Ulster University

COM738 - Data Science Foundations

Week 1

Module Co-Ordinator: Dr Priyanka Chaurasia

Contents

- Overview of Data Science
- Applications of Data Science
- Introduction to Python
- Practical experience-development environment & syntax

Ulster University

What is Data Science?

Ulster University

What is Data Science?

- The process of extracting knowledge or insights from data in various forms, either structured or unstructured

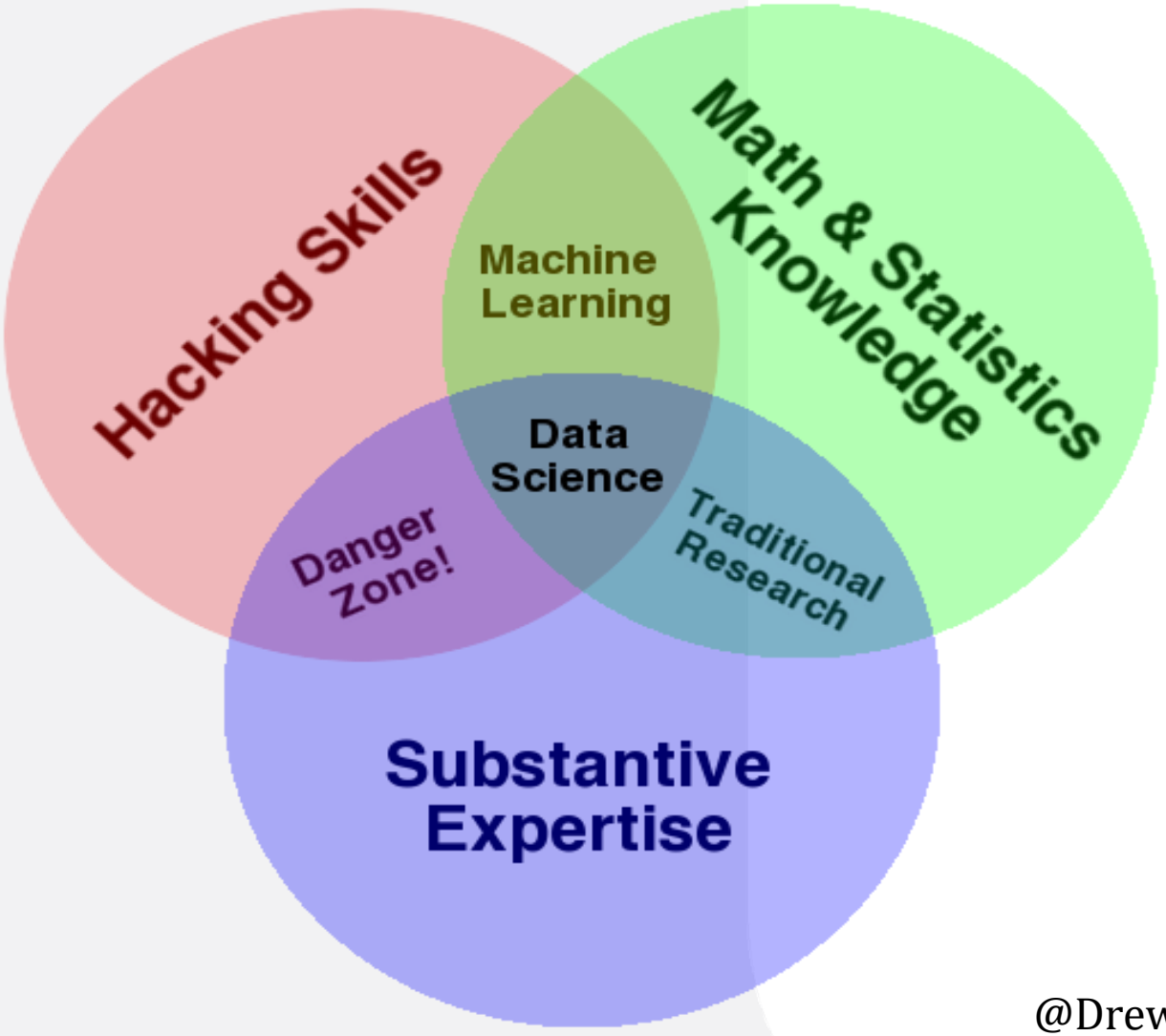
Analyse large datasets to:

- Find novel
- Commercially valuable
- Exploitable patterns

} **Meaningful insights**

Ulster University

Data Science Venn Diagram



Ulster University

@Drew Conway

What is a Data Scientist?

Someone with a diverse skill set and capable of:

- Identifying organisational needs
- Collecting, cleaning, and storing diverse datasets
- Performing in-depth analysis on collected data through application of software engineering, statistics and machine learning
- Perform experiments to test hypotheses and identify hidden trends
- Summarise findings and communicate insights to a range of stakeholders to create impact

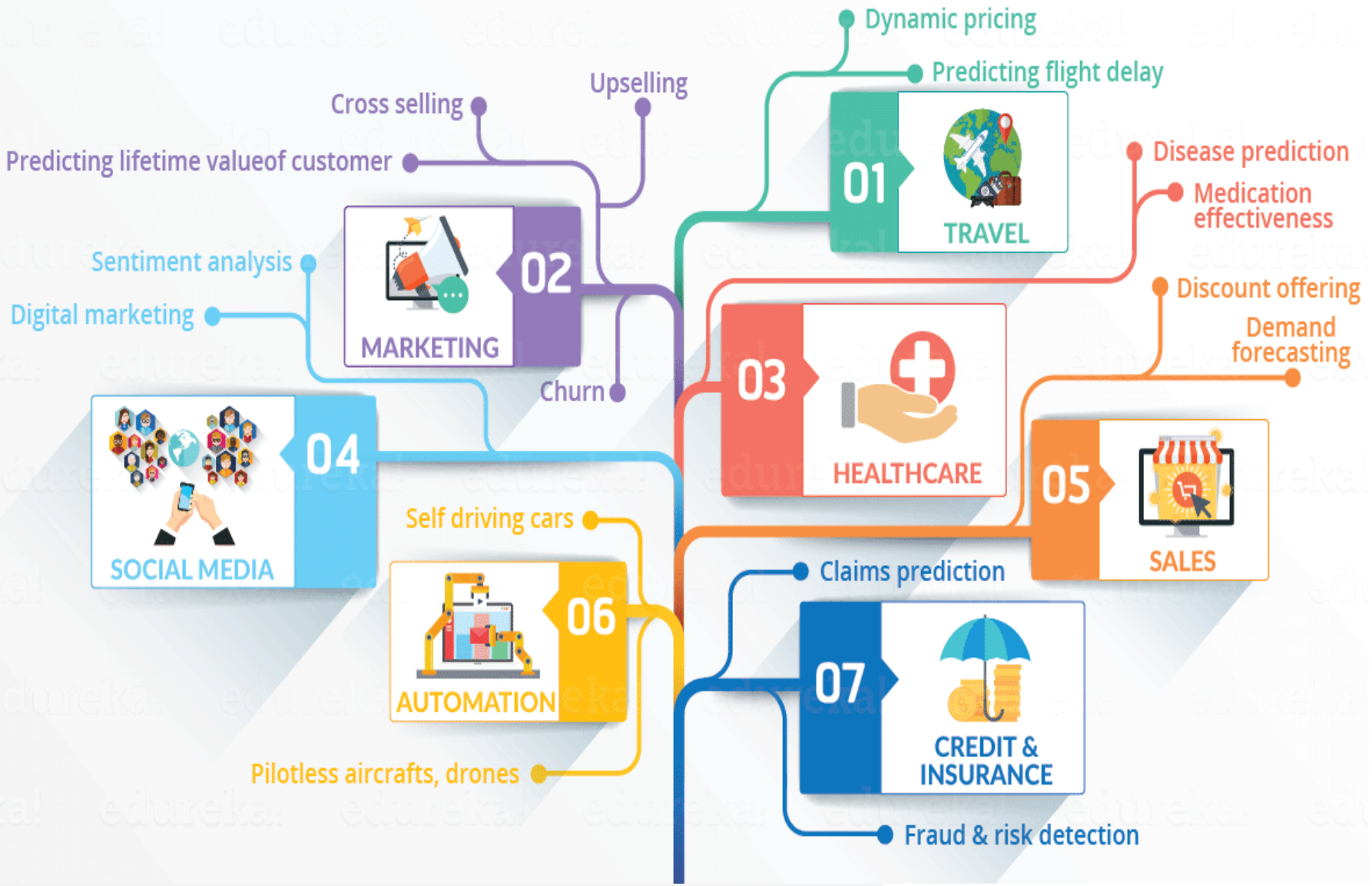
Ulster University

Potential of Data Science

Ulster University

What Areas can Benefit from Data Science?

Ulster University



Ulster University

Deliverables

- **Prediction**
  - Predict a value based on inputs
- **Classification**
  - E.g., spam or not spam
- **Recommendations**
  - E.g., Amazon and Netflix recommendations
- **Pattern detection and grouping**
  - E.g., classification without known classes

Ulster University

Deliverables

- **Anomaly detection**
  - E.g., fraud detection
- **Recognition**
  - E.g., image, text, audio, video, facial, etc.
- **Actionable insights**
  - via dashboards, reports, visualizations, …
- **Automated processes and decision-making**
  - E.g., credit card approval

Ulster University

Deliverables

- **Scoring and ranking**
  - E.g., FICO score
- **Segmentation**
  - E.g., demographic-based marketing
- **Optimization**
  - E.g., risk management
- **Forecasts**
  - E.g., sales and revenue

Ulster University

Data Science-Meaningful Insight

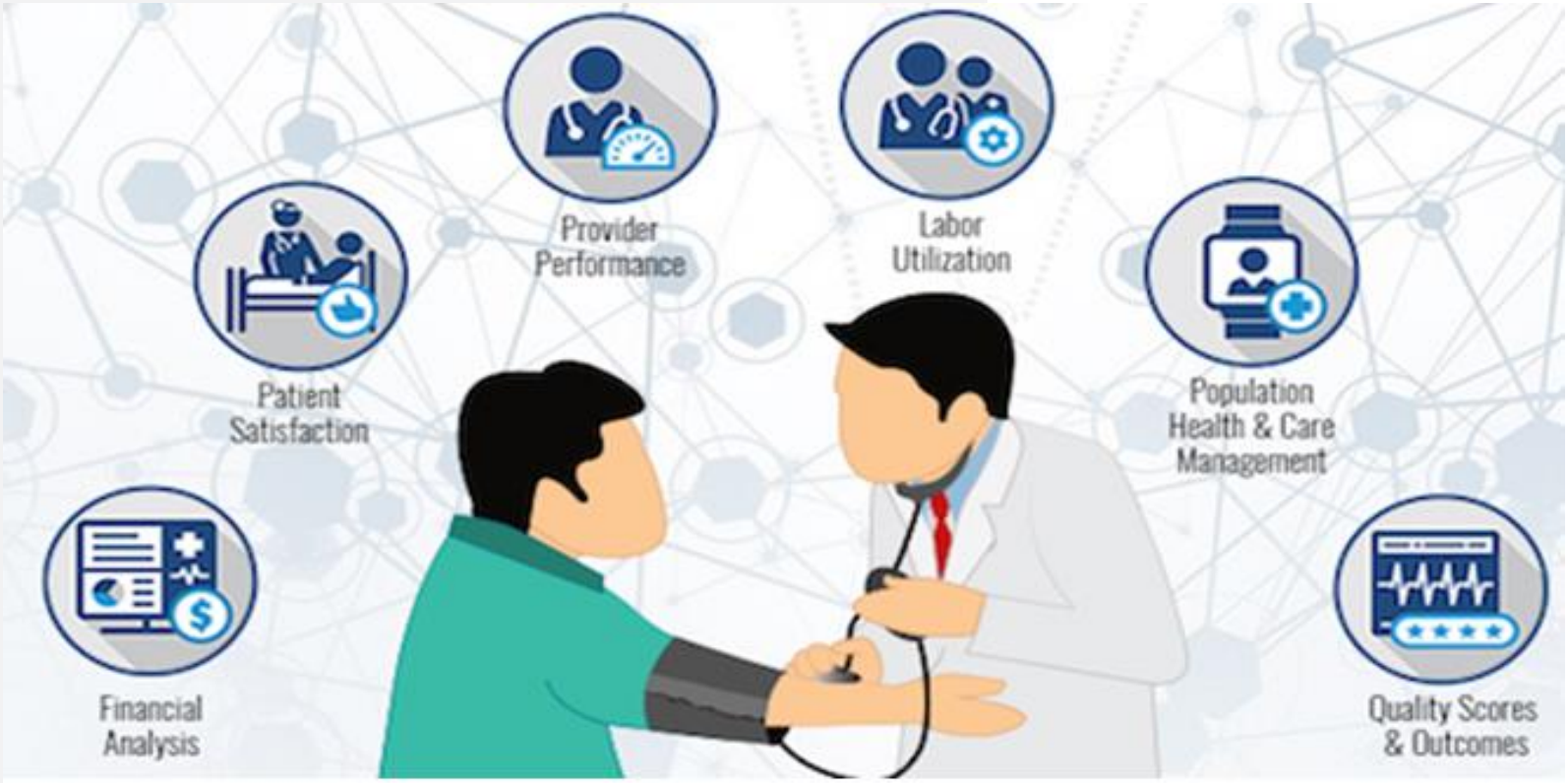
- Predicting from data with certain degree of certainty
- Generalise what has been observed in data
- Insight found in data are discoveries
- Discoveries are expressed as models
- Models are used for making prediction

Discoveries→ Models → Prediction

Ulster University

Example

Data Science is Transforming Healthcare Systems



Ulster University

Data Science – A Brief History

1962 – J.W. Tukey – Published "The Future of Data Analysis", mentioning the relationship between data analysis and statistics.

1974 – P. Naur – Published the "Concise Survey of Computer Methods", mentioning "Data Science".

1996 – "Data Science" used in a conference title

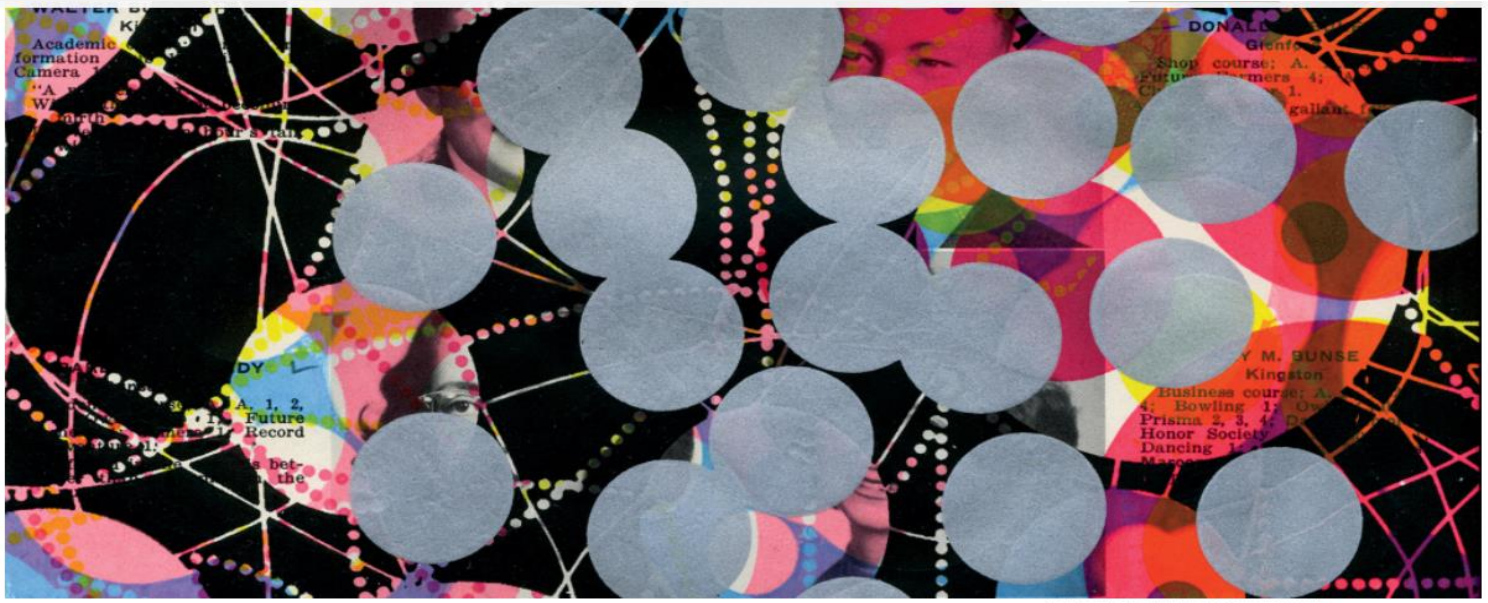
2001 – Introduced as an independent discipline.

2002 – Data Science Journals start publishing

Ulster University

Data Science – A Brief History

2012 – Harvard Business Review



ARTWORK: TAMAR COHEN, ANDREW J. BUBOLTZ, 2011, SILK SCREEN ON A PAGE FROM A HIGH SCHOOL YEARBOOK, 8.5" X 10"

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

SUMMARY SAVE SHARE 13 COMMENT TEXT SIZE PRINT $8.95 BUY COPIES

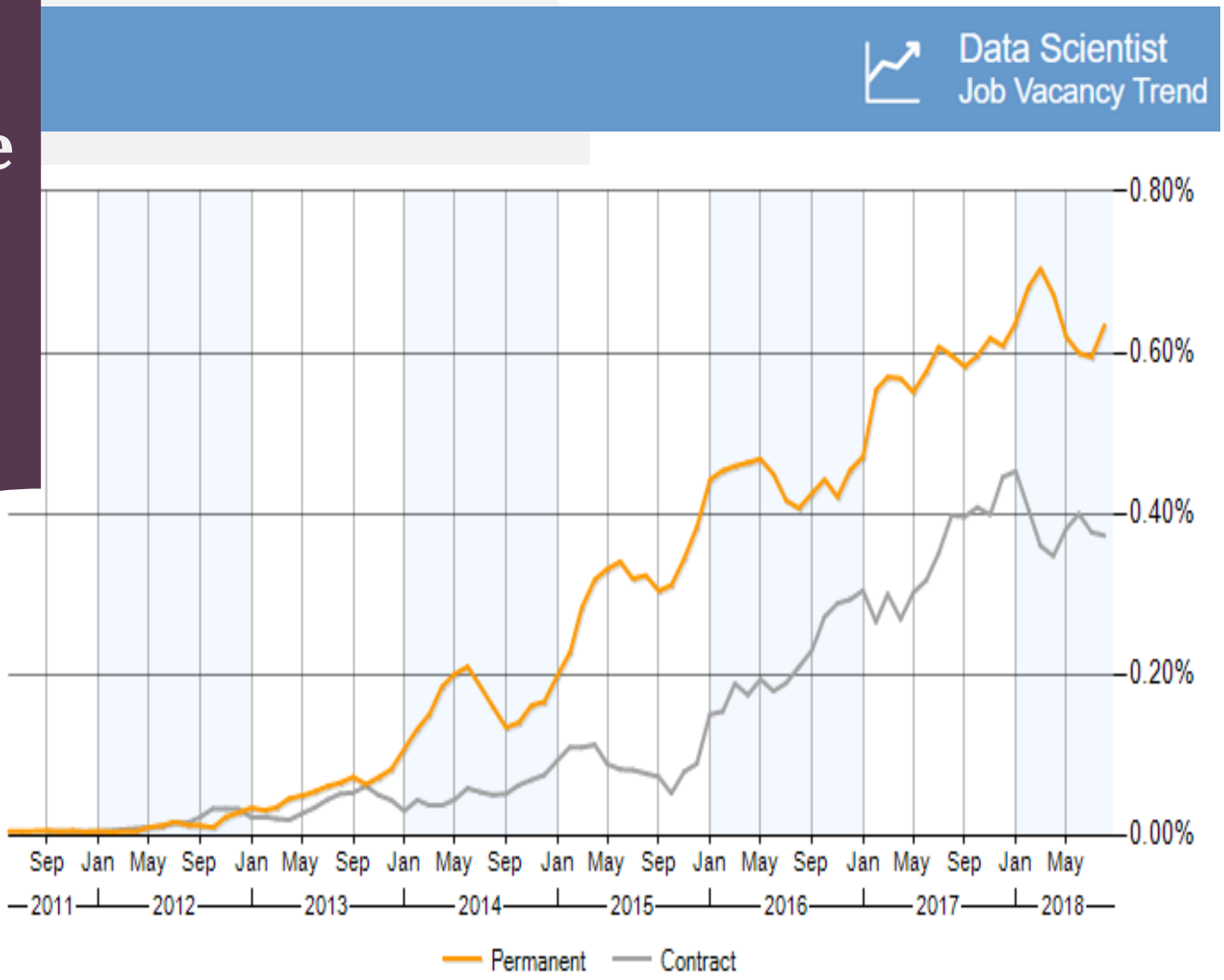
WHAT TO READ NEXT

Big Data: The Management Revolution

Ulster University

https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

Data Science – A Brief History



Job postings that featured Data Scientist in the job title as a percentage of all IT jobs advertised

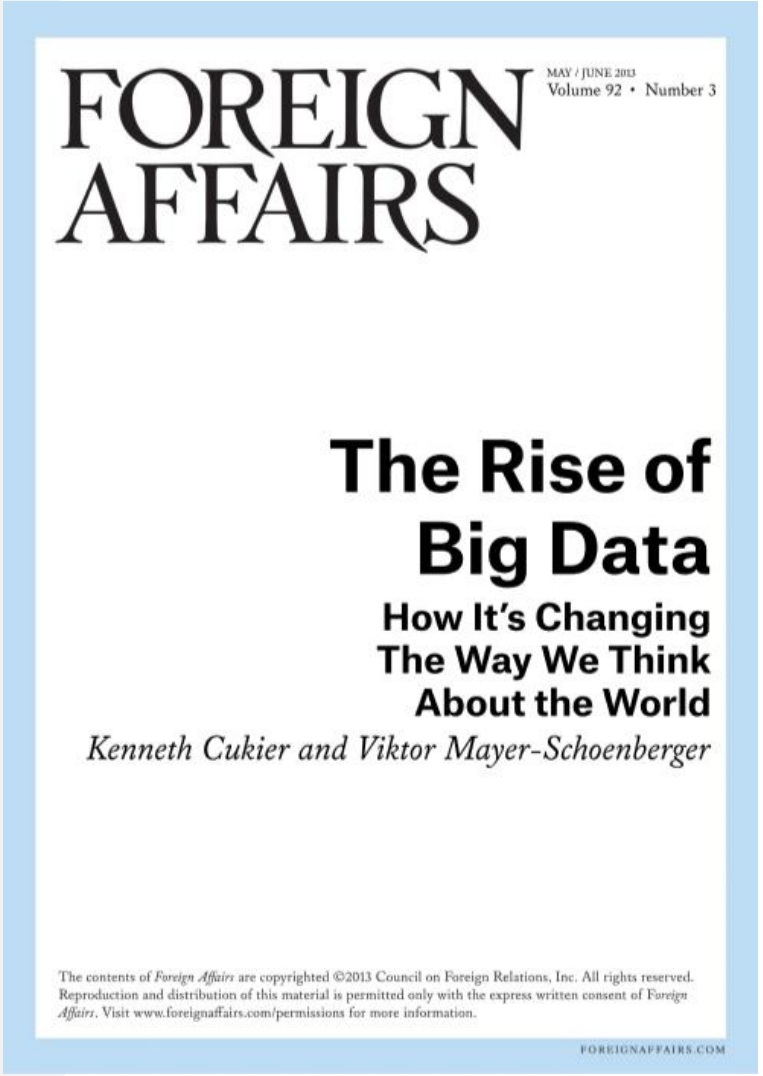
Ulster University

https://www.itjobswatch.co.uk/jobs/uk/data%20scientist.do

Big Data

The Rise of Big Data

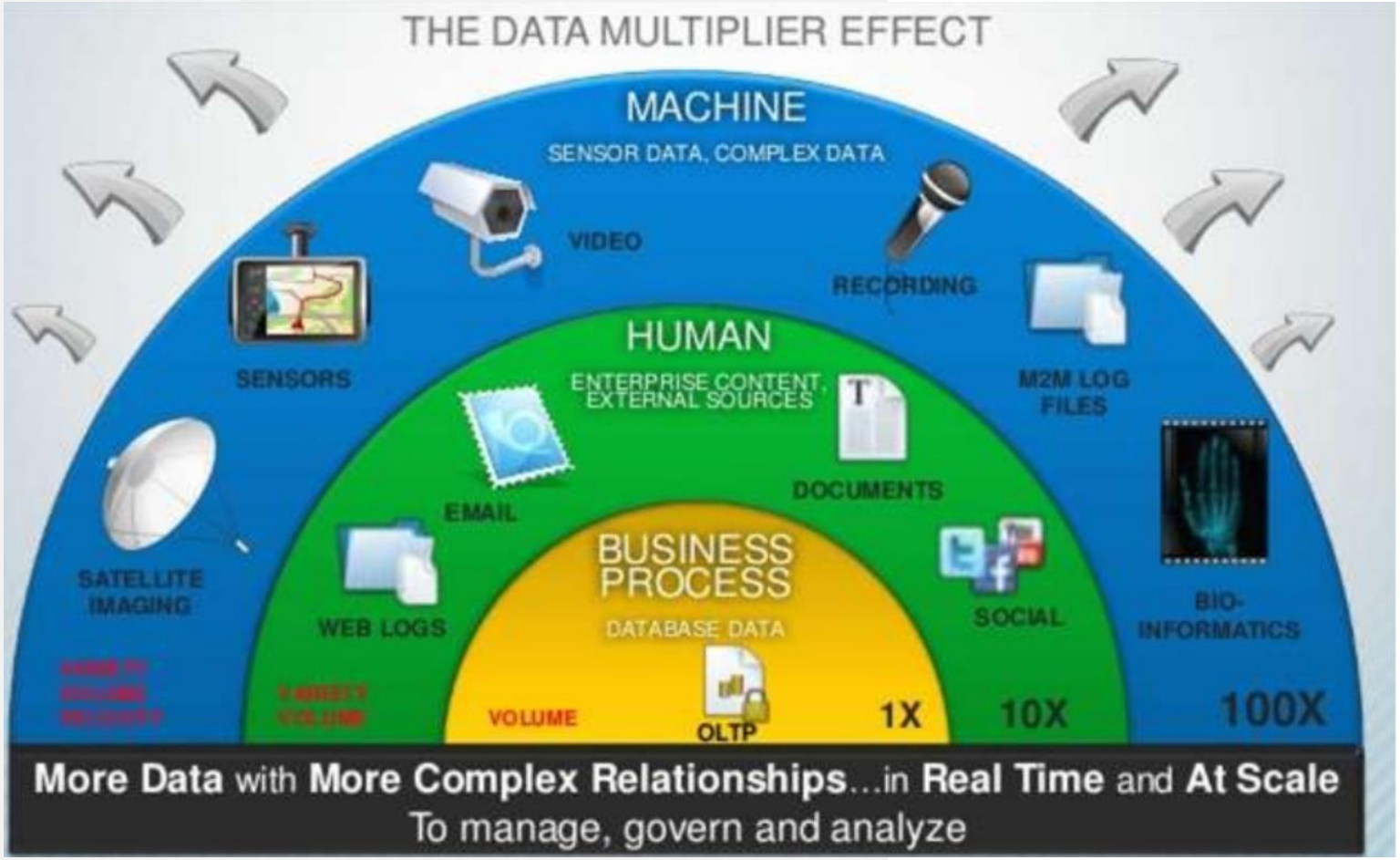
- ✓ **2000** – 25% of data is digital
- ✓ The amount of digital data doubles
- ✓ **2013** – Less than 2% of stored data
- ✓ **Datafication** – The ability to render into data of the world that have never been quantified before, e.g.:
  - Location (GPS)
  - Friendships (Facebook Likes)
  - Activity levels



Ulster University

https://www.foreignaffairs.com/system/files/pdf/articles/2013/92305.pdf

Data Types



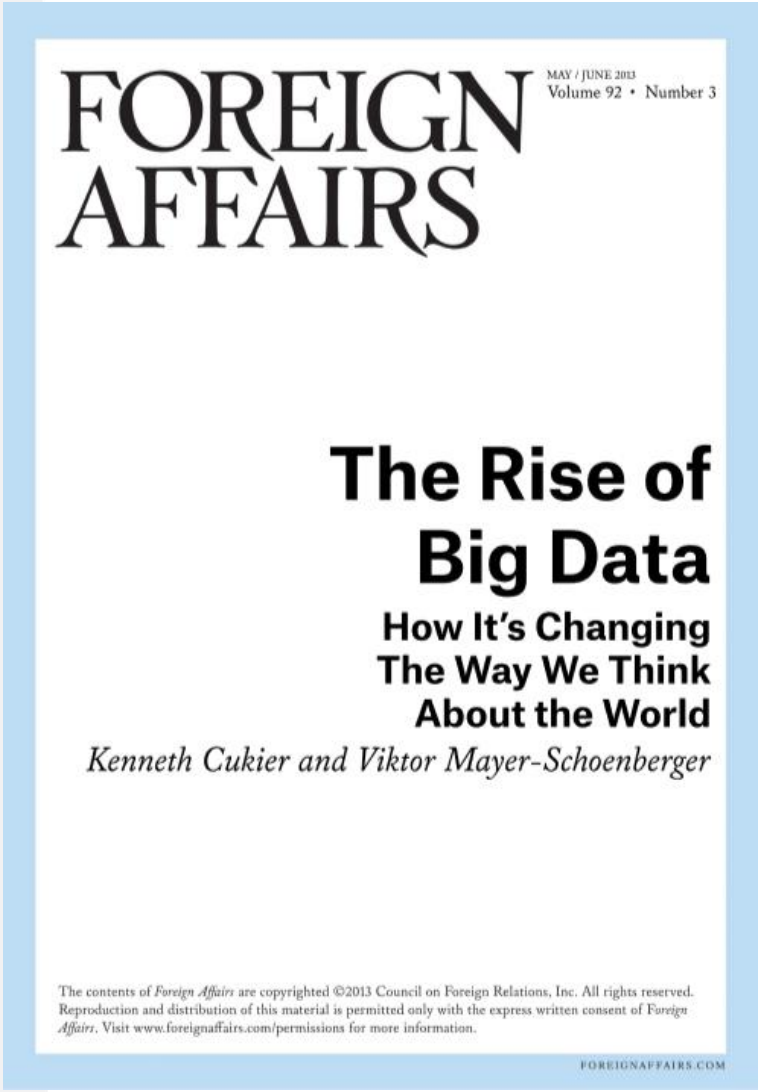
Ulster University

Big Data

The Rise of Big Data

Profound changes in approaching data:

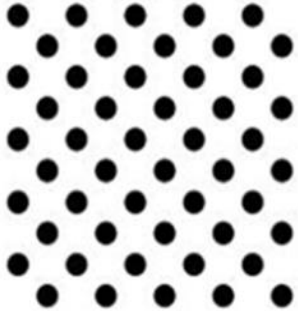
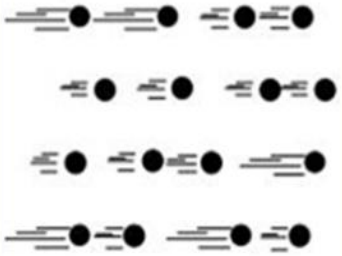
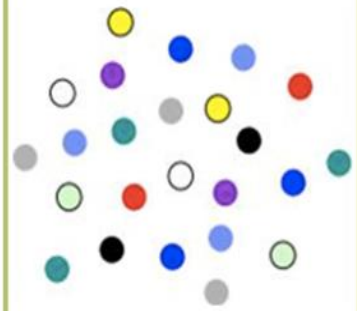
- ✓ Massive vs small samples
- ✓ Messiness vs Pristine
- ✓ Correlation vs Causation
  - E.g. Predicting delivery van breakdown



Ulster University

https://www.foreignaffairs.com/system/files/pdf/articles/2013/92305.pdf

Big Data - The 5 Vs Everyone Must Know

Volume	Velocity	Variety	Veracity	Value
				
Data at Rest	**Data in Motion**	**Data in Many Forms**	**Data in Doubt**	**Data into Money**
Terabytes to Exabytes of existing data to process	Streaming data, requiring milliseconds to seconds to respond	Structured, unstructured, text, multimedia,...	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations	Business models can be associated to the data

Ulster University

Big Data Variety

Ulster University

Ulster University

Sizing "Mobile + Social" Big Data Stats

The Multiple Dimensions of Big Data

Data Management

Data Architectures

Data Analytics

Data Protection

Data Visualization

Ulster University

Big Data vs Data Science

Big data

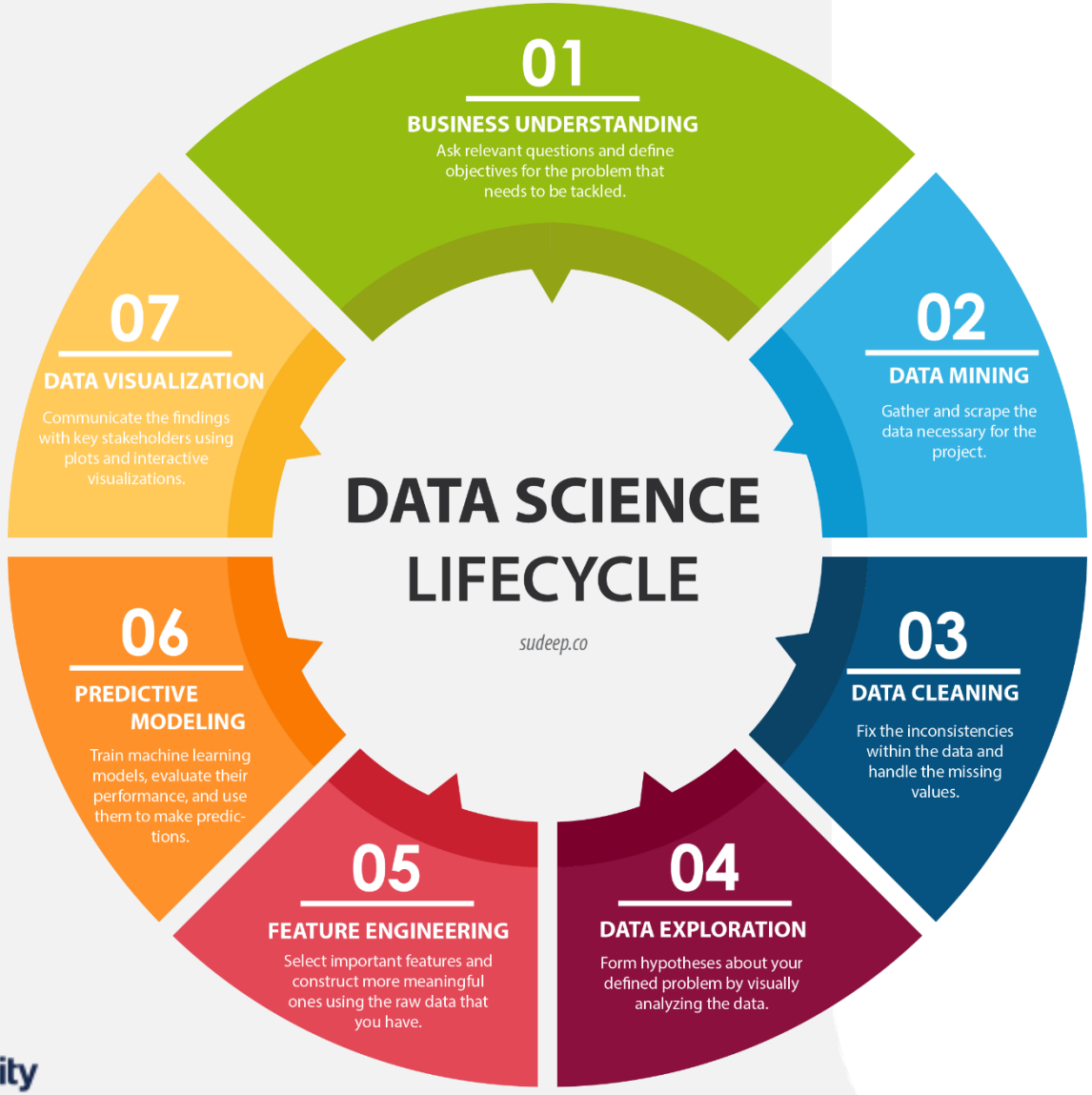
- → Relates to the efficient collection of a large volume of heterogeneous data
- → Data not stored in traditional way can be structured, semi-structured or unstructured
- → Not stored in traditional databases

Data Science

- → Deals with slicing and dicing of the big chunks of data
- → Finding insightful patterns and trends using technology, mathematics and statistical techniques
- → Intelligent analysis of large data

Ulster University

Lifecycle of Data Science



Ulster University

Lifecycle of Data Science

Phase 1—Discovery

Before you begin the project:

- Understand various specifications, requirements, priorities and required budget
- Ask the right questions
- Assess if you have the required resources to support the project:
  - People, technology, time and data to support the project
- Frame the business problem and formulate initial hypotheses (IH) to test

Ulster University

Lifecycle of Data Science

Phase 2—Data preparation

- Need to explore, preprocess and condition data prior to modeling
- Extract, transform, and load data
- Example:

  Can use R for data cleaning, transformation, and visualisation
- Will help to spot the outliers and establish a relationship between the variables

Ulster University

Lifecycle of Data Science

Phase 3—Model planning

- Get insights into the nature of the data
- Do **exploratory data analytics- Descriptive statistics**

  using various statistical formulas and visualization tools
- Determine methods and techniques to draw the relationships between variables→ Input features and output
- These relationships will set the base for the algorithms implemented in the next phase

Ulster University

Lifecycle of Data Science

Phase 4—Model building

- Build model using training dataset and test the performance of the model using test dataset
- Analyze various learning techniques like classification, association and clustering to build the model

Ulster University

Lifecycle of Data Science

Phase 5—Operationalize

- Final reports, briefings, code and technical documents
- Sometimes a pilot project is also implemented in a real-time production environment
- This will provide a clear picture of the performance and other related constraints on a small scale before full deployment

Ulster University

Lifecycle of Data Science

Phase 6—Communicate results

- Evaluate if goal was achieved that was planned in the first phase
- Identify all the key findings
- Communicate to the stakeholders
- Determine if the results of the project are a success or a failure based on the criteria developed in Phase 1

Ulster University

Case Study: Diabetes Prevention

Research Question:

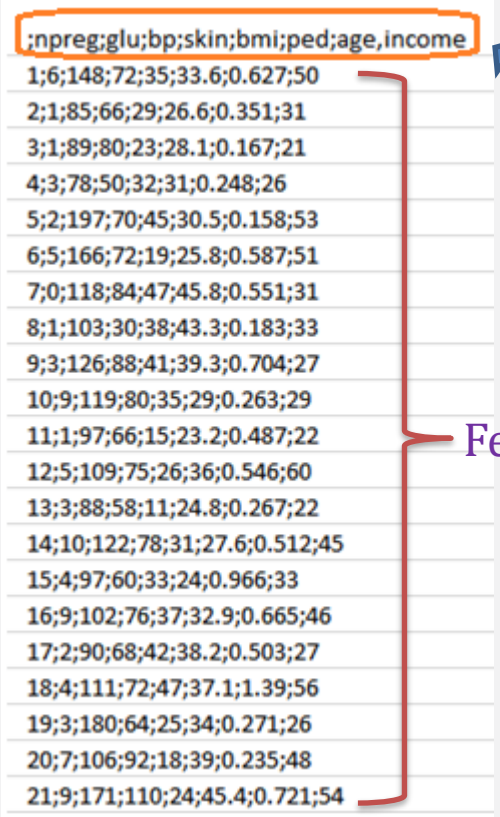
Could we predict the occurrence of diabetes and take appropriate measures beforehand to prevent it???

Ulster University

Case Study: Diabetes Prevention

Step 1:

Collect medical history data of the patient



Attributes referred to as **Features**

Feature values

Attributes	Details
npreg	Number of times pregnant
glucose	Plasma glucose concentration
bp	Blood pressure
skin	Triceps skinfold thickness
bmi	Body mass index
ped	Diabetes pedigree function
age	Age
income	Income

Ulster University

Case Study: Diabetes Prevention

Step 2:

- Now, once we have the data, clean and prepare the data for data analysis
- This data has a lot of inconsistencies like missing values, blank columns, abrupt values and incorrect data format which need to be cleaned

Ulster University

Case Study: Diabetes Prevention

Step 2:

- Let's have a look at the sample data below

	npreg	glu	bp	skin	bmi	ped	age	income
1	6	148	72	35	33.6	0.627	50	
2	1	85	66	29	26.6	0.351	31	
3	1	89	6600	23	28.1	0.167	21	
4	3	78	50	32	31	0.248	26	
5	2	197	70	45	30.5	0.158	53	
6	5	166	72	19	25.8	0.587	51	
7	0	118	84	47	45.8	0.551	31	
8	one	103	30	38	43.3	0.183	33	
9	3	126	88	41	39.3	0.704	27	
10	9	119	80	35	29	0.263	29	
11	1	97	66	15	23.2	0.487	22	
12	5	109	75	26	36	0.546	60	
13	3	88	58	11	24.8	0.267	22	
14	10	122	78	31	27.6	0.512	45	
15	4		60	33	24	0.966	33	
16	9	102	76	37	32.9	0.665	46	
17	2	90	68	42	38.2	0.503	27	
18	4	111	72	47	37.1	1.39	56	
19	3	180	64	25	34	0.271	26	
20	7	106	92	18		0.235	48	
21	9	171	110	24	45.4	0.721	54	

- *npreg*, "one" is written in words whereas it should be in the numeric form like
- *bp* one of the values is 6600 which is impossible
- *Income* column is blank and also makes no sense in predicting diabetes.
- Therefore, it is redundant to have it here and should be removed from the table

Ulster University

Case Study: Diabetes Prevention

Step 2:

- Clean and preprocess this data:
  - Removing outliers
  - Fill up the null values
  - Normalizing the data type

Data preprocessing→ Phase 2

	npreg	glu	bp	skin	bmi	ped	age
1	6	148	72	35	33.6	0.627	50
2	1	85	66	29	26.6	0.351	31
3	1	89	80	23	28.1	0.167	21
4	3	78	50	32	31	0.248	26
5	2	197	70	45	30.5	0.158	53
6	5	166	72	19	25.8	0.587	51
7	0	118	84	47	45.8	0.551	31
8	1	103	30	38	43.3	0.183	33
9	3	126	88	41	39.3	0.704	27
10	9	119	80	35	29	0.263	29
11	1	97	66	15	23.2	0.487	22
12	5	109	75	26	36	0.546	60
13	3	88	58	11	24.8	0.267	22
14	10	122	78	31	27.6	0.512	45
15	4	97	60	33	24	0.966	33
16	9	102	76	37	32.9	0.665	46
17	2	90	68	42	38.2	0.503	27
18	4	111	72	47	37.1	1.39	56
19	3	180	64	25	34	0.271	26
20	7	106	92	18	39	0.235	48
21	9	171	110	24	45.4	0.721	54

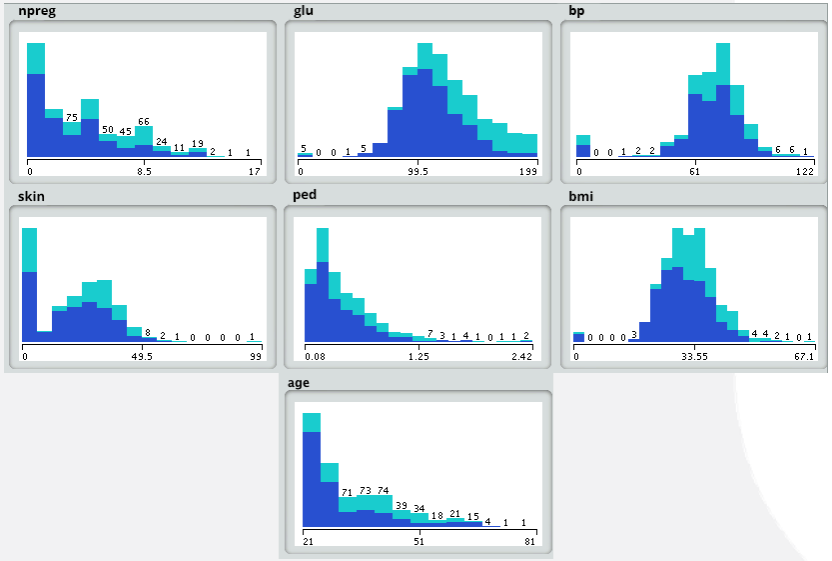
Ulster University

Case Study: Diabetes Prevention

Step 3:

Exploratory data analytics

- Use functions like *describe* in R→ Number of missing values and unique values
- *Summary* function which will give us statistical information like mean, median, range, min and max values
- Use visualization techniques like histograms, line graphs, box plots to get a fair idea of the distribution of data

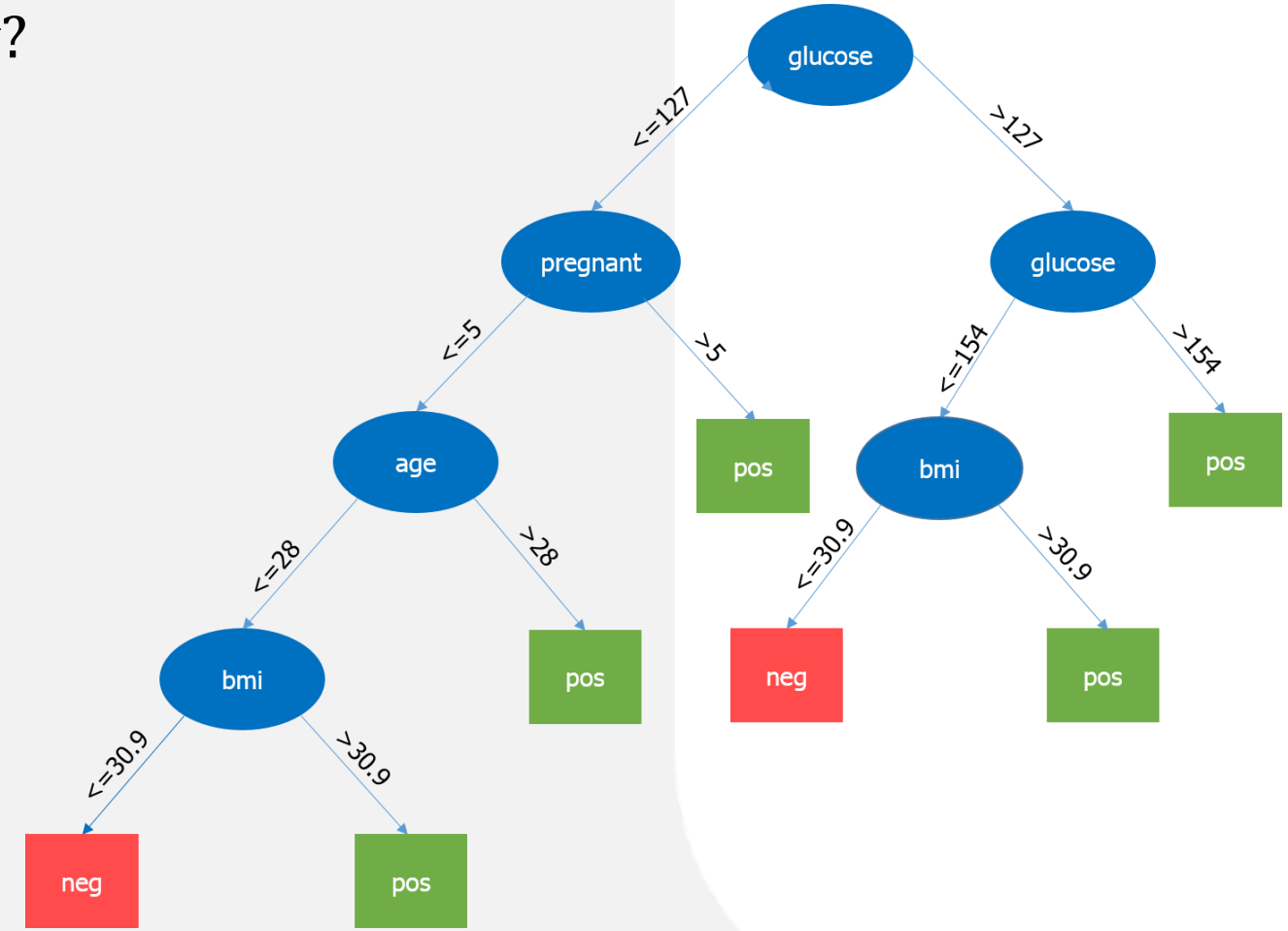


Ulster University

Case Study: Diabetes Prevention

Step 4:

- Based on insights derived from the previous step, the best fit for this kind of problem is the **decision tree**
- Let's see how?



Ulster University

Case Study: Diabetes Prevention

Step 5:

- Run a small pilot project to check if results are appropriate→ **Validation**
- Also look for performance constraints if any
- If the results are not accurate, replan and rebuild the model.

Step 6:

- Once the project executed successfully
- Share the output for full deployment

Ulster University

Introduction to Python

ulster.ac.uk

Python

- Released in 1989
- Current version: 3.9.0 (January 2021)
- Simplicity -Precise, efficient, readable syntax
- High-Level – abstracts details such as memory management
- General purpose – AI, statistics, visualisation, etc.
- Comprehensive libraries - analysis, machine learning, visualisation
- Interpreted language – evaluation of code happens immediately, rather than going through a compile and run cycle
- Recommended for those with a Computer Science / Software Engineering background

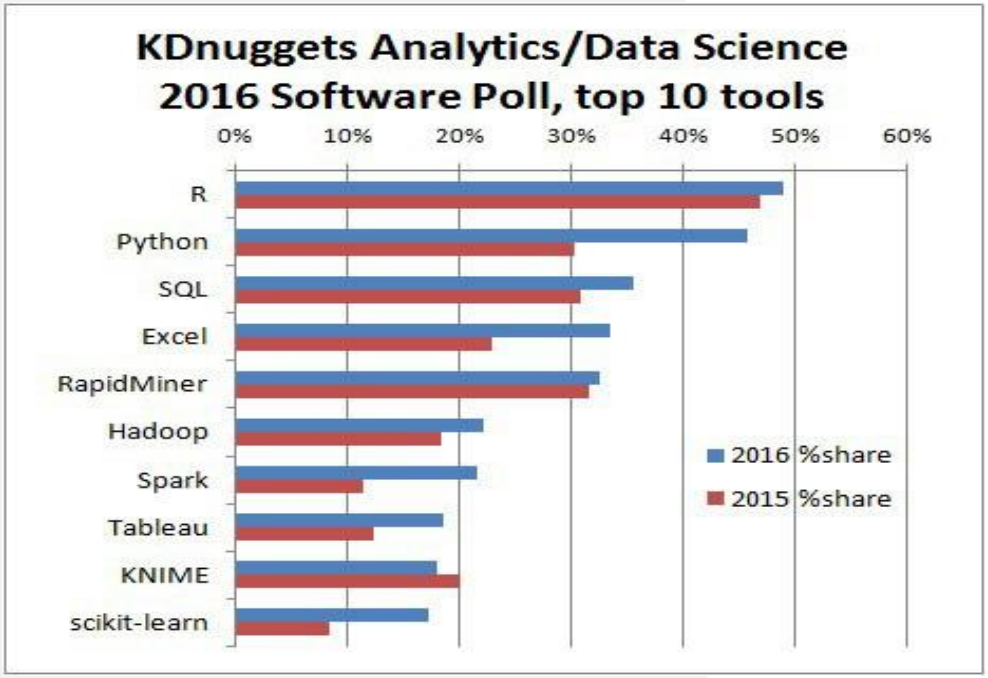
R

- Released in 1992
- Current version: 3.6.3
- Comprehensive statistics and data mining packages
- Steeper learning curve
- More intuitive for those with a mathematics / statistician background

Ulster University

Python vs R

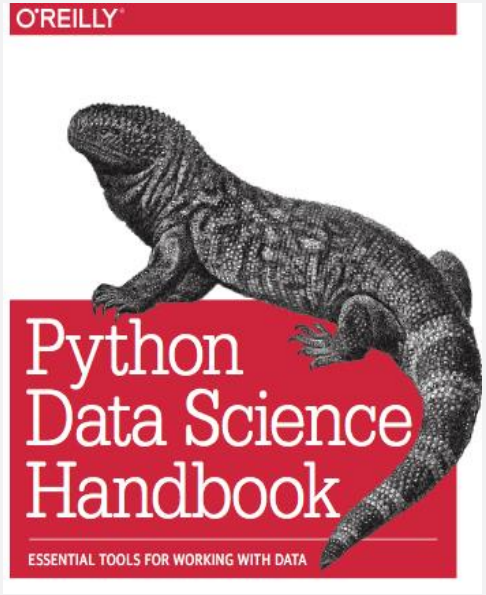
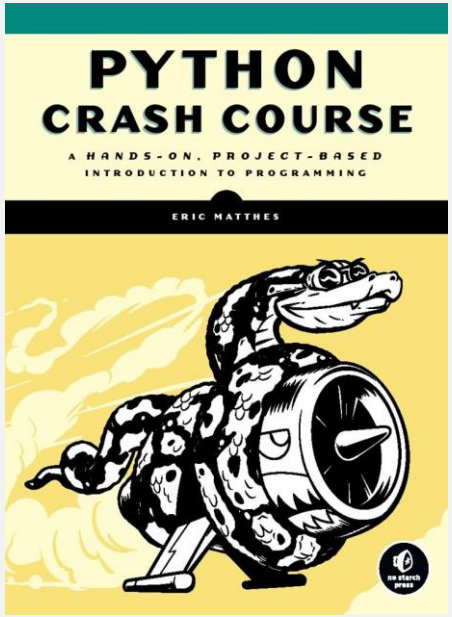
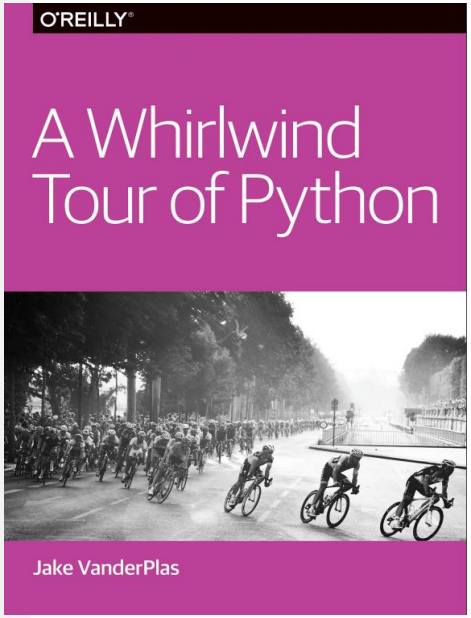
- ✓ Both excellent languages
- ✓ No “one size fits all”
- ✓ Python serves as a good platform for introduction to data science (less time learning, more time doing)
- ✓ R will be introduced from Week 7 onwards



Ulster University

@KDnuggets Analytics/Data Science 2016 Software Poll: top 10 most popular tools in 2016

Learning Python-Free Learning Resource

		
https://github.com/jakevdp/PythonDataScienceHandbook	https://github.com/MrAlex6204/Books/blob/master/python-crash-course.pdf	https://www.oreilly.com/programming/free/files/a-whirlwind-tour-of-python.pdf

http://python.org/
- documentation, tutorials, beginners guide, core distribution, ...

Ulster University

Installing Python

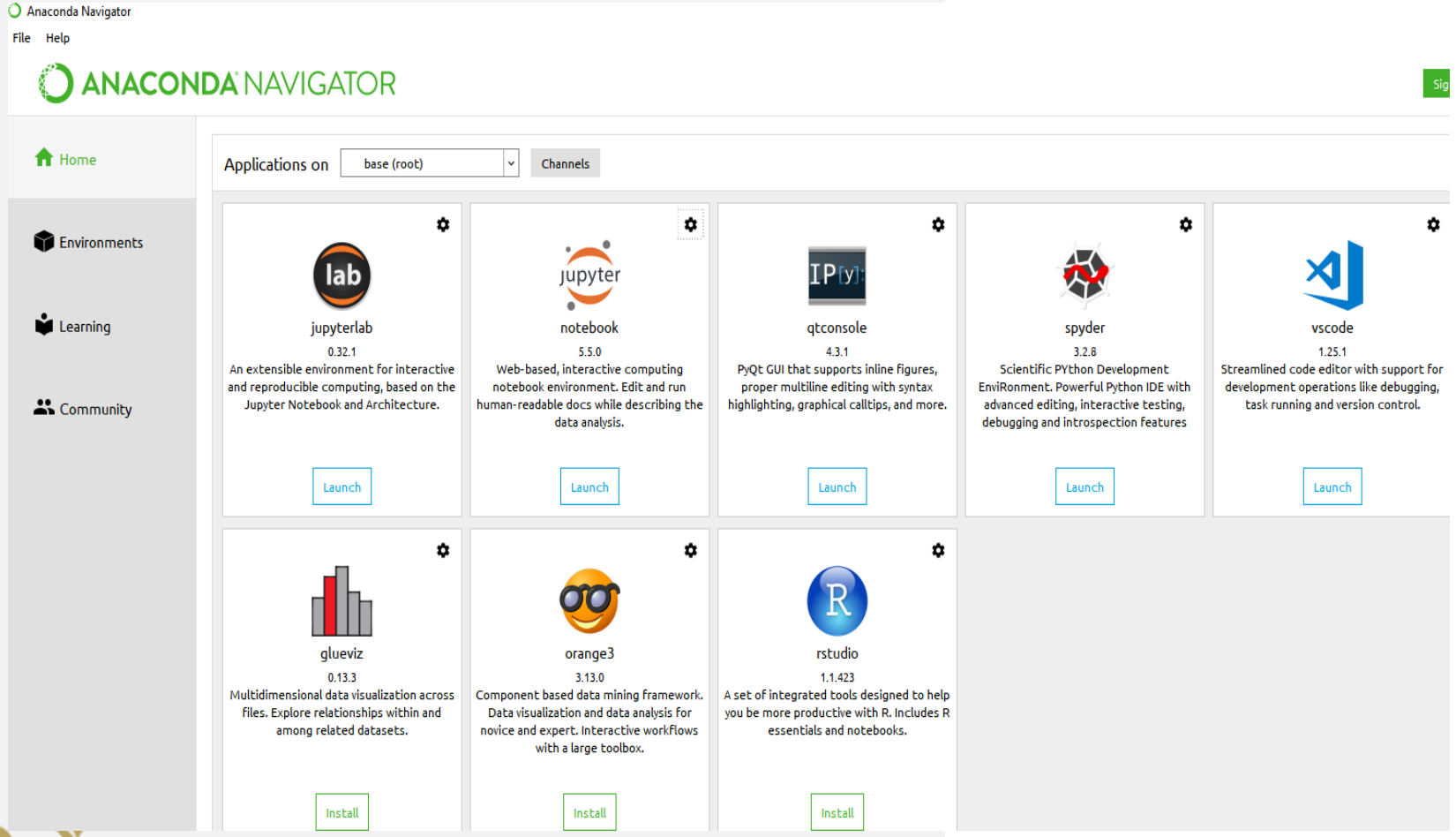
Easiest method: Use Anaconda

- ✓ A free Python package manager
- ✓ Can be used to install and update Python Packages
- ✓ Including all dependencies
- ✓ Both a Command Line Interface and GUI
- ✓ Already installed in the labs
- ✓ For home use and documentation: *https://docs.continuum.io/*

Ulster University

Anaconda

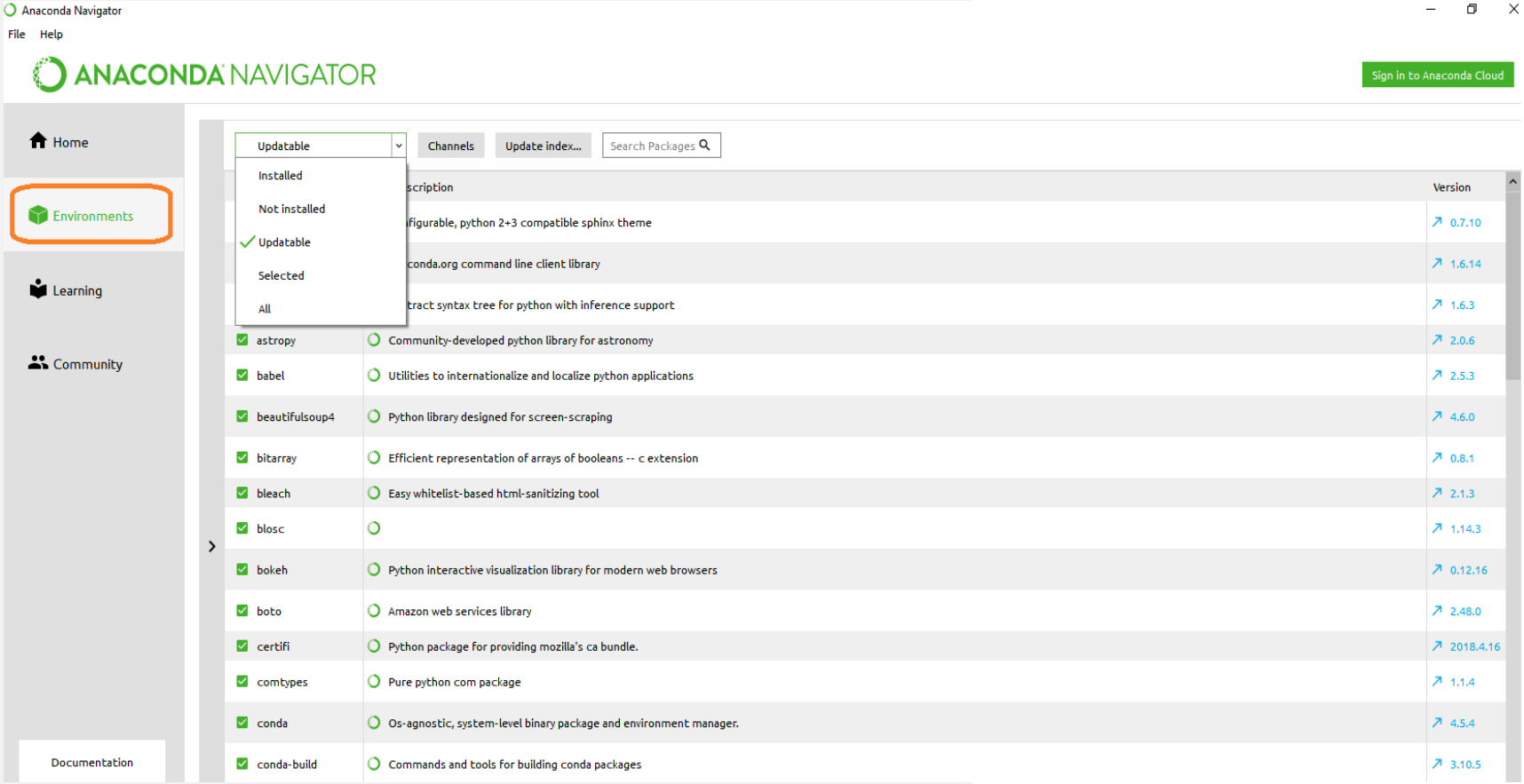
GUI



Ulster University

Anaconda

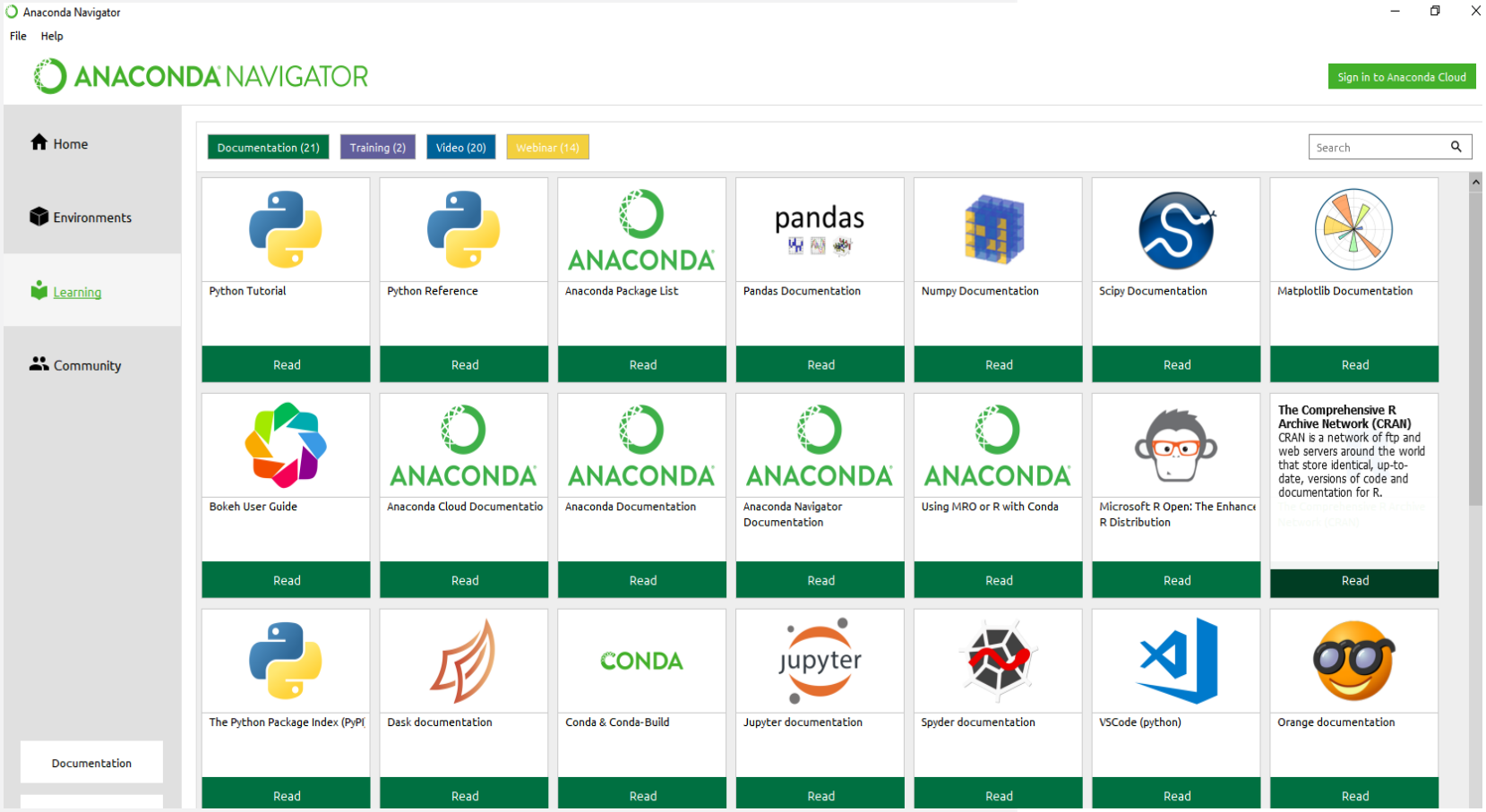
Installed Packages and Version



Ulster University

Anaconda

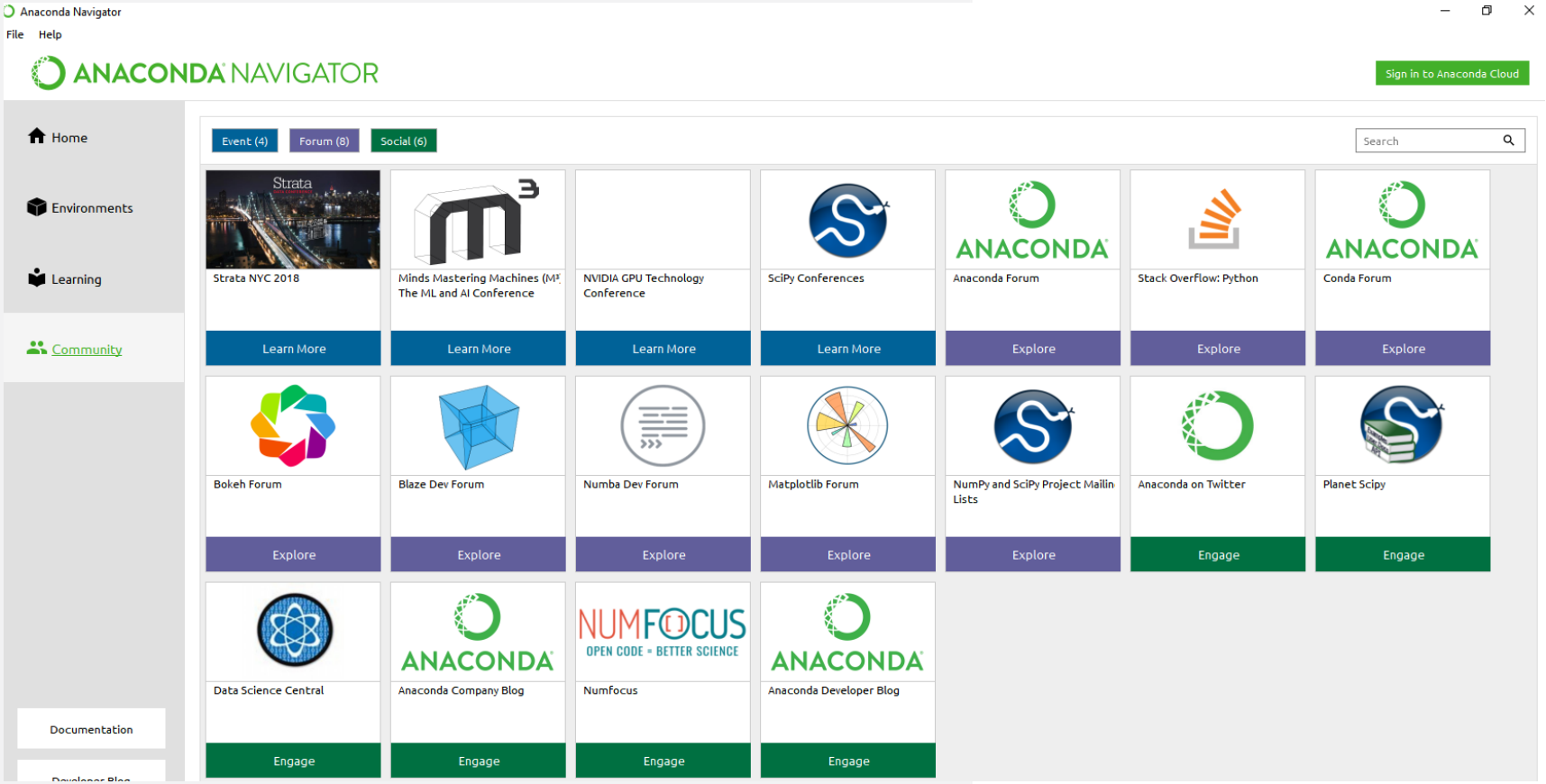
Learning materials



Ulster University

Anaconda

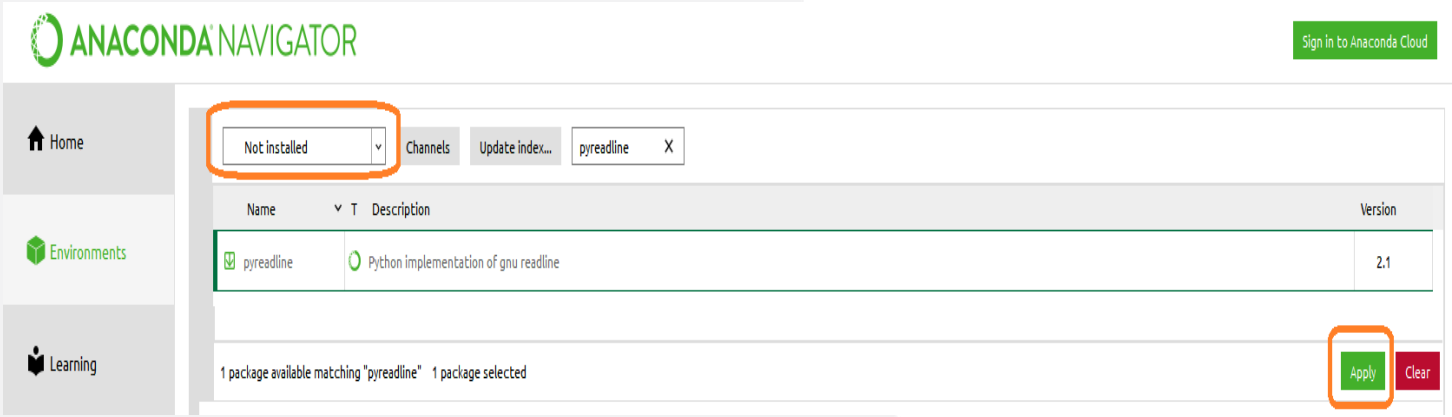
Access to community:



Ulster University

Jupyter Notebook

- ✓ A web application to create documents with live code, visualisations, images and text
- ✓ Ideal for beginners and advanced users
- ✓ Jupyter = Julia, Python, and R – the first languages supported
- ✓ Main Components:
  - Kernel – Runs and inspects user code
  - Dashboard – For access to all notebooks created, and for control over the kernel
- ✓ To enable Tab Completion, ensure the “pyreadline” package is installed



Ulster University

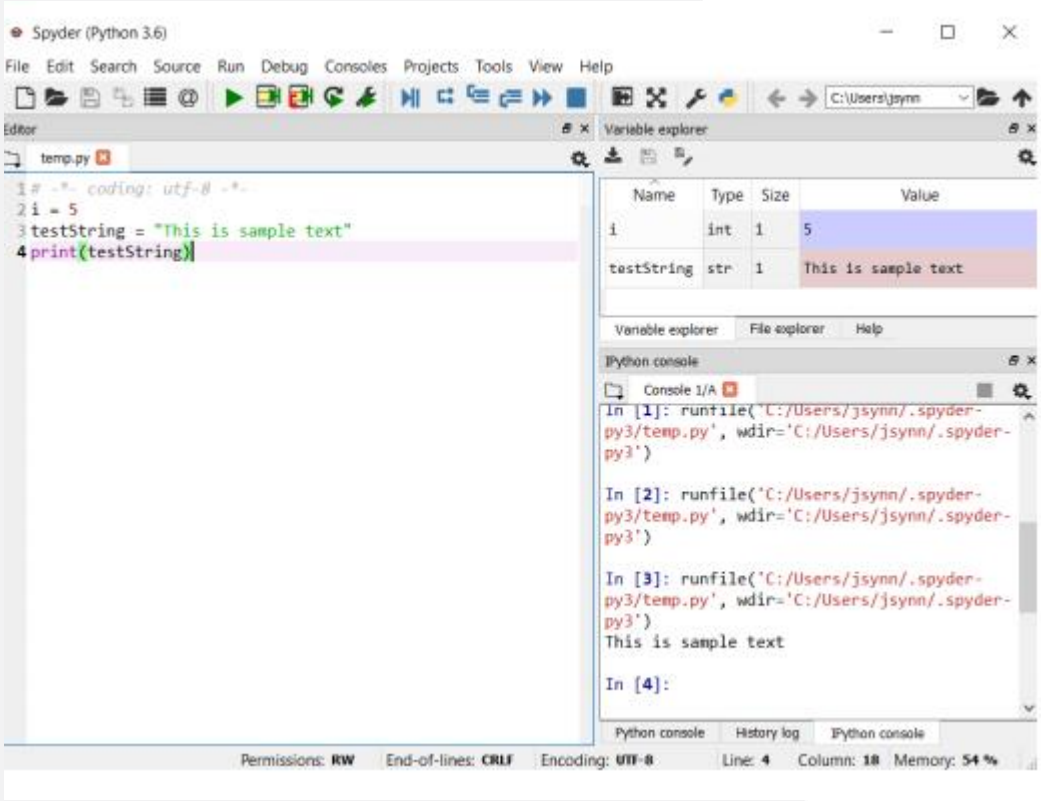
Jupyter Notebook

- ✓ Allows segments of code to be re-run quickly
- ✓ Assess impact, experiment, debug
- ✓ **View code and output together**
- ✓ Annotate and share code
- ✓ Host remotely, (e.g. high powered server), access anywhere

Ulster University

Spyder

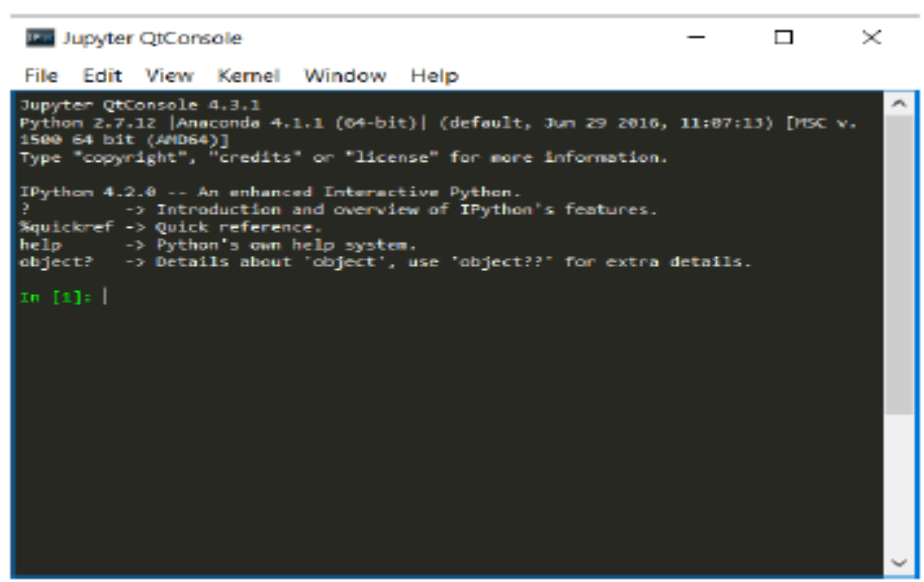
- ✓ Scientific Python Development IDE
- ✓ Full featured IDE. Ideal for larger codebases
- ✓ Syntax highlighting
- ✓ Exploration and editing of variables from a GUI



Ulster University

Jupyter QtConsole

- ✓ A very lightweight application, acts like an “enhanced terminal”
- ✓ Useful for quickly checking Python object help documents
- ✓ Also supports inline figures, multi-line editing, syntax highlighting etc.

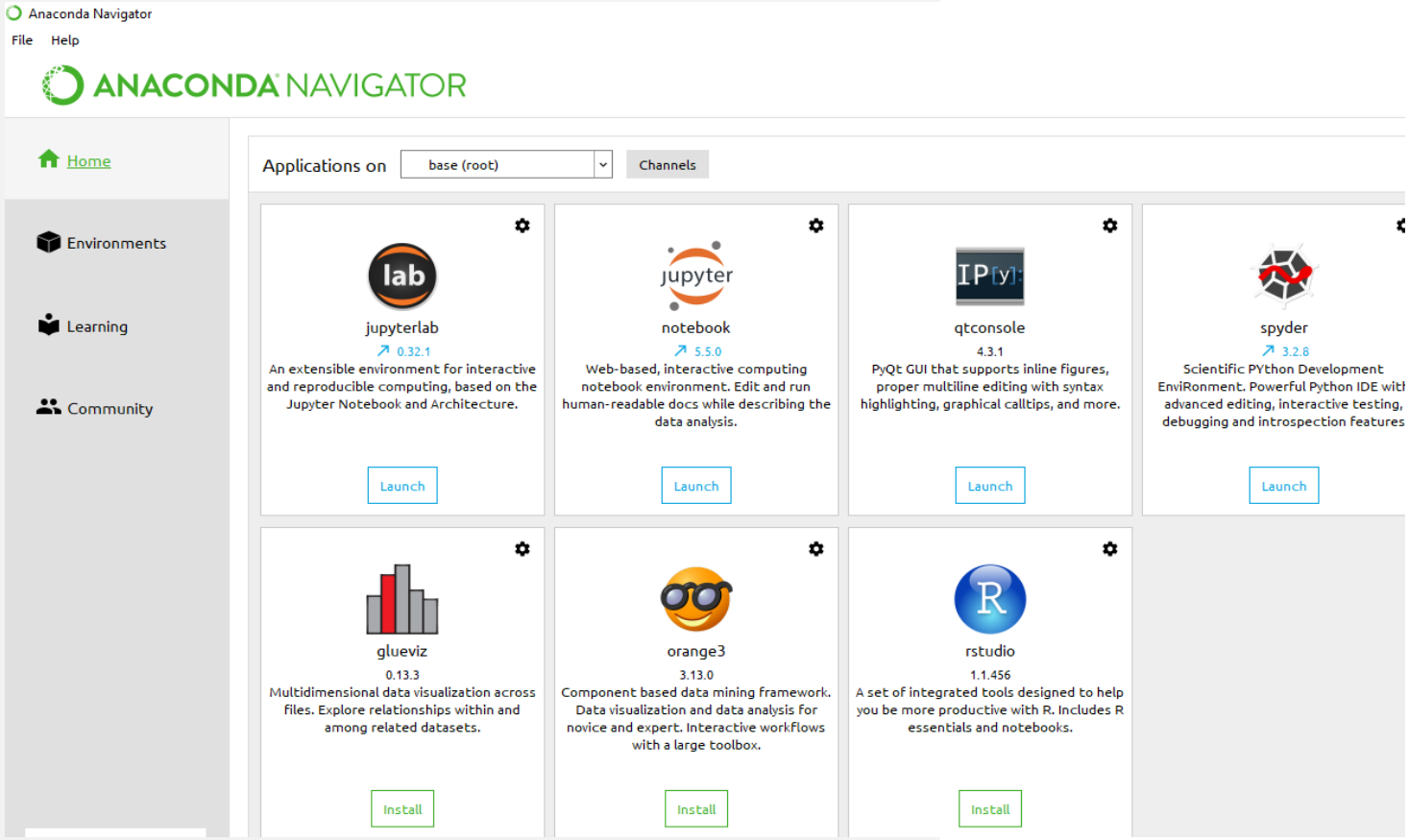


Ulster University

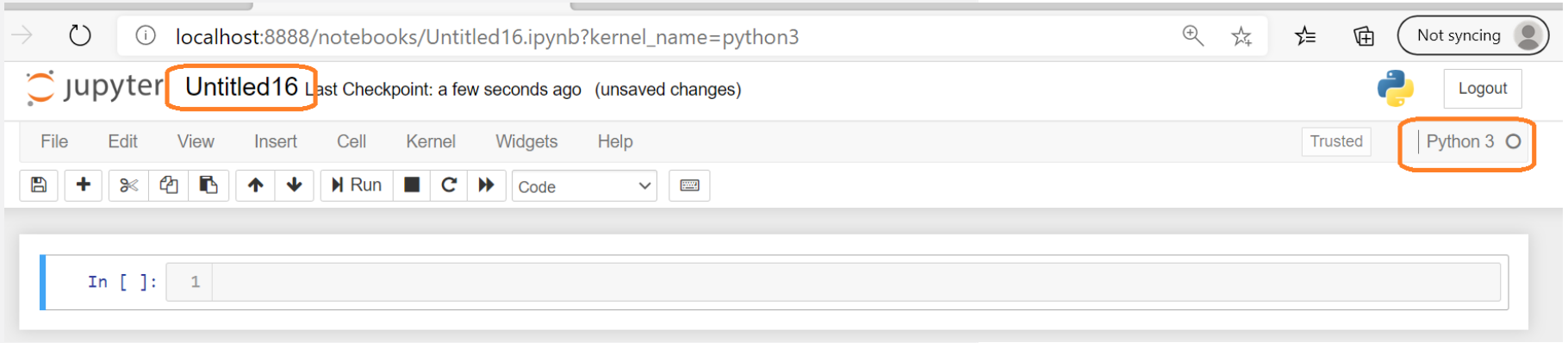
For this module I recommend using Jupyter notebook

Working with Jupyter Notebook

- Launch the Jupyter Notebook from Anaconda Navigator



Ulster University



Ulster University

Python Fundamentals

- Whitespace Formatting
  - Python uses **indentation** to delimit blocks of code.
  - Other languages use curly brackets {}
  - E.g.

```
for i in [1,2,3,4,5]:
    print(i) #first line in 'for i' block
    for j in [6,7,8,9,10]:
        print(j)
        print(i + j) #Last line in 'for j' block
    print(i) #Last line in 'for i' block
print("end") #First line after loops.
```

  - Whitespace **within** lines has no impact

Ulster University

Python Fundamentals

- Whitespace Formatting
  - Ignored inside:
    - Parentheses (round brackets) – ( )
    - Curly brackets, aka. Braces – { }
    - Brackets, aka. Square Brackets – [ ]

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

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

* Both variables are same. But the second one is more readable presentation

- You can also use a **backslash to continue** a statement on a new line

```
multiplication = 4 * \
                 3
```

Ulster University

Python Fundamentals

- Comments:

```
testString = "functional line" #Comment in line
#Comment in separate line
```

- Parenthesis ()
  - Used for grouping mathematical operations:

```
In [18]: 2*(5+2)
Out[18]: 14
```

  - And to indicate a function is being called:

```
print('text')
text
```

  - **Note:** Must be used during a function call even if no parameters are being supplied:

```
numbers = [4,3,1,2]
numbers.sort()
print(numbers)
[1, 2, 3, 4]
```

Ulster University

Python Fundamentals

- Terminology
  - A **statement** is an instruction that the language implementation understands. In most languages, statements do not return values.
  - A **function** is a subroutine that can be called elsewhere in the program. They often return values.
  - A **control structure** is a statement that can be used to direct the flow of execution (e.g. an if statement)
  - An **expression** is a piece of code that evaluates to a value (e.g. 1+3)

Ulster University

Python Fundamentals

- The *print* function
  - Something you will notice when following online tutorials:
    - Python **2.x** – *print* acts is a **statement**

```
numbers = [4,3,1,2]
numbers.sort()
print numbers
```

- Python **3.x** – *print* is a **function**

```
numbers = [4,3,1,2]
numbers.sort()
print(numbers)
```

Ulster University

Python Fundamentals

Saving the work in Jupyter notebook

Ulster University

Python Fundamentals

```
x = 1        #x is an integer
x = 'hello'  #x is now a string
x = [1,2,3]  #x is now a list
```

Variables

- In Python, these are **pointers** to a bucket in memory, e.g.:

```
x=2
```

- This is defining a pointer, called *x*
- This pointer points to **memory bucket**, containing the value *2*
- Therefore:
  - Python variables just point to various objects
  - A variable can be assigned to different object types
  - There is no need to declare the variable or its type (e.g. int x = 2)
  - Python is **dynamically typed** to its variable type

Ulster University

Python Fundamentals

Python Collections (Arrays)- Built-In Data Structures

There are four collection data types in Python:

- **List -** ordered and changeable. Allows duplicate members
- **Tuple -** ordered and unchangeable. Allows duplicate members
- **Set -** unordered and unindexed. No duplicate members
- **Dictionary -** unordered, changeable and indexed. No duplicate members

LIST	TUPLE	DICTIONARY	SET
Allows duplicate members	Allows duplicate members	No duplicate members	No duplicate members
Changeble	Not changeable	Changeable, indexed	Cannot be changed, but can be added, non -indexed
Ordered	Ordered	Unordered	Unordered
Square bracket [ ]	Round brackets ( )	Curly brackets{ }	Curly brackets{ }

Ulster University

Python Fundamentals

Built-In Data Structures- Collection types

- Aka Compound types
- Act as containers for other types
- These are:

Type Name	Description	Example
list	Ordered collection	[1, 2, 3]
tuple	**Immutable** ordered collection	(1, 2, 3)
dict	Unordered (Key, Value) mapping	{'a':1, 'b':2,' c':3}
set	Unordered collection of **unique** values	{1, 2, 3}

Ulster University

Python Fundamentals

- Built-In Data Structures
  - **Lists**
    - Ordered and Mutable.
    - Definition:
    - Useful properties and methods:

```
animals = ["cat","dog","mouse"]
```

Property/Method	Description	Example
len()	Return the number of elements in the list	len(animals) returns 3
append()	Add the supplied object to the end of the list	animals.append("elephant")
Concatenation	Combine two lists	animals + ["cow","eel"]
sort()	Sorts in ascending order.	animals.sort()

Ulster University

Python Fundamentals

Python index starts from 0

Built-In Data Structures

- **Lists**
  - Can contain multiple data types simultaneously:

```
mixedItems = ["green", 3, None, [1,2,3], True]
mixedItems
```

```
['green', 3, None, [1, 2, 3], True]
```

  - **Indexing** and **Slicing** allow access to specific elements.
    - **Indexing**
      - Single elements, using a zero-based index.

```
animals = ["cat","dog","chicken"]
animals[1]
```

```
'dog'
```

      - Negative indexes allow us to start from the end of the list:

```
animals = ["cat","dog","chicken"]
animals[-1]
```

```
'chicken'
```

Ulster University

Python Fundamentals

List

```
In [3]:
1 thislist = ["apple", "banana", "cherry"]
2 print(thislist)
```

```
['apple', 'banana', 'cherry']
```

Access Items

```
In [4]:
1 thislist = ["apple", "banana", "cherry"]
2 print(thislist[1])
```

```
banana
```

Change Item Value

```
In [5]:
1 thislist[1] = "blackcurrant"
2 print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

Loop Through a List

```
In [6]:
1 thislist = ["apple", "banana", "cherry"]
2 for x in thislist[1]:
3   print(x)
```

```
b
a
n
a
n
a
```

List Length

```
In [8]:
1 thislist = ["apple", "banana", "cherry"]
2 print(len(thislist))
```

```
3
```

Add Items

To add an item to the end of the list:
use the append() method:

```
In [9]:
1 thislist = ["apple", "banana", "cherry"]
2 thislist.append("orange")
3 print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

To add an item at the specified index:
use the insert() method:

```
In [10]:
1 thislist = ["apple", "banana", "cherry"]
2 thislist.insert(1, "orange")
3 print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

Python Fundamentals

- Built-In Data Structures
  - **Lists**
    - **Slicing**
      - Facilitates accessing multiple values in a list
      - Use the format: **listName[firstIndex:lastIndex]**

```
word="sentence"
word[0:3]
```

```
'sen'
```

      - Note that ***first Index* is inclusive, *last Index* is not.**
    - The following table shows which positive/negative index corresponds to which list item

Index	0	1	2	3	4	5	6	7	8
List item	**s**	**e**	**n**	**t**	**e**	**n**	**c**	**e**	**!**
Negative Index	-9	-8	-7	-6	-5	-4	-3	-2	-1

Ulster University

Python Fundamentals

- Built-In Data Structures
  - **Lists**
    - **Slicing**

[start index included : last index excluded]

Index	0	1	2	3	4	5	6	7	8	9
List item	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Negative Index	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- We can also omit the first, or last index, defaulting to the length of the index, e.g.:

```
numbers[:3]
[1, 2, 3]
```

```
numbers[3:]
[4, 5, 6, 7, 8, 9, 10]
```

```
numbers[-3:]
[8, 9, 10]
```

-3+1= -2
-2+1= -1

Next retrieval is done by adding +1 to the current index, therefore traversal always goes in forward direction for both positive and negative index

Ulster University

Python Fundamentals

- Built-In Data Structures
  - **Lists**
    - **Slicing**

[start index included : last index excluded : specify pace]

Index	0	1	2	3	4	5	6	7	8	9
List item	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Negative Index	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
numbers[::2]
```

```
[1, 3, 5, 7, 9]
```

```
numbers[2:8:2]
```

```
[3, 5, 7]
```

```
numbers[::-1]
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

- Sections of lists can be replaced as follows:

```
numbers[0:3] = ["dog", "cat"]
numbers
```

```
['dog', 'cat', 4, 5, 6, 7, 8, 9, 10]
```

Ulster University

Python Fundamentals

- Built-In Data Structures
  - **Tuples**
    - Similar to lists, however: Defined with parenthesis or no brackets.

```
t = ("red","green","blue")
```

```
t = "red", "green", "blue"
```

      - Are **Immutable** (They can not be changed once created)
    - Slicing and indexing functions as in Lists.

```
In [15]:
1 thistuple = ("apple", "banana", "cherry")
2 print(thistuple[1])

banana
```

```
In [16]:
1 thistuple = ("apple", "banana", "cherry")
2 thistuple[1] = "blackcurrant" # test changeability
3 print(thistuple)

TypeErrorTraceback (most recent call last)
<ipython-input-16-c942b1dda348> in <module>()
      1 thistuple = ("apple", "banana", "cherry")
----> 2 thistuple[1] = "blackcurrant" # test changeability
      3 print(thistuple)

TypeError: 'tuple' object does not support item assignment
```

Ulster University

Python Fundamentals

- Built-In Data Structures
  - **Sets**
    - Unordered collection of **unique** items. Will not store duplicates.

```
primes = {2,3,5,7}
odds = {1,3,5,7,9}
```

    - Useful operations including:
      - **Union** of sets – containing items appearing in either

```
primes | odds
primes.union(odds)
```

```
{1, 2, 3, 5, 7, 9}
```

      - **Intersection** of sets – containing items appearing in both

```
primes & odds
primes.intersection(odds)
```

```
{3, 5, 7}
```

      - **Difference** in sets – containing items in set 1 but not set 2.

```
primes - odds
primes.difference(odds)
```

```
{2}
```

      - **Symmetric Difference** in sets – items appearing in only one set.

```
primes ^ odds
primes.symmetric_difference(odds)
```

```
{1, 2, 9}
```

Ulster University

Python Fundamentals

- Built-In Data Structures
  - **Dictionaries**
    - Unordered mappings of keys to values
    - Mutable
    - Created by comma-separated lists of *key: value* pairs

```
numbers = {'one':1,'two':2,'three':3}
```

```
{'one': 1, 'three': 3, 'two': 2}
```

    - Items are accessed using the list indexing syntax
    - However, the index is a key in the dictionary, not an integer:

```
numbers['two']
```

```
2
```

      - New entries can be added via:

```
numbers['four'] = 4
```

```
{'four': 4, 'one': 1, 'three': 3, 'two': 2}
```

Ulster University

Python Fundamentals

- Variables and mutable objects
  - **Mutable object** = object whose value can change
  - E.g. **list, set, dict**
  - **Note:** Be aware when using two variables to point at the same *mutable* object! e.g.

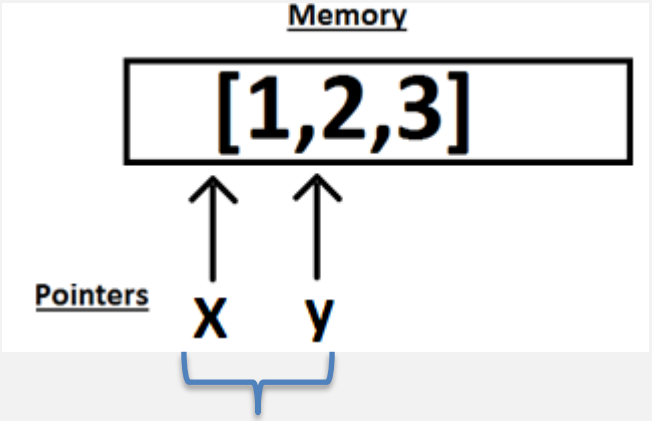
```
x = [1,2,3]
y = x         #x and y now both point to the same object which contains [1,2,3]
y.append(4)   #x and y now both point to the same object which contains [1,2,3,4]
x
```

```
[1, 2, 3, 4]
```

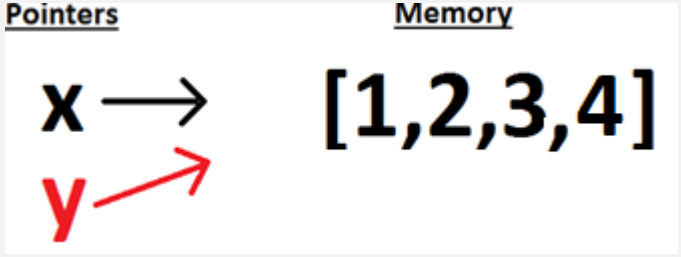
- Modifying y has appeared to modify x. This won't make sense unless we remember that python **variables are pointers to objects, not the objects themselves**

Ulster University

Python Fundamentals



- *x* and *y* are pointing to the same memory location
- So this memory location can be accessed either using *x* or *y*
- Therefore, *y.append*(4) is adding 4 to the list



Ulster University

Python Fundamentals

- Variables and **immutable** objects
  - **Immutable object** – object whose value is unchangeable once it is created
  - Int, float, bool, string, tuple, and other simple types
  - This makes arithmetic operations perform as expected, e.g.

```
x = 5
y = x
x = x * 2
print("x = ",x)
print("y = ",y)
```

```
x =  10
y =  5
```

  - When we call *x = x * 2*, we are not modifying the value of the 5 object, instead we are changing which object *x* points to.

Ulster University

Python Fundamentals

Mutable vs Immutable

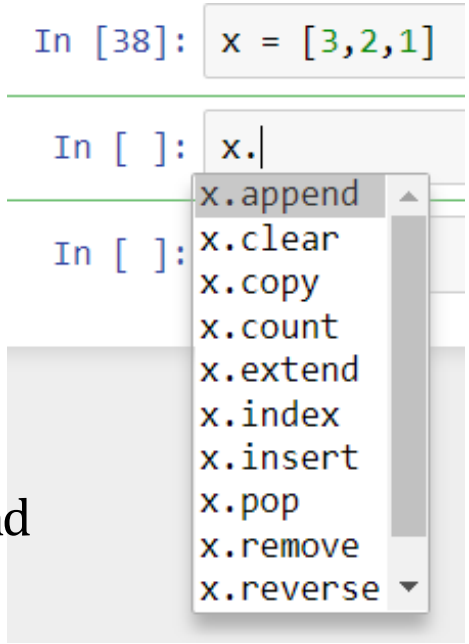
- int, float, bool, str, tuple, unicode are immutable
- list, set, dict are mutable

Check the difference between variable holding mutable object and when variable holding immutable objects what happens?

Ulster University

Python Fundamentals

- Objects in Python
  - Everything is an object!
  - Objects have:
    - *attributes* (metadata)
    - *methods (*functionality*)*
  - These are accessed using the dot syntax(.)
  - Tab auto complete helps identify available attributes and methods
  - **Note:**
    - In Jupyter Notebook, you must ensure the variable assignment has been executed prior to using tab autocomplete
      - Otherwise no suggestions, or incorrect suggestions, will be given



Ulster University

Python Fundamentals

- **Operators**
  - Arithmetic Operators
  - Assignment Operators
  - Comparison Operators
  - Boolean Operators
  - Identity & Membership Operators

Ulster University

Python Fundamentals

- **Arithmetic**

Operator	Name	Description	Example
a + b	Addition	Sum of a and b	2 + 3 = 5
a – b	Subtraction	Difference of a and b	2 – 3 = -1
a* b	Multiplication	Product of a and b	2 * 3 =6
a/ b	True Division	Quotient of a and b	2 / 3= 0.6666..
a// b	Floor Division	Quotient of a and b, without fractional parts	2 // 3 =0
a%b	Modulus	Remainder after division of a by b	2%3 = 2
a** b	Exponentiation	a raised to the power of b	2**3 =8
-a	Negation	The negative of a	a = 2, -a outputs -2

Ulster University

Python Fundamentals

- **Assignment Operators**
  - Python contains built-in update operators for all the arithmetic operations.

```
a = a + 2  #instead of this,
a += 2     #we can do this.
```

  - Other examples:
    - a-= b
    - a*= b
    - a/= b
    - etc.

Ulster University

Python Fundamentals

- **Comparison Operators**
  - To compare different values, returning *True* or *False*.

Operator	Description	Example
a == b	a equal to b	2==2 returns *True*
a != b	a not equal to b	2==3 returns *True*
a < b	a less than b	1<2 returns *True*
a > b	a greater than b	2>1 returns *True*
a <= b	a less than or equal to b	1 <= 2 returns *True*
a >= b	a greater than or equal to b	2 >= 1 returns *True*

- They can be combined for more complicated comparisons, e.g.

```
a = 120

40 < a < 125 #Check if a is between 40 and 125
```

```
True
```

Ulster University

Python Fundamentals

- **Boolean Operators**
  - To compare the values of Boolean statements
  - Useful in control flow
  - ***and*, *or*, *not***:

```
x=10
(x>1) and (x<9)
```

```
False
```

```
(x>1) or (x<9)
```

```
True
```

```
not (x<9)
```

```
True
```

Ulster University

Python Fundamentals

- **Identity and Membership Operators**

Category	Operator	Description
Identity	a **is** b	True if a and b are identical objects
	a **is not** b	True if a and b are not identical objects
Membership	a **in** b	True if a is a member of b
	a **not in** b	True if a is not a member of b

- Identity and Equality are not the same

```
a = [1,2,3]
b = [1,2,3]
```

```
a==b #These objects ARE equal
```

```
True
```

```
a is b #These objects do not have the same identity / are not identical
```

```
False
```

```
a=b
a is b #The objects that a and b point to are identical (they are pointing to the same object)
```

```
True
```

Ulster University

Python Fundamentals

- **Membership Operators**
  - Check for membership within compound objects

```
'elephant' in ['giraffe','elephant','computer']
```

```
True
```

```
7 not in [4,3,1]
```

```
True
```

- This is distinctly easier to use than other languages which may require the creation of loops and equality checks

Ulster University

Python Fundamentals

- **Built-In Types – Simple Values**

Type	Example	Description
int	x = 1	Integers (whole numbers)
Float	x = 1.0	Floating-point numbers (real numbers)
Complex	x = 1 + 2j	Complex numbers (real + imaginary components)
Bool	x = True	Boolean: True/False values
Str	x = 'hello'	String: Characters or Text
NoneType	x = None	Special object indicating nulls.

- **Integers**
  - Any number without a decimal point
  - Division returns a floating-point type (Python3.x)

```
5/2
2.5
```

Ulster University

Python Fundamentals

- **Built-In Types – Simple Values**
  - **Floating-Point Numbers**
    - Store fractional numbers – either decimal notation (0.000005) or exponential notation (5e-6)
    - Can cast from int to float using:

```
float(5)
```

```
5.0
```

  - **Strings**
    - Created with 'single' or "double" quotes
    - Many useful string functions are builtin, including:

Function	Description
len(s)	Length of s
s.upper()	Make s uppercase
s.lower()	Make s lowercase
s.capitalize()	Capitalise s
s+s	Concatenation
s[0]	Access the first character in s

Ulster University

Python Fundamentals

- **Built-In Types – Simple Values**
  - Boolean Type
    - Two possible values – *True* and *False*
    - Returned by comparison operators.
    - **Note:** They are case sensitive! *True* and *False* must be capitalised
    - The *bool()* object constructor can be used to generate Boolean values based on these rules:

Type	Result	Example
Integer/Float	True if not 0 False if 0	bool(0) returns False
None	False	bool(None)
Strings	True if not empty False if empty	bool("") returns False
Sequences	True if not empty False if empty empty	bool([]) returns false

Ulster University

Python Fundamentals

- Built-In Types – Simple Values
  - **None Type**
    - A special type with only one possible value: *None*
    - Used to refer to null objects
    - Often the default return value for functions that do not return any value

Ulster University

Getting Help from Documentation

- Use Jupyter Notebook get information on the variousclasses you are using. For example, type:
- *<className>?* - Useful information about a classor collection, e.g. docstrings, function call arguments, constructors, etc.
- *<className>??* - Further information, such as source code.
- **Note: If using QtConsole, Press 'q' to exit the further information page**

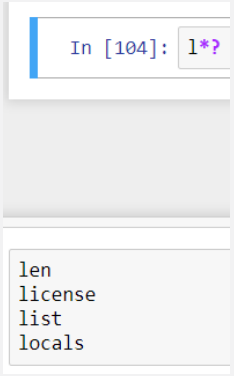
```
In [107]: list?
```

```
Init signature: list(self, /, *args, **kwargs)
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
Type:           type
```

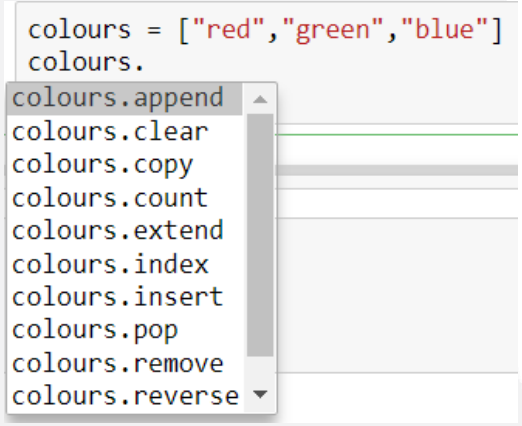
Ulster University

Getting Help from Documentation

- **Wildcards** can also be used to search for objects in the current namespace. E.g. *l*?*



- **Tab Completion** provides a way to explore the structure / **attributes** of any object you are using. This also works with **filenames** and **directory** structures.



Ulster University

Getting Help from Documentation

- Information about a specific object can be obtained by:

```
In [103]: colours = ["red","green","blue"]
          colours.append("Yellow")
          colours?
```

```
Type:        list
String form: ['red', 'green', 'blue', 'Yellow']
Length:      4
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

- Press Shift + Tab in Jupyter Notebook within function parameters to view the function signature:

```
In [103]: colours = ["red","green","blue"]
          colours.sort()
```

```
Docstring: L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
Type:      builtin_function_or_method
```

Ulster University

Python Fundamentals

- Control Flow
  - Conditional statements are used to execute certain blocks of code only when particular conditionsare met.
  - **If Statement**:

```
x=False

if x == True:
    print("This will not be printed")
elif x == False:
    print("This will be printed")
else:
    print("this will not be printed")
```

Ulster University

Python Fundamentals

- **Loops**
  - Repetitions often used to process collections
  - **For Loop:**

```
for x in range(1,5):
    print(x)
```

```
1
2
3
4
```

```
words = ['giraffe','helicopter','elephant']
for w in words:
        print(w, len(w))
```

```
giraffe 7
helicopter 10
elephant 8
```

  - Optional Keywords:
    - Continue – Skip the remainder of the current loop and continue to the next iteration
    - Break – Exit the loop entirely

Ulster University

Python Fundamentals

- **Loops**
  - **While Loop**: When the number of iterations is notknown

```
while x < 100:
    x=x*x
    print(x)
```

```
4
16
256
```

Ulster University