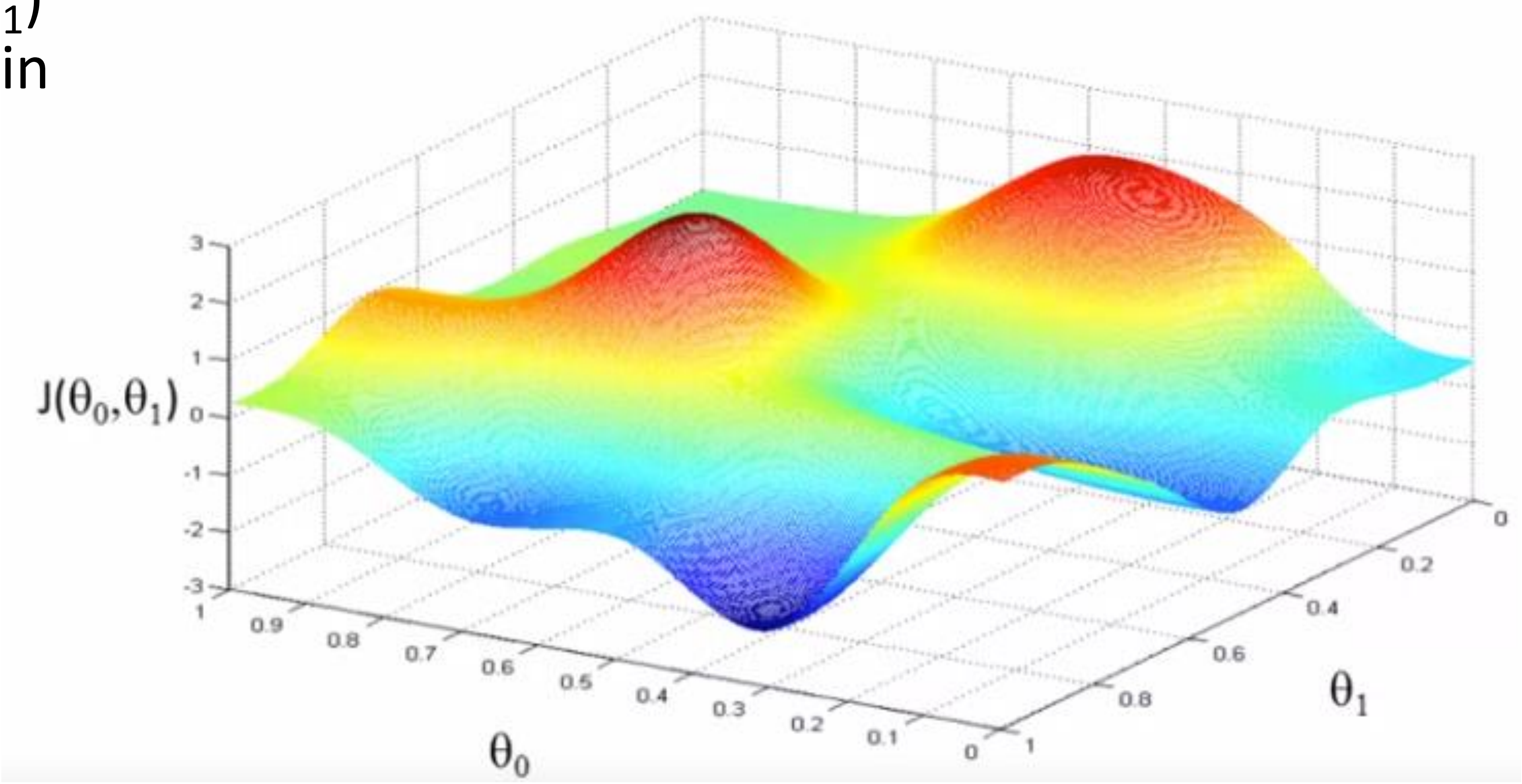


Graph of J($\vartheta_{0}, \vartheta_1$)

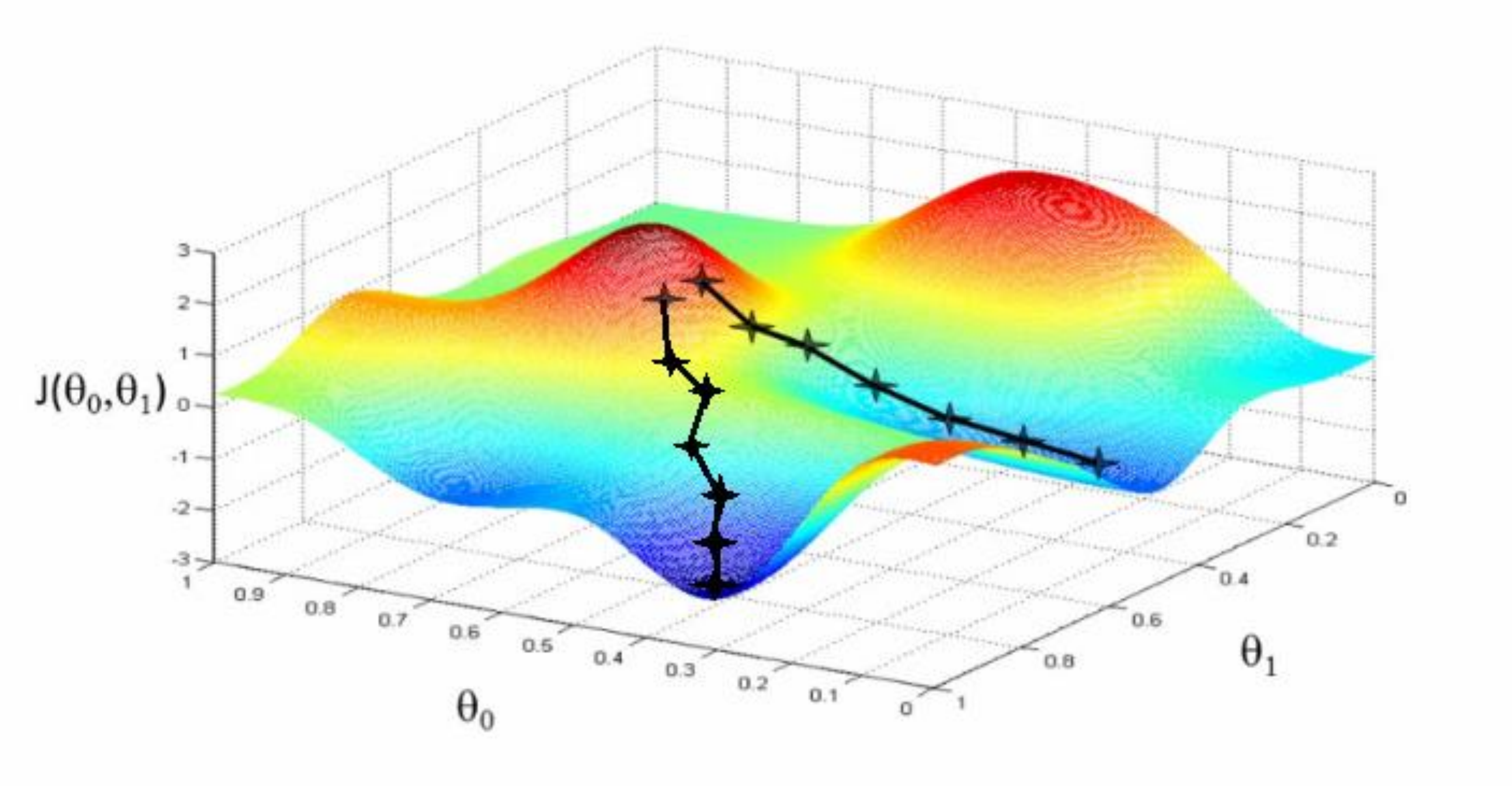


Gradient Descent Algorithm

- We have J(ϑ_0, ϑ_1) and we want min J(ϑ_0, ϑ_1)



Solving Minimization Problem



Derivatives

$f(x) = 4x$

$f(x) = x^3$

$f(x) = (x + 2)^4$

$f(x,y) = (3x + 2y + 2)^2$

Gradient Descent Algorithm

- $\vartheta_j \coloneqq \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1)$ (for j = 0 and j = 1)
- $\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1)$ is a partial derivative term
- Alpha is learning rate
- Simultaneous Update
- temp0 = $\vartheta_0 - \alpha \frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1)$
- temp1 = $\vartheta_1 - \alpha \frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1)$
- $\vartheta_0 \coloneqq$ temp0
- $\vartheta_1 \coloneqq$ temp1

Linear Regression with Gradient Descent

$$\vartheta_j := \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1)$$

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(h\vartheta(x^{(i)}) - y^{(i)} \right)^2\right)$$

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right)^2\right)$$

Linear Regression with Gradient Descent

$$\frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right)^2\right)$$

$$\frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_0} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right)^2\right)$$

$$\frac{\partial}{\partial \vartheta_0} J(\vartheta_0, \vartheta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right)$$

$$\frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1) = \frac{\partial}{\partial \vartheta_1} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right)^2\right)$$

$$\frac{\partial}{\partial \vartheta_1} J(\vartheta_0, \vartheta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)} \right) x^{(i)} \right)$$

Linear Regression with Gradient Descent

- Repeat until **converge**

$$\vartheta_0 := \vartheta_0 - \alpha \left(\frac{1}{m}\sum_{i=1}^{m}\left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)}\right)\right)$$

$$\vartheta_1 := \vartheta_1 - \alpha \left(\frac{1}{m}\sum_{i=1}^{m}\left(\vartheta_0 + \vartheta_1 x^{(i)} - y^{(i)}\right) x^{(i)}\right)$$

- Simultaneous update

Linear Regression with Multiple Features

- **Hypothesis Function**

$y' = h\vartheta(x) = \vartheta_0 + \vartheta_1 x_1 + \vartheta_1 x_1 + \vartheta_2 x_2 + \ldots + \vartheta_n x_n$

$y' = h\vartheta(x) = \vartheta_1 x_0 + \vartheta_1 x_1 + \vartheta_1 x_1 + \vartheta_2 x_2 + \ldots + \vartheta_n x_n$ *where* $x_1 = 1$

$y' = h\vartheta(x) = \theta^T X$

Simultaneous update

$$\vartheta_j \coloneqq \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\vartheta_0, \vartheta_1, \ldots, \vartheta_n)$$

$$\vartheta_j \coloneqq \vartheta_j - \alpha \frac{\partial}{\partial \vartheta_j} J(\theta)$$

What is Python?

- High level programming language
- First released in 1991 by Guido van Rossum
- Object-oriented, imperative, functional and procedural
- Supported by many operating systems
- Commonly used for computation in the field of artificial intelligence and machine learning

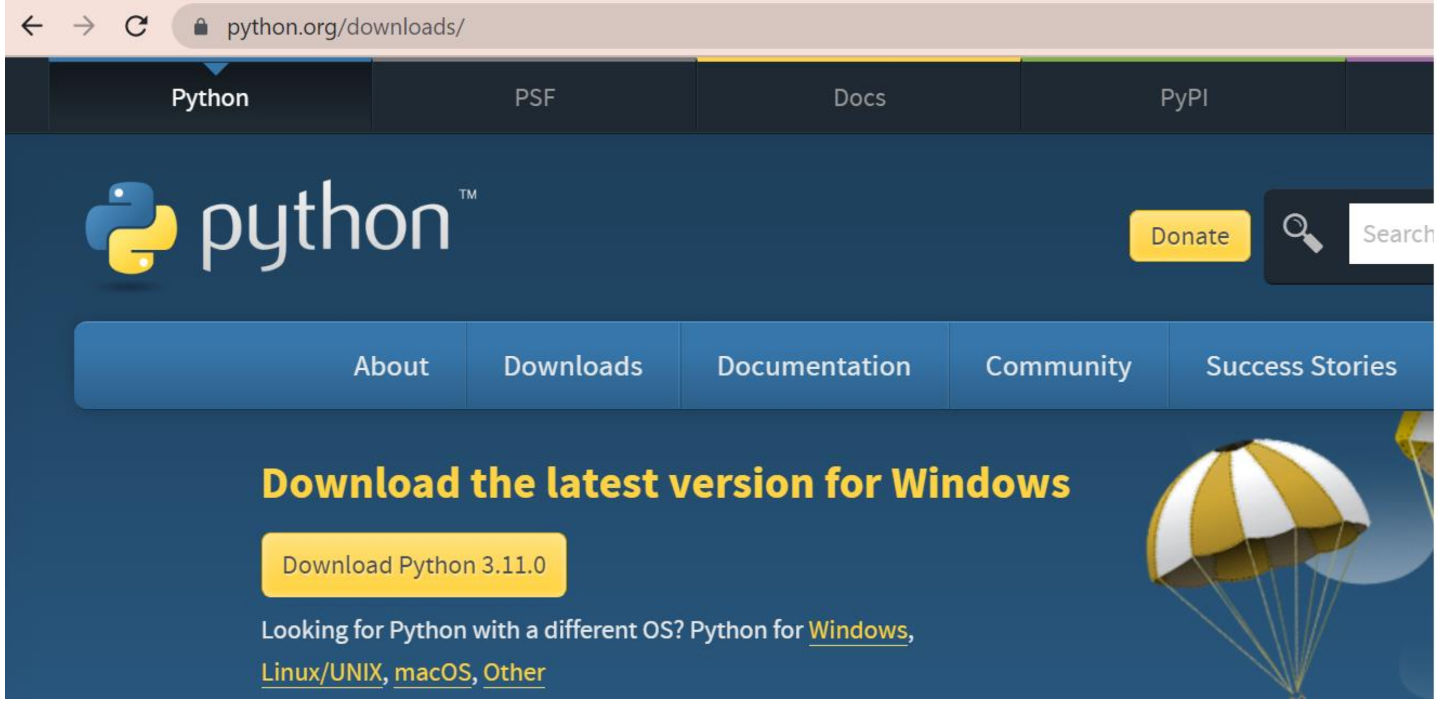
Why Python?

- Wide scientific community:
  - Many different helpful libraries
  - Open source frameworks and tools
  - Python Package Index (PyPI): 433,519
- Easy to learn/use
- Simplicity and consistency,
- High-level language: ideas can be implemented quickly
- Flexibility,
- Platform independence

Installation

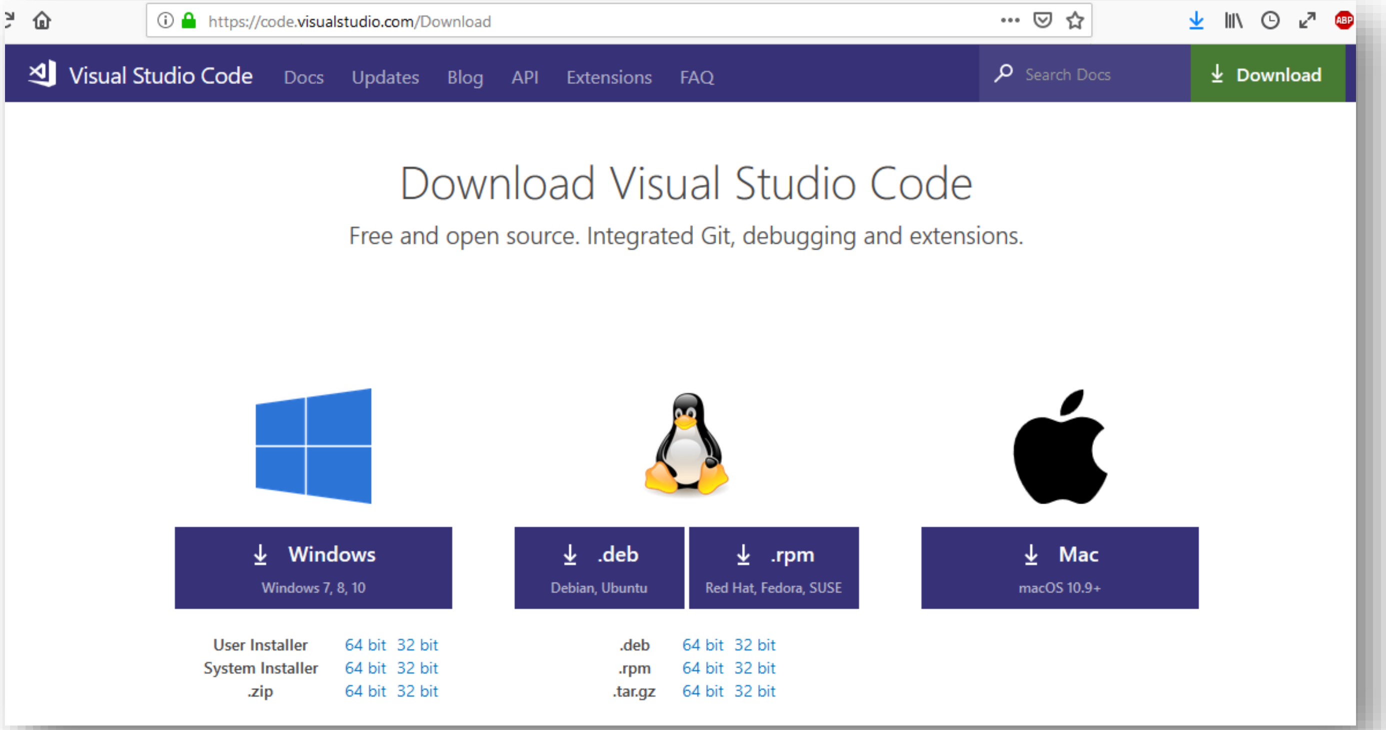
- **Windows**: visit https://www.python.org/downloads/ to download Python (recommended version: 3.9 or above)
- **Linux**: run the following command in terminal

```
sudo apt-get install python3.9
```



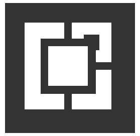
Installation IDE

- The choice of the IDE is up to you. I nevertheless recommend **VS Code** and will provide instructions for this IDE.
  - To download VS Code, visit https://code.visualstudio.com/Download



Setup VS Code

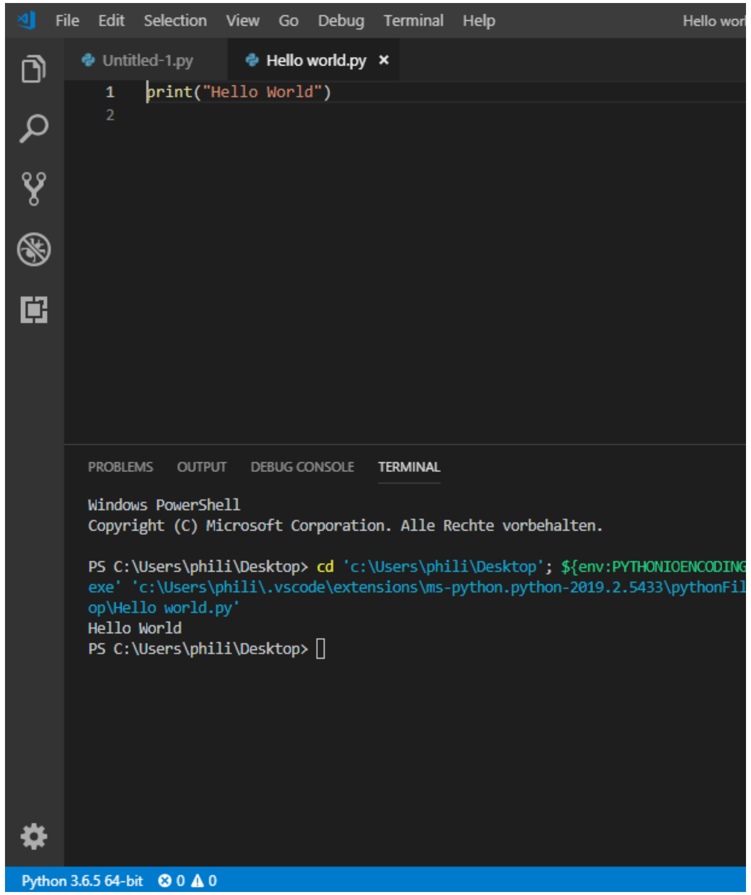
- Open the „Extensions“ view by:
  - Selecting „View“ -> „Extensions“
  - CTRL + Shift + X



- Search and install „Python“

VS Code (Python script)

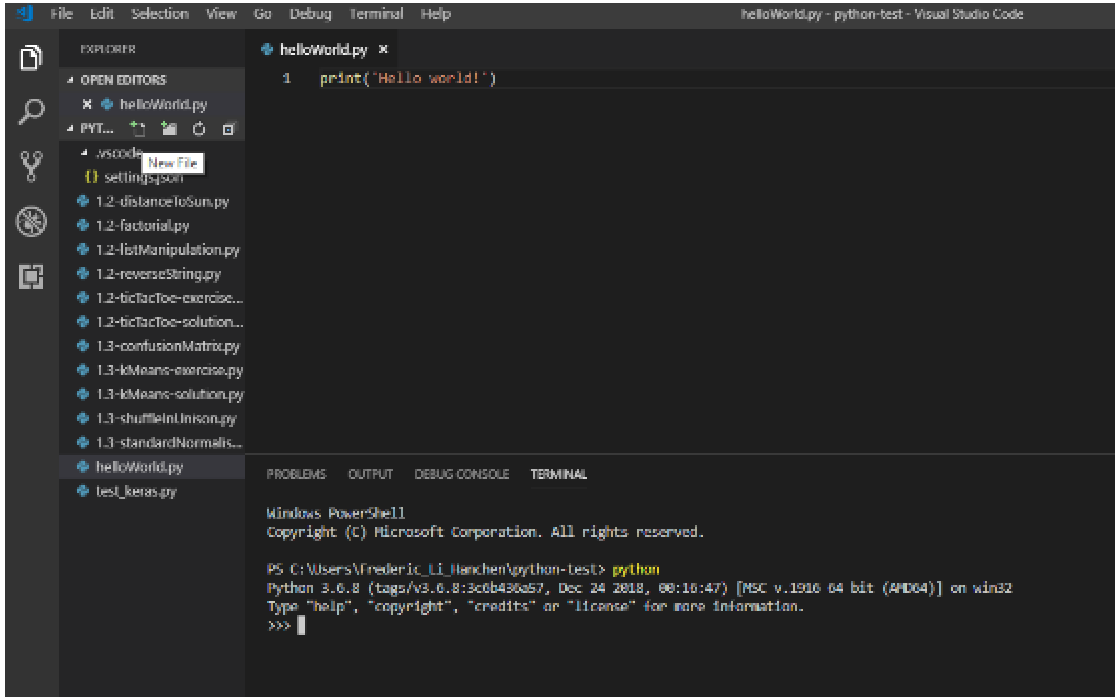
- Create a new file (File - New File)
- Save the file (File-> Save As...)
- Make sure it is of type python (.py)
- Scripts can be executed with F5 (debug mode), or right click + “Run Python file in terminal”
- To check your installation: print “Hello world!”



How to run Python code?

- Two main ways to execute Python code:
  1. Start the **Python interpreter** by typing > python in command line

  2. Write a **Python script** (.py file) then execute it by typing > python name_of_the_script.py in command line



Python Identifiers

- A **Python identifier** is a name used to identify a variable, function, class, module or other object:
  - Starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
  - Case sensitive.
- When using a Python identifier, avoid:
  - Using a Python keyword.
  - Starting or ending the identifier with underscore (_) unless you have a specific reason to do so.
- Python is a **dynamically typed** language:
  - No need to declare any variable type.
  - The type of a variable can be changed in the same script.
  - E.g.:

```
aVariable = 0
…
aVariable = "Hello world!"
```

Python keywords

- Reserved words (cannot use them as constant or variable)

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Naming conventions

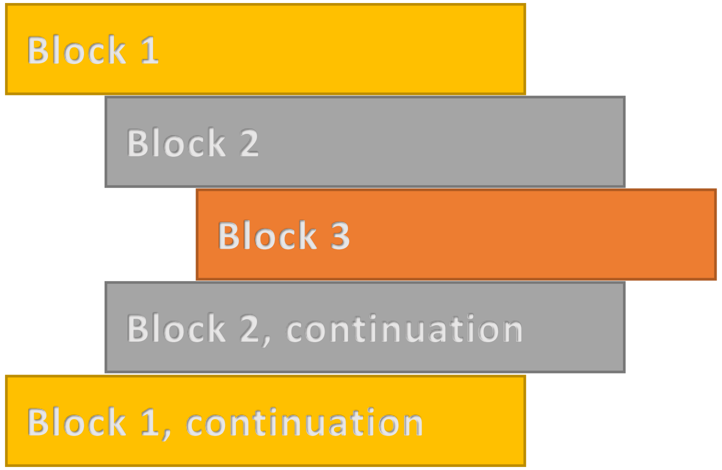
https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html

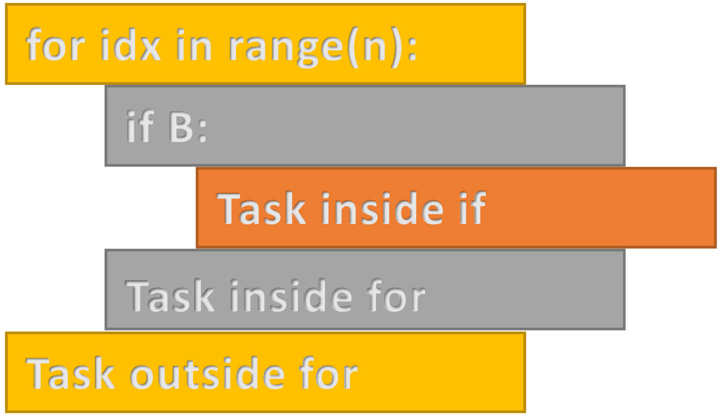
- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- Identifiers which start and end with two trailing underscores are language-defined special names. The most useful examples are:
  - `__init__` which refer to the class constructor
  - `__name__` and `__main__` which are used to define a main file:

```
if __name__ == "__main__":
    ……
```

Indentation

- In Python, no braces to indicate blocks of code
- Denoted by **line indentation**
- Number of spaces in the indentation up to you, but all statements within the block must be indented the same amount
  - Commonly used indentations: **tabs** or **4 spaces.**
- Notes:
  - mixing spaces and tabs together → error!
  - no need to use any character to denote an end of line (e.g. “;”)





Multi-Line Statements

- Statements in Python typically end with a new line
- Use of the line continuation character (\)

```
total = item_one + \
    item_two + \
    item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character

```
days = ['Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday']
```

Quotation

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals
- Start and end same type
- Triple quotes are used to span the string across multiple lines

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and
sentences."""
```

Comments

- A **hash sign (#)** that is not inside a string literal begins a comment -> whole line is commented (ignored by the interpreter)
- Can be used to comment multiple lines in a row
- **Triple-quoted string** is also ignored by Python interpreter and can be used as a multiline comments

```
#This is a comment
print("Hello World")

'''
This is
a
multirow comment
'''
```

Basic operations

- The operators +, -, * and / work just like in most other languages
- The standard comparison operators are written the same as in C:
  - < (less than)
  - > (greater than)
  - == (equal to)
  - <= (less than or equal to)
  - >= (greater than or equal to)
  - != (not equal to)
- Modulo operation: %
- Floor division: a // b

Assigning values to variables

- Variables do not need explicit declaration to reserve memory space
- No type needed
- Equal sign (=)
- delete entire variables: *del *variable name**

```python
# An integer assignment
counter = 100
# A floating point
miles = 1000.0
# A string
name = "John"
# one line
counter, miles, name = 100, 1000.0, "John"
# Assign single value to several variables
a = b = c = 1
# Delete a variable
del a
```

Standard data types

- Python has six standard data types
  - Numbers
  - Boolean
  - String
  - List
  - Tuple
  - Dictionary
- For numbers: integer, float, complex
- For Booleans: `True, False`
  - Note: in Python 3.x, Booleans inherit from integers, i.e. `True == 1` and `False == 0` will both return `True`.

Lists

- Items separated by commas and enclosed within square brackets ([])
  - Similar to arrays in C.
- **Indexing starts at 0**
- Items can be of different data type
- Items can deleted: *del*

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list) # Prints complete list
print(list[0]) # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:]) # Prints elements starting from 3rd element
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

IF statement

```
x = 1
if x < 0:
        x = 0
        print('Negative changed to zero')
elif x == 0:
        print('Zero')
elif x == 1:
        print('Single')
else:
        print('More')
```

- Form

  if *condition* :

    code block

- Elif/else are optional

Loops - For

- Iterates over the items of any sequence
- Iterate over a slice copy of the entire list with *list[:]*

```
list = ['element1', 'element2', 'element3']
for element in list:
        print(element)
```

- Iterate over a sequence of numbers: *range* - function

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

```
0
1
2
3
4
```

* *range()* returns an iterator, not a list!

Loops - While

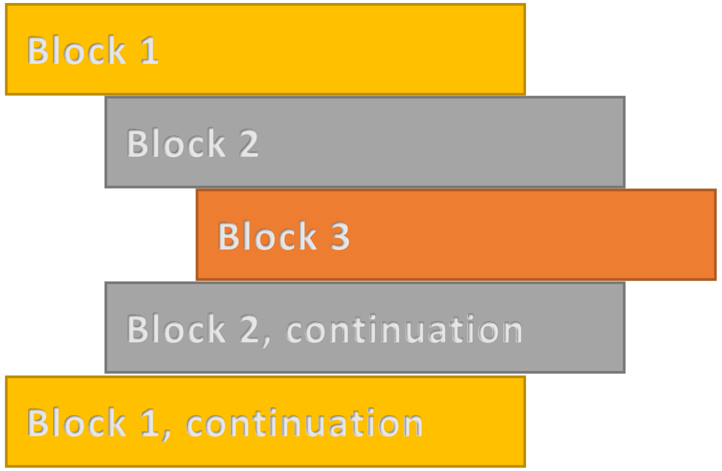
- The while statement is used for repeated execution as long as an expression is true
- Optional else clause: will be executed the first time the expression is false and the loop terminates

```
i = 0
while i<5:
        print(i)
        i += 1
else:
        print("Finished")
```

- The *break* statement, like in C, breaks out of the innermost enclosing *for* or *while* loop.

Conclusion

- Basic informal introduction to Python
- First basic Python exercises
- Next week: Data Structures and Functions



Exercise 1

- Write a program `printNegative` which prints all negative elements of an integer or float input list.

- Write a program `filterOdd` which takes a list of integers as input and returns a sub-list of the input containing only its odd elements (tip: the Python modulo operator is %; e.g. `7%2` returns `1`).

- Write a program `removeDuplicate` which takes a list of any type as input, and returns a list without any duplicate elements as output.

Exercise 2

Write a function `printReverseString` which takes a sentence under the string format as input (e.g. `"Hello world!"`) and prints the sentence with words in the reverse order (e.g. `"world! Hello"`).

Tips:

- A sentence is an ensemble of words separated by spaces
- The Python string methods `split` and `join` should be useful for this exercise

Exercise 3

Write a function `distanceToSun` which asks the user to input the name of a planet of our solar system in command line, and prints the distance between the planet and the Sun in kilometres.

Tips:

- Pick the online source of your choice to find distances between planets and the Sun.
- Asking for an user input in terminal can be done using the `userInput = input("text to display")` syntax.
- Remember to treat the case where the user input is invalid.
- Stopping a function can be done using `return`

What is Python?

- High level programming language
- First released in 1991 by Guido van Rossum
- Object-oriented, imperative, functional and procedural
- Supported by many operating systems
- Commonly used for computation in the field of artificial intelligence and machine learning

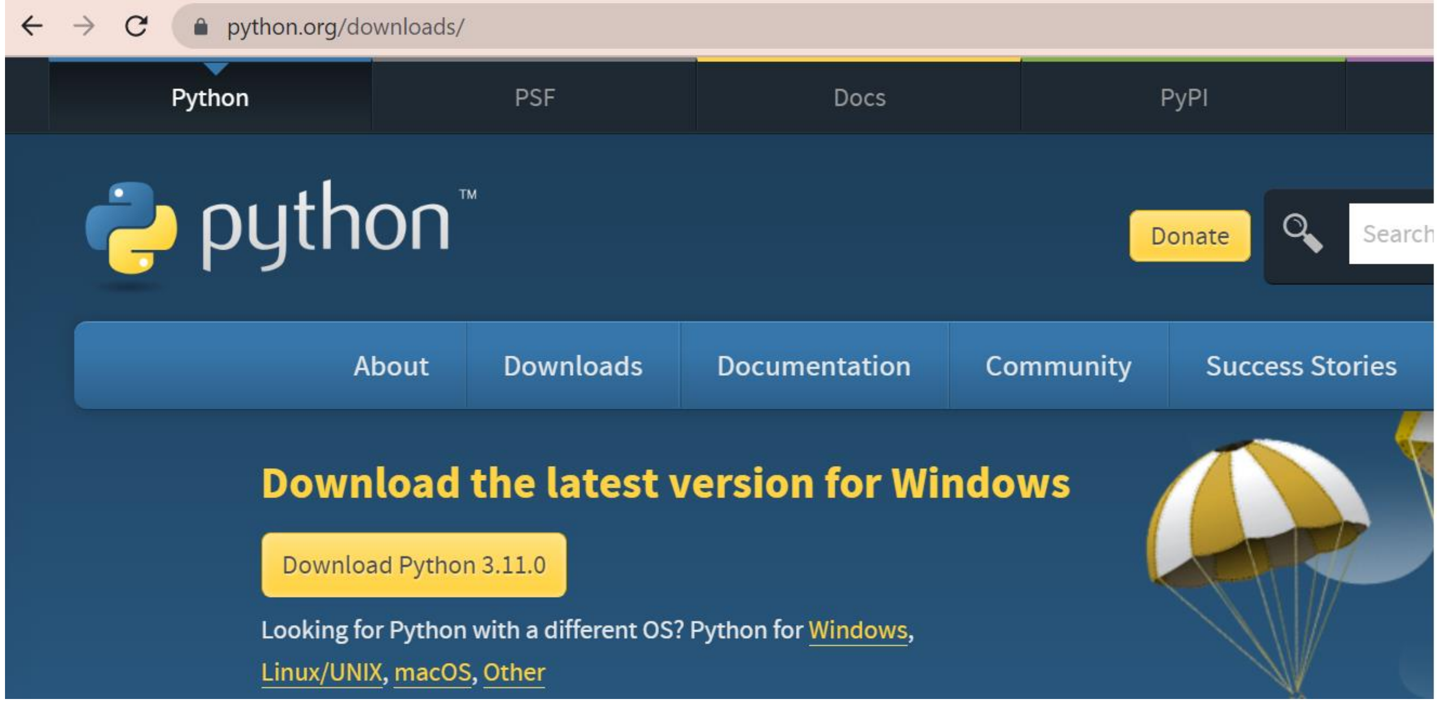
Why Python?

- Wide scientific community:
  - Many different helpful libraries
  - Open source frameworks and tools
  - Python Package Index (PyPI): 433,519
- Easy to learn/use
- Simplicity and consistency,
- High-level language: ideas can be implemented quickly
- Flexibility,
- Platform independence

Installation

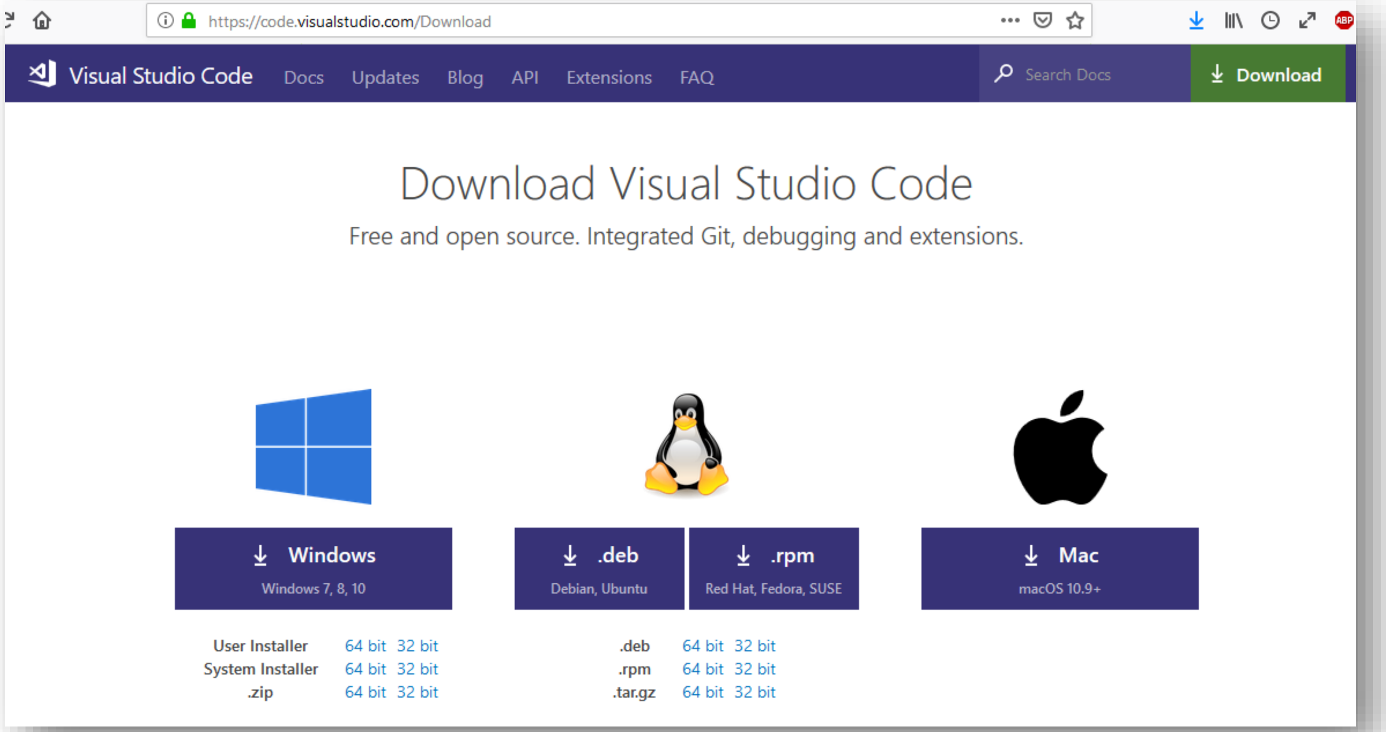
- **Windows**: visit https://www.python.org/downloads/ to download Python (recommended version: 3.9 or above)
- **Linux**: run the following command in terminal

```
sudo apt-get install python3.9
```



Installation IDE

- The choice of the IDE is up to you. I nevertheless recommend **VS Code** and will provide instructions for this IDE.
  - To download VS Code, visit https://code.visualstudio.com/Download

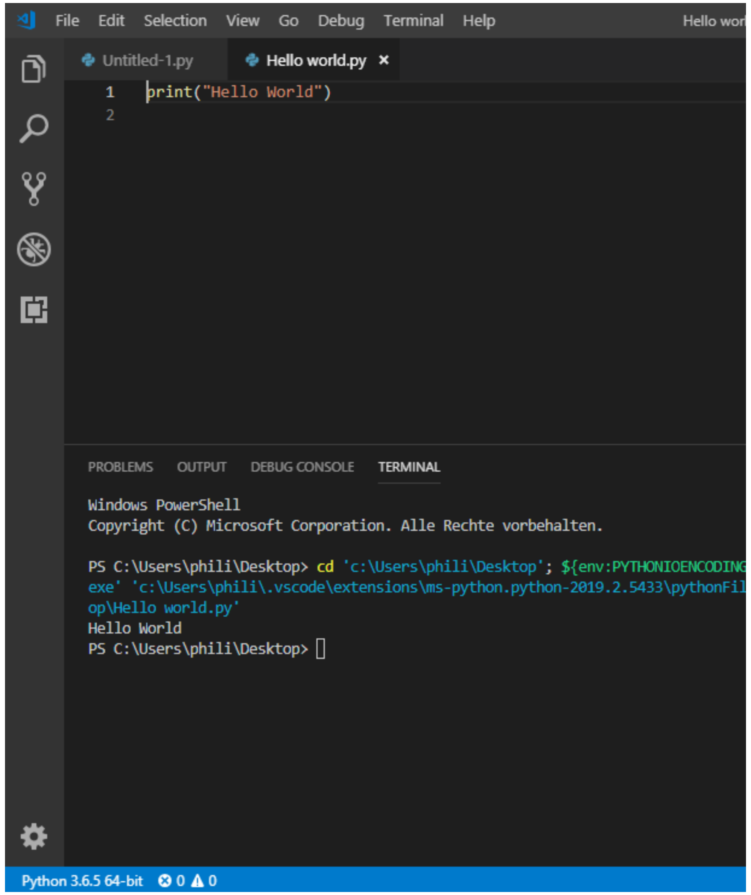


Setup VS Code

- Open the „Extensions“ view by:
  - Selecting „View“ -> „Extensions“
  - CTRL + Shift + X
- Search and install „Python“

VS Code (Python script)

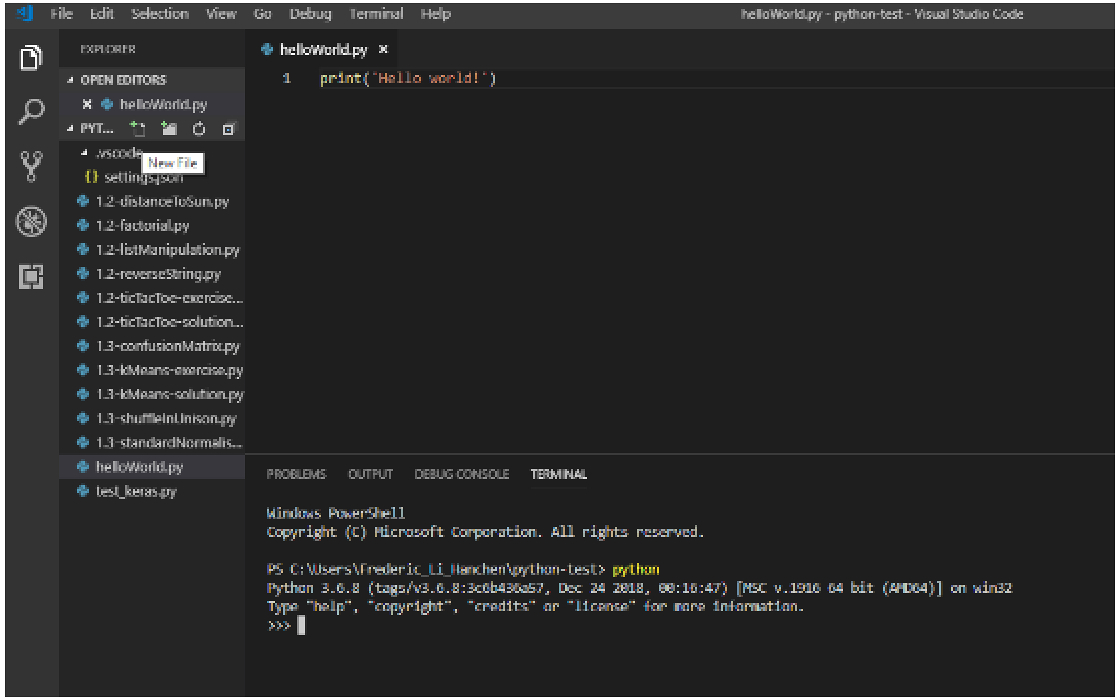
- Create a new file (File - New File)
- Save the file (File-> Save As…)
- Make sure it is of type python (.py)
- Scripts can be executed with F5 (debug mode), or right click + "Run Python file in terminal"
- To check your installation: print "Hello world!"



How to run Python code?

- Two main ways to execute Python code:
  1. Start the **Python interpreter** by typing > python in command line

  2. Write a **Python script** (.py file) then execute it by typing > python name_of_the_script.py in command line



Python Identifiers

- A **Python identifier** is a name used to identify a variable, function, class, module or other object:
  - Starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
  - Case sensitive.
- When using a Python identifier, avoid:
  - Using a Python keyword.
  - Starting or ending the identifier with underscore (_) unless you have a specific reason to do so.
- Python is a **dynamically typed** language:
  - No need to declare any variable type.
  - The type of a variable can be changed in the same script.
  - E.g.:

```
aVariable = 0
…
aVariable = “Hello world!”
```

Python keywords

- Reserved words (cannot use them as constant or variable)

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Naming conventions

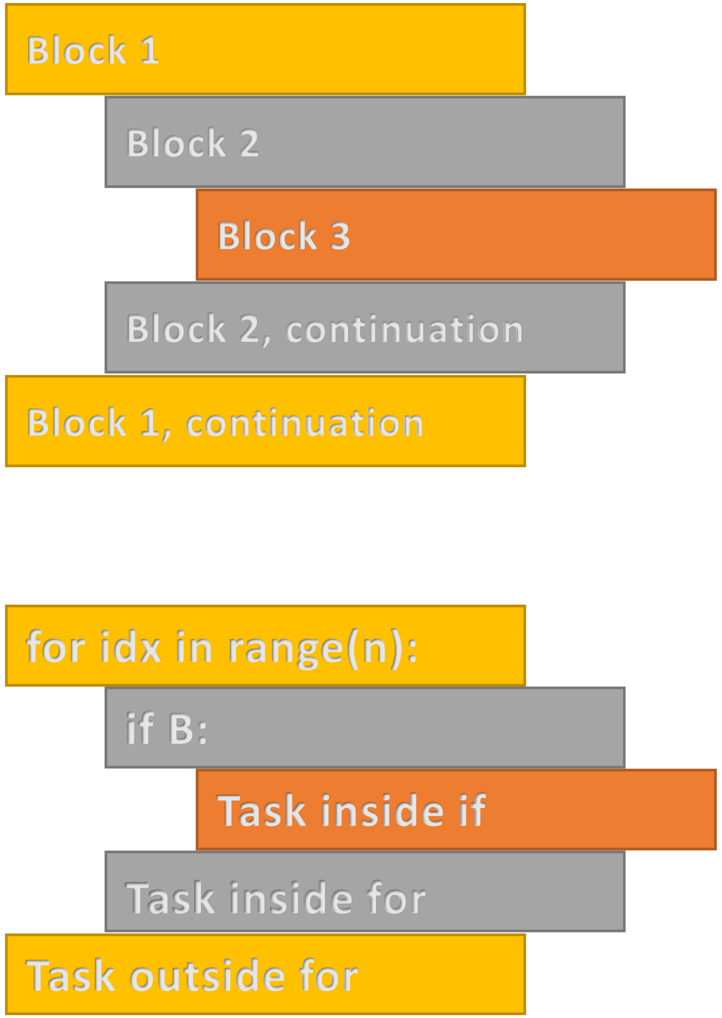
https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- Identifiers which start and end with two trailing underscores are language-defined special names. The most useful examples are:
  - `__init__` which refer to the class constructor
  - `__name__` and `__main__` which are used to define a main file:

```
if __name__ == “__main__”:
    ……
```

Indentation

- In Python, no braces to indicate blocks of code
- Denoted by **line indentation**
- Number of spaces in the indentation up to you, but all statements within the block must be indented the same amount
  - Commonly used indentations: **tabs** or **4 spaces.**
- Notes:
  - mixing spaces and tabs together → error!
  - no need to use any character to denote an end of line (e.g. “;”)



Multi-Line Statements

- Statements in Python typically end with a new line
- Use of the line continuation character (\)

```
total = item_one + \
    item_two + \
    item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character

```
days = ['Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday']
```

Quotation

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals
- Start and end same type
- Triple quotes are used to span the string across multiple lines

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and
sentences."""
```

Comments

- A **hash sign (#)** that is not inside a string literal begins a comment -> whole line is commented (ignored by the interpreter)
- Can be used to comment multiple lines in a row
- **Triple-quoted string** is also ignored by Python interpreter and can be used as a multiline comments

```python
#This is a comment
print("Hello World")

'''
This is
a
multirow comment
'''
```

Basic operations

- The operators +, -, * and / work just like in most other languages
- The standard comparison operators are written the same as in C:
  - < (less than)
  - > (greater than)
  - == (equal to)
  - <= (less than or equal to)
  - >= (greater than or equal to)
  - != (not equal to)
- Modulo operation: %
- Floor division: a // b

Assigning values to variables

- Variables do not need explicit declaration to reserve memory space
- No type needed
- Equal sign (=)
- delete entire variables: *del *variable name**

```
# An integer assignment
counter = 100
# A floating point
miles = 1000.0
# A string
name = "John"
# one line
counter, miles, name = 100, 1000.0, "John"
# Assign single value to several variables
a = b = c = 1
# Delete a variable
del a
```

Standard data types

- Python has six standard data types
  - Numbers
  - Boolean
  - String
  - List
  - Tuple
  - Dictionary
- For numbers: integer, float, complex
- For Booleans: `True, False`
  - Note: in Python 3.x, Booleans inherit from integers, i.e. `True == 1` and `False == 0` will both return `True`.

Lists

- Items separated by commas and enclosed within square brackets ([])
  - Similar to arrays in C.
- **Indexing starts at 0**
- Items can be of different data type
- Items can deleted: *del*

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list) # Prints complete list
print(list[0]) # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:]) # Prints elements starting from 3rd element
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

IF statement

```
x = 1
if x < 0:
        x = 0
        print('Negative changed to zero')
elif x == 0:
        print('Zero')
elif x == 1:
        print('Single')
else:
        print('More')
```

- Form

  if *condition* :

  code block

- Elif/else are optional

Loops - For

- Iterates over the items of any sequence
- Iterate over a slice copy of the entire list with *list[:]*

```
list = ['element1', 'element2', 'element3']
for element in list:
        print(element)
```

- Iterate over a sequence of numbers: *range* - function

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

```
0
1
2
3
4
```

* *range()* returns an iterator, not a list!

Loops - While

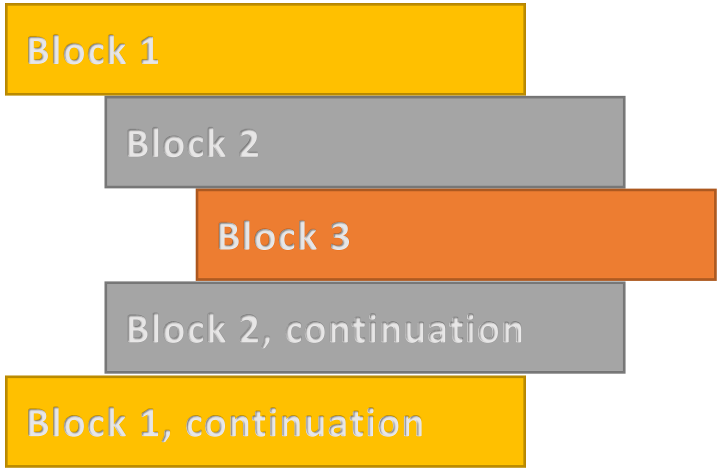
- The while statement is used for repeated execution as long as an expression is true
- Optional else clause: will be executed the first time the expression is false and the loop terminates

```
i = 0
while i<5:
        print(i)
        i += 1
else:
        print("Finished")
```

- The *break* statement, like in C, breaks out of the innermost enclosing *for* or *while* loop.

Conclusion

- Basic informal introduction to Python
- First basic Python exercises
- Next week: Data Structures and Functions



Exercise 1

- Write a program `printNegative` which prints all negative elements of an integer or float input list.

- Write a program `filterOdd` which takes a list of integers as input and returns a sub-list of the input containing only its odd elements (tip: the Python modulo operator is %; e.g. `7%2` returns `1`).

- Write a program `removeDuplicate` which takes a list of any type as input, and returns a list without any duplicate elements as output.

Exercise 2

Write a function `printReverseString` which takes a sentence under the string format as input (e.g. `"Hello world!"`) and prints the sentence with words in the reverse order (e.g. `"world! Hello"`).

Tips:

- A sentence is an ensemble of words separated by spaces
- The Python string methods `split` and `join` should be useful for this exercise

Exercise 3

Write a function `distanceToSun` which asks the user to input the name of a planet of our solar system in command line, and prints the distance between the planet and the Sun in kilometres.

Tips:

- Pick the online source of your choice to find distances between planets and the Sun.
- Asking for an user input in terminal can be done using the `userInput = input("text to display")` syntax.
- Remember to treat the case where the user input is invalid.
- Stopping a function can be done using `return`

Linear Algebra review (optional)

Matrices and vectors

Courtesy: Andrew Ng

Matrix: Rectangular array of numbers:

Dimension of matrix: number of rows x number of columns

Matrix Elements (entries of matrix)

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

$A_{ij} =$ "i, j entry" in the i^{th} row, j^{th} column.

Vector: An n x 1 matrix.

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$y_i = i^{th}$ element

1-indexed vs 0-indexed:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Linear Algebra review (optional)

Addition and scalar multiplication

Matrix Addition

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \end{bmatrix} =$$

Scalar Multiplication

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} / 4 =$$

Combination of Operands

$$3 \times \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix} /3$$

Linear Algebra review (optional)

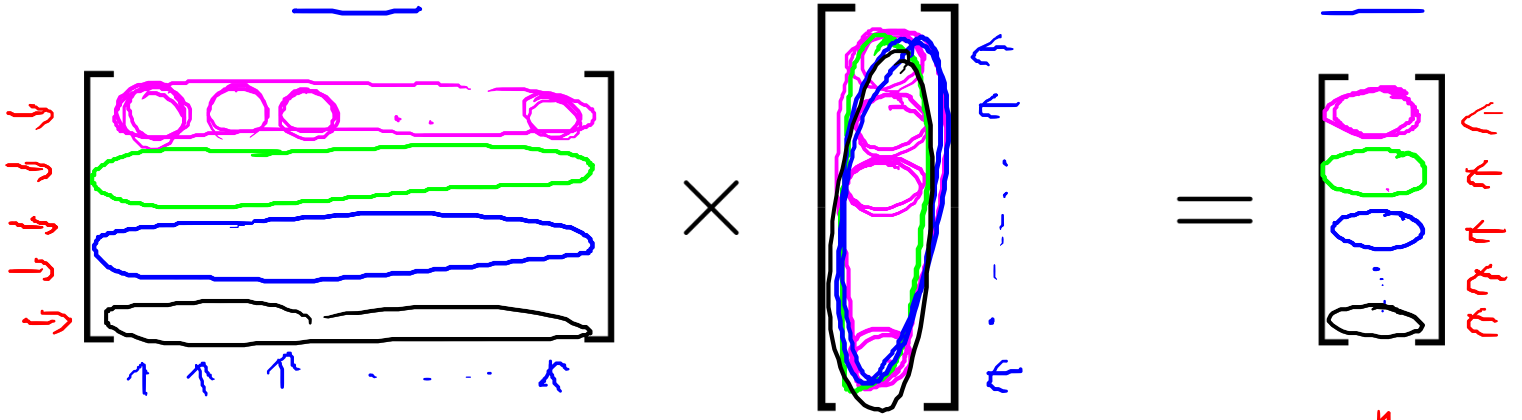
Matrix-vector multiplication

Example

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \end{bmatrix} =$$

Details:

$$A \times x = y$$



m x n matrix (m rows, n columns)

n x 1 matrix (n-dimensional vector)

m-dimensional vector

To get y_i, multiply A's i^{th} row with elements of vector x, and add them up.

Example

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -1 & -2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \boxed{14} \\ \boxed{13} \\ \boxed{-7} \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix}$$

$\boxed{3} \times 4$ $\quad$ 4×1 $\quad$ 3×1

$$1 \times 1 + 2 \times 3 + 1 \times 2 + 5 \times 1 = 14$$

$$0 \times 1 + 3 \times 3 + 0 \times 2 + 4 \times 1 = 13$$

$$-1 \times 1 + (-2) \times 3 + 0 \times 2 + 0 \times 1 = -7$$