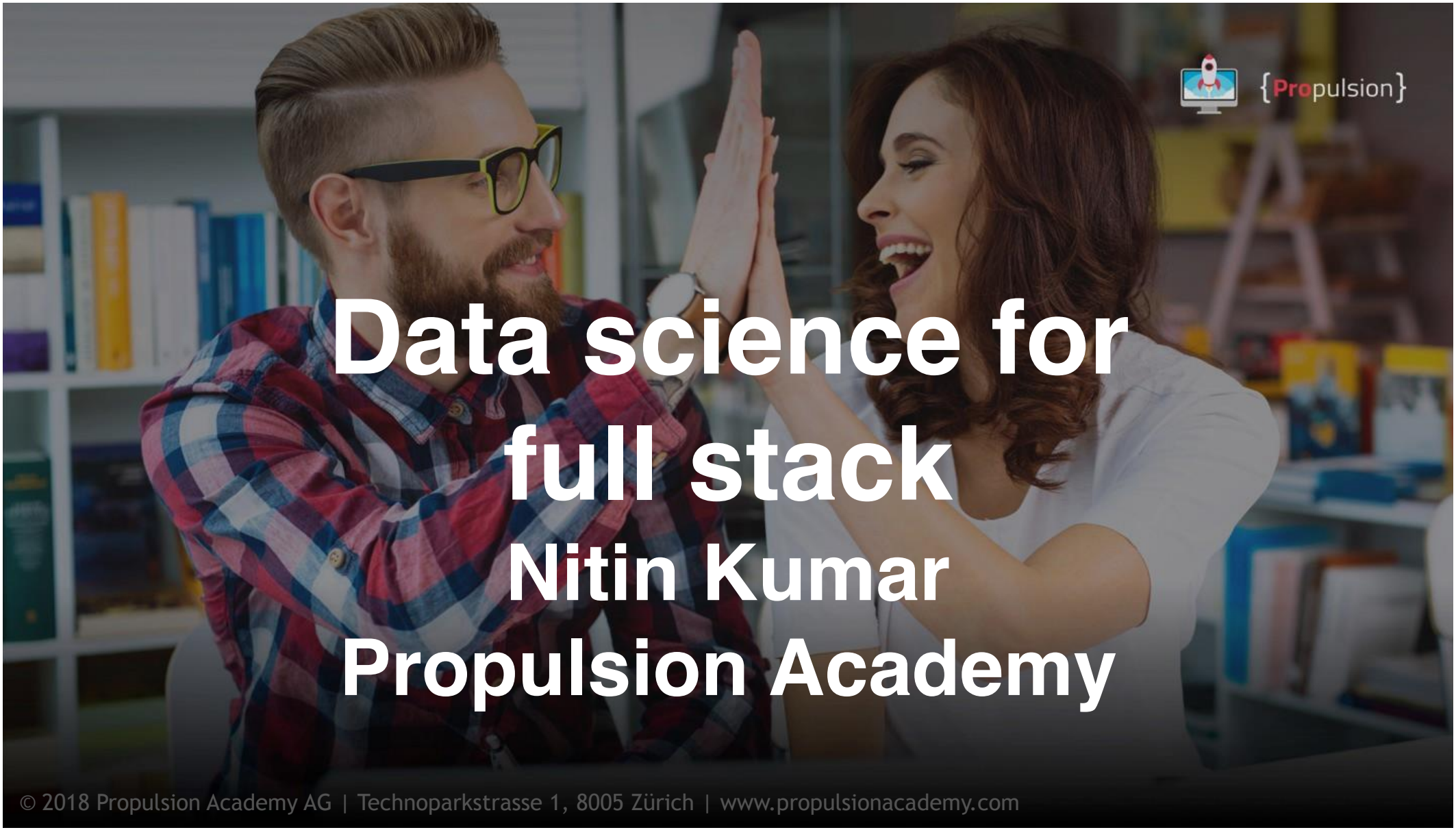


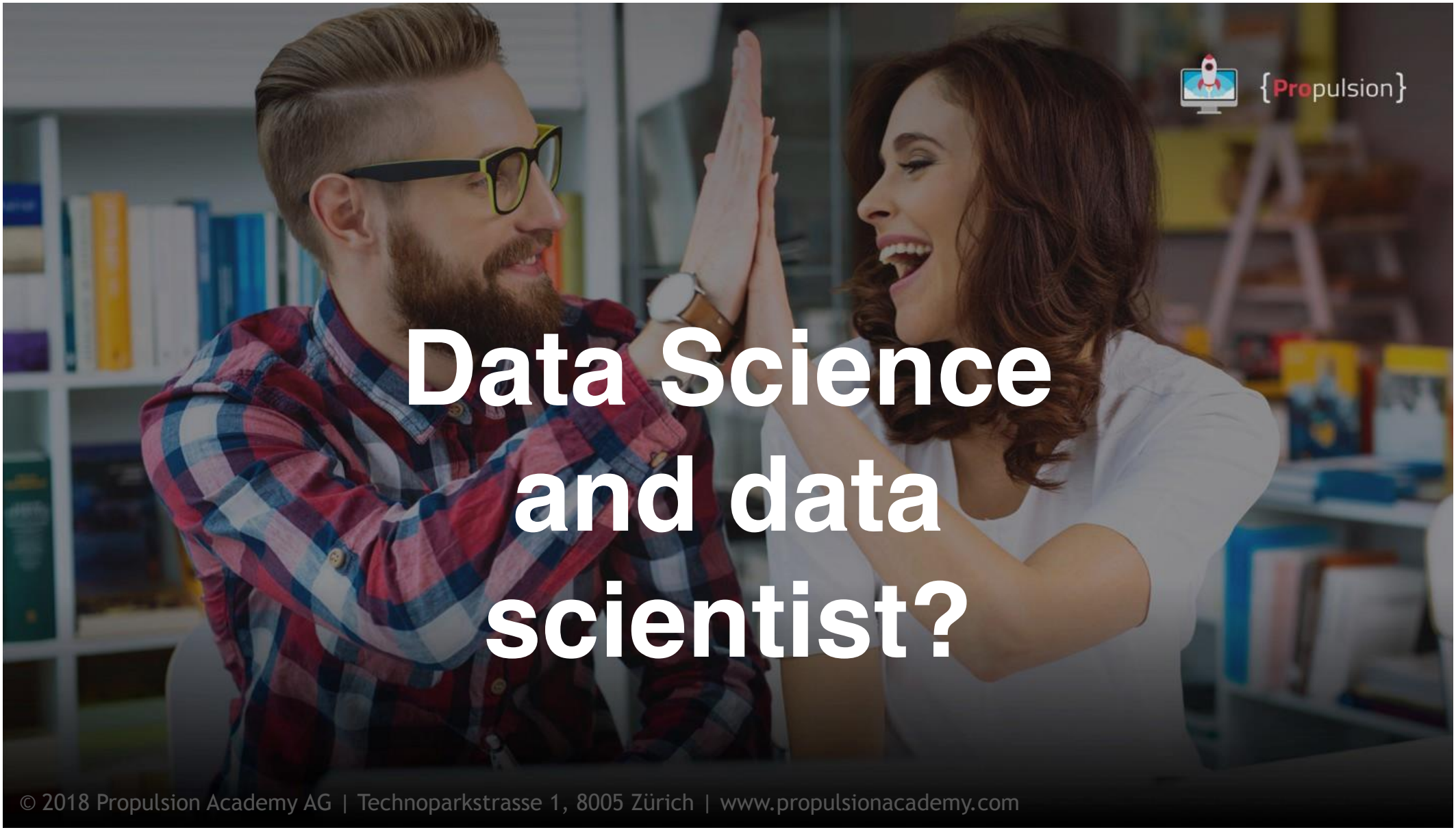
Data science for full stack

Nitin Kumar
Propulsion Academy

Today's plan

{Propulsion}

- Data science and data scientist?
- Python for data science
- Pandas
- Exercise - Data exploration



{Propulsion}

DATA

Data Scientist: The Sexiest Job of the 21st Century

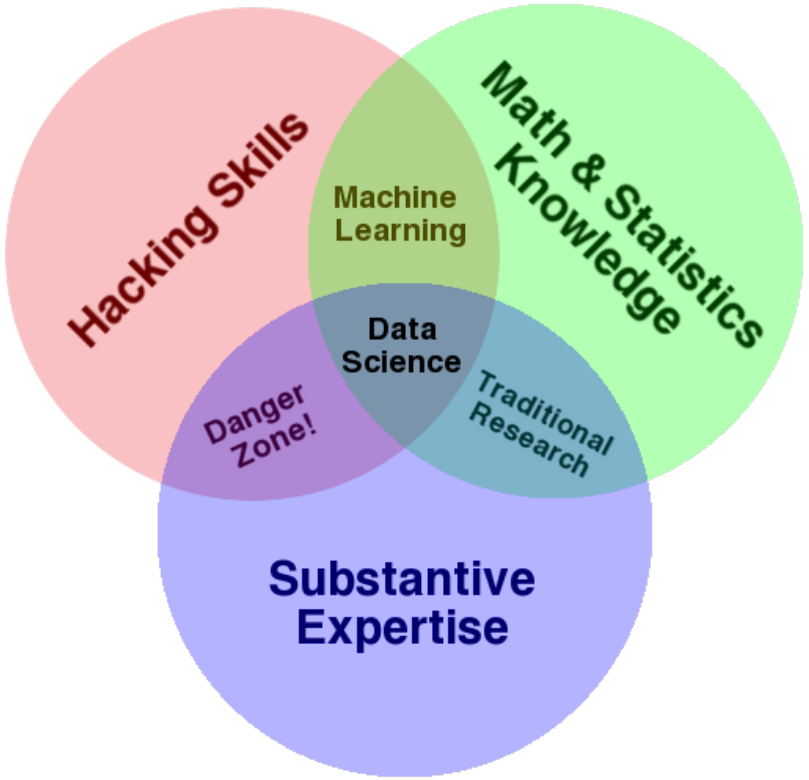
by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

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

Getting the buzzwords right

- Data Science
- Machine Learning
- Artificial Intelligence
- Big Data

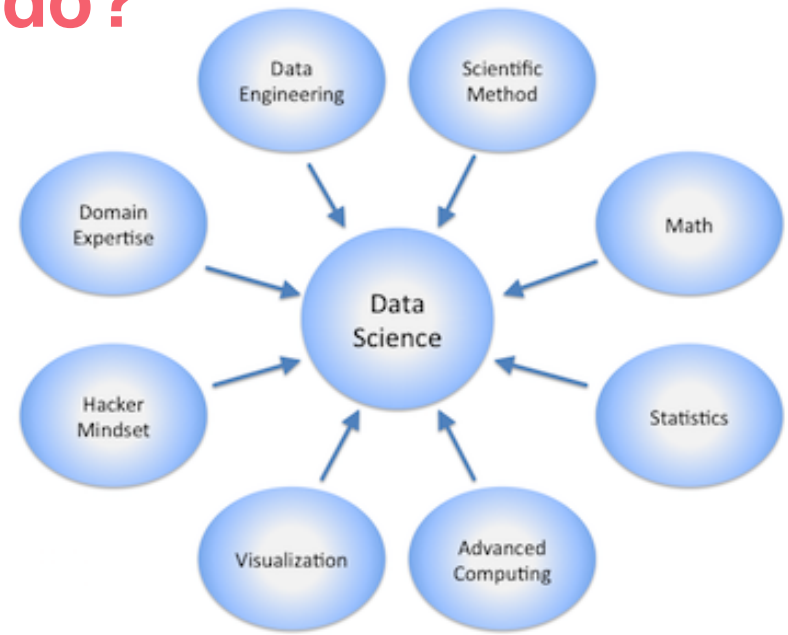


How do these all differ?

So what does a data scientist do?

“4 pillars”:

- Business domain
- Statistics and probability
- Computer science and programming
- Written and oral communication



Data scientist: person strong on *several* of these pillars who can leverage existing data sources and create new ones to extract meaningful information and actionable insights.

The “science” in data science

Data science is the application of the scientific method to problems that can be solved through the analysis and modelling of data.

Workflow

- Ask a question and/or define a problem
- Collect and leverage data to come up with answers/solutions
- Test your solutions to see if the problem is solved
- Iterate as needed, and finalise the solution.

Examples of data science problems/deliverables

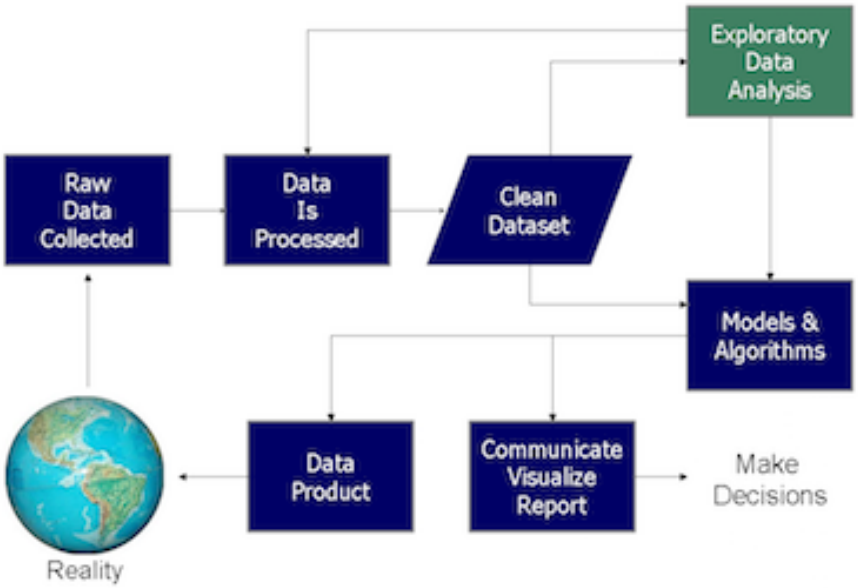
{Propulsion}

- 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)
- Anomaly detection (e.g., fraud detection)
- Recognition (image, text, audio, video, facial, …)

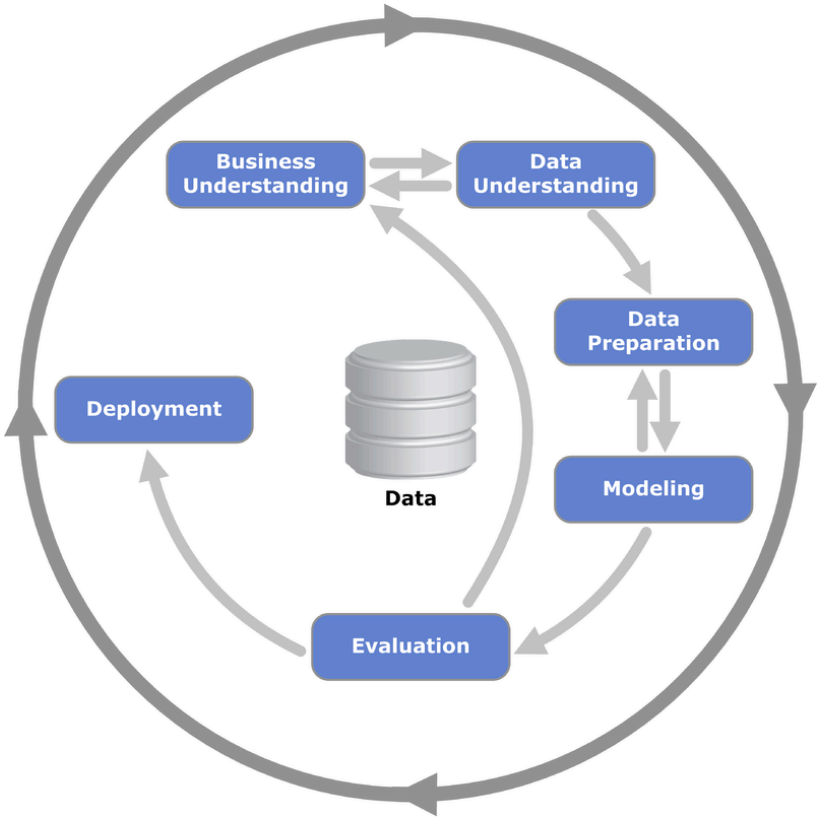
- Actionable insights (via dashboards, reports, visualizations, …)
- Automated processes and decision-making (e.g., credit card approval)
- Scoring and ranking (e.g., FICO score)
- Segmentation (e.g., demographic-based marketing)
- Optimization (e.g., risk management)
- Forecasts (e.g., sales and revenue)

The data science process

- Data acquisition, collection, and storage
- Discovery and goal identification (ask the right questions)
- Access, ingest, and integrate data
- Processing and cleaning data (munging/wrangling)
- Initial data investigation and exploratory data analysis
- Choosing one or more potential models and algorithms
- Apply data science methods and techniques (e.g. ML)
- Measuring and improving results (validation and tuning)
- Delivering, communicating, and/or presenting final results
- Business decisions and/or changes are made based on the results
- Repeat the process to solve a new problem



Cross-industry standard process for data mining (CRISP-DM)

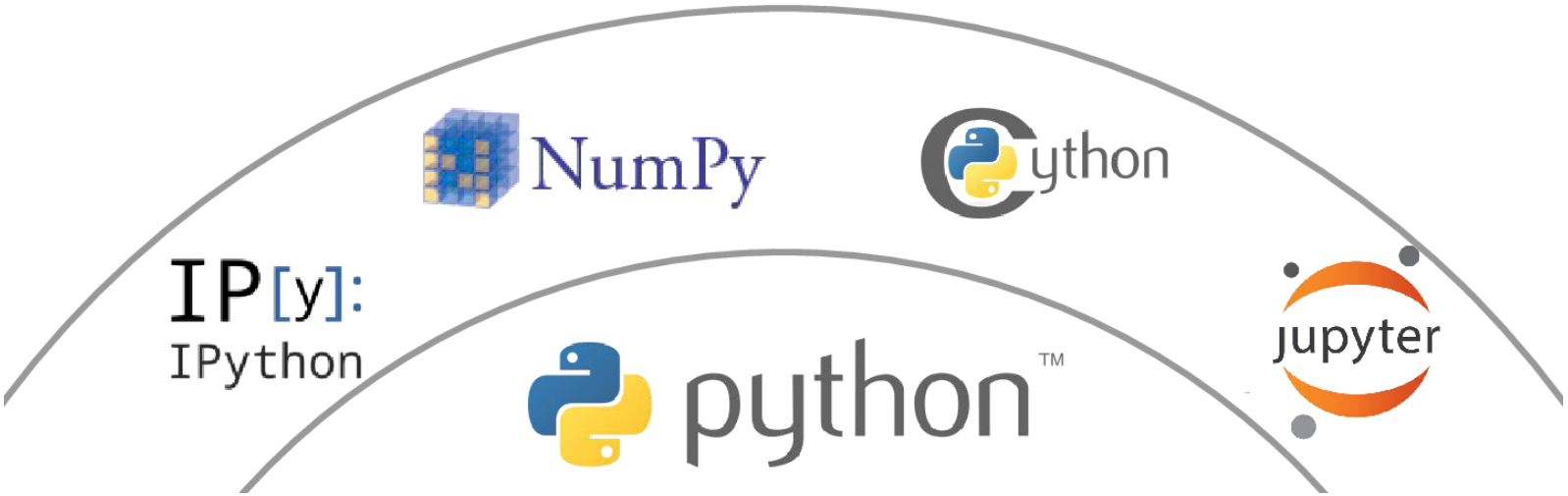


https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data


{Propulsion}
Python for
Data Science
© 2018 Propulsion Academy AG | Technoparkstrasse 1, 8005 Zürich | www.propulsionacademy.com

Python's Data Science Stack

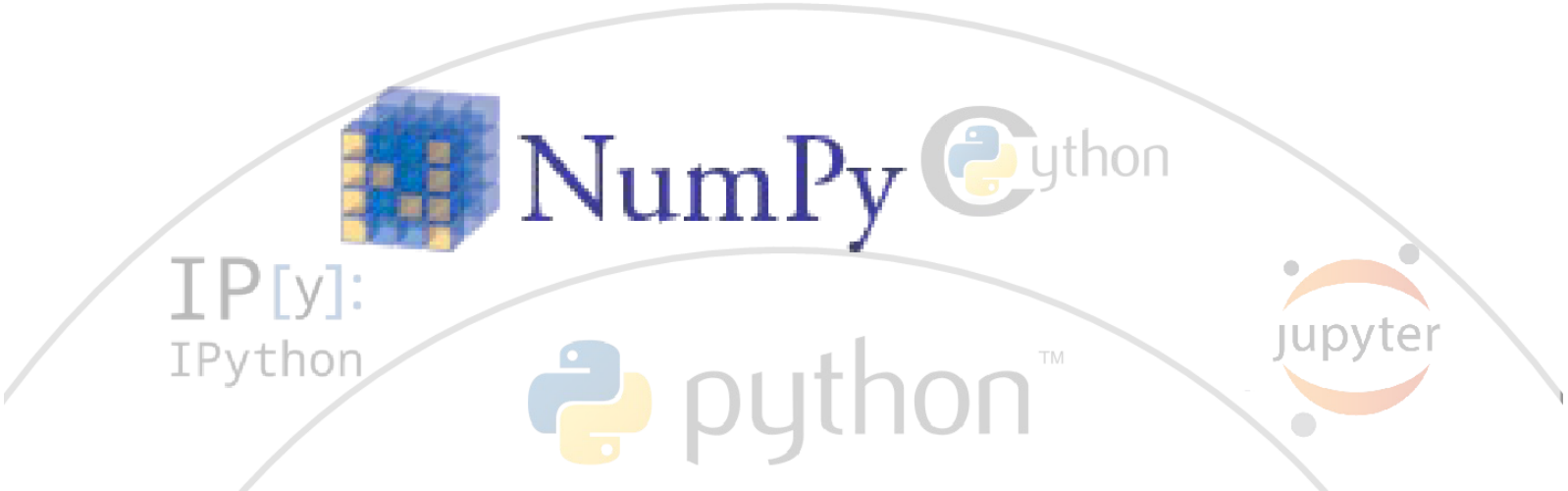
{Propulsion}



NumPy = Numerical Python

{Propulsion}

Efficient array storage, manipulation, and computation

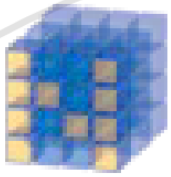


NumPy = Numerical Python

{Propulsion}

Efficient array storage, manipulation, and computation

```
In [1]: import numpy as np

        # Create a 5x5 uniform random matrix
        M = np.random.rand(5, 5)

        # Compute the SVD
        U, S, VT = np.linalg.svd(M)
        print(S)

        [ 2.46102945  0.94542853  0.53550015  0.20705388  0.13071452]
```



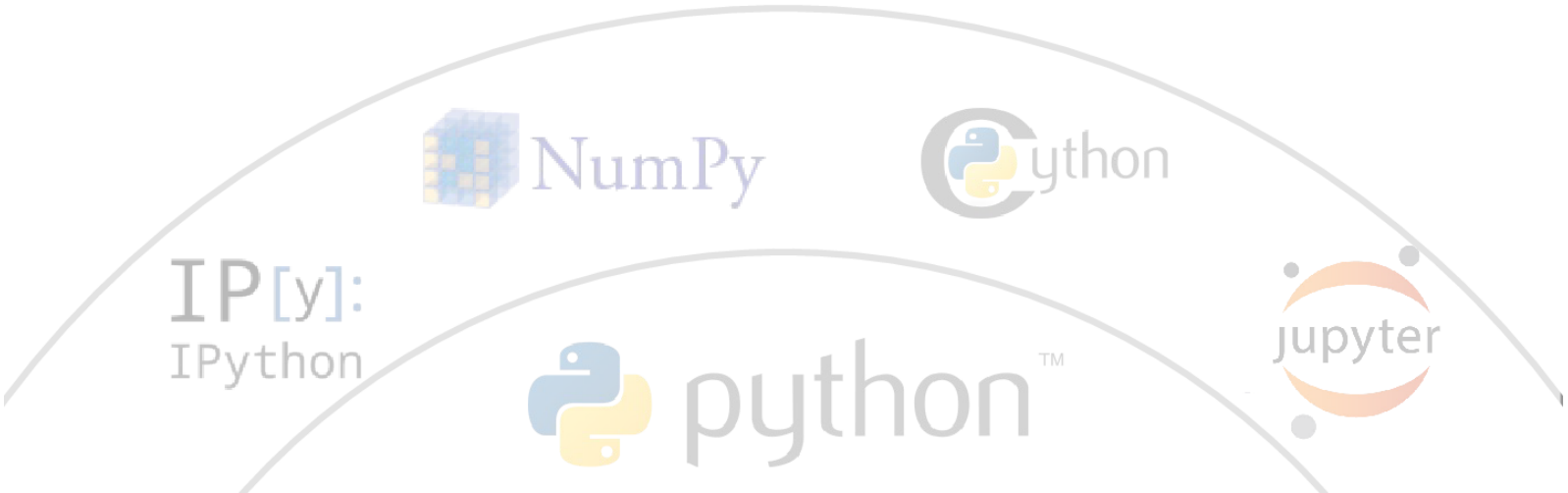
NumPy

ython

IP[y]:
IPython

python™

jupyter

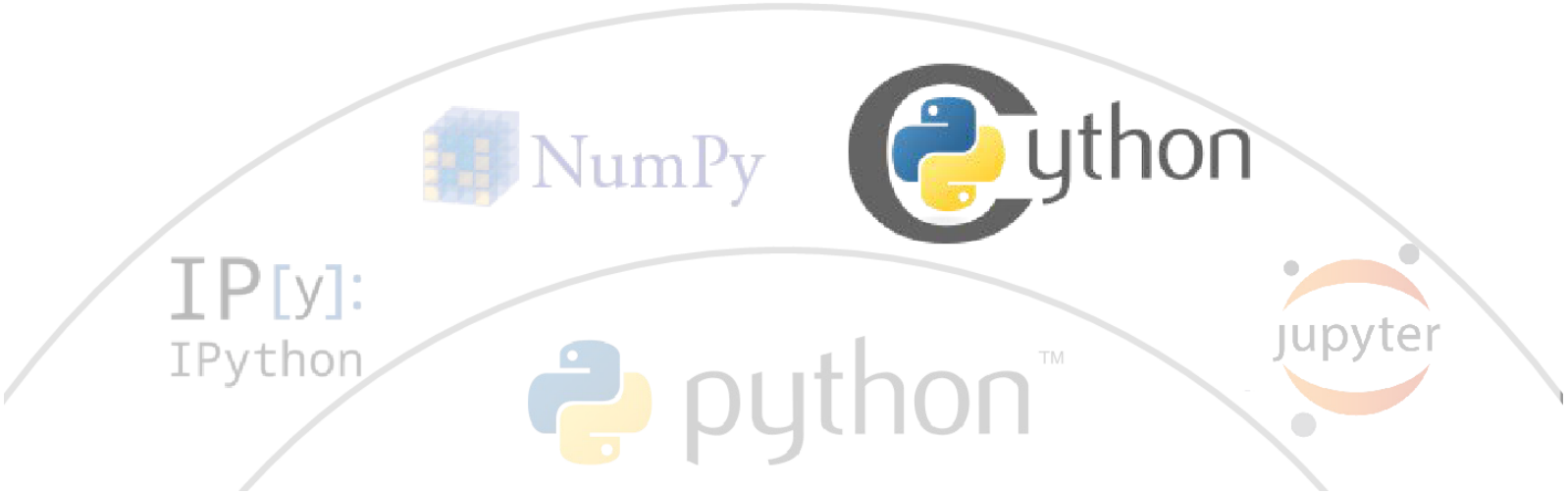


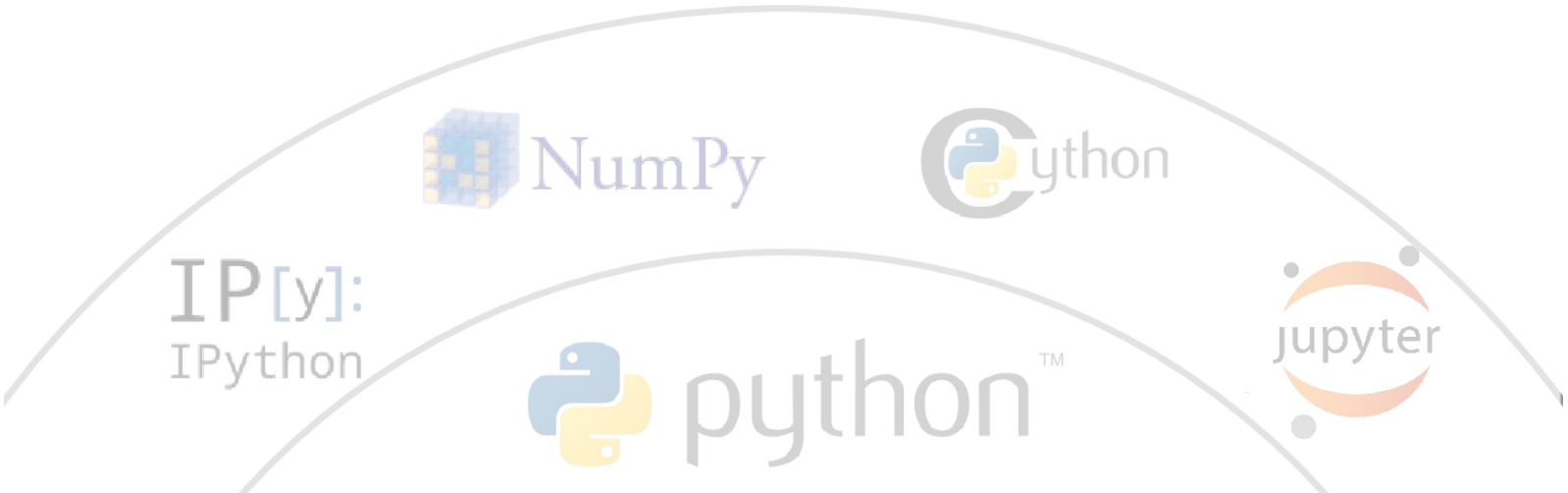
Cython = C + Python

{Propulsion}

Super-set of the Python language that allows easy interfacing with C & Fortran libraries (e.g. BLAS, LAPACK, etc.) and also fast Python code.

Drives many of the packages in the data science stack.

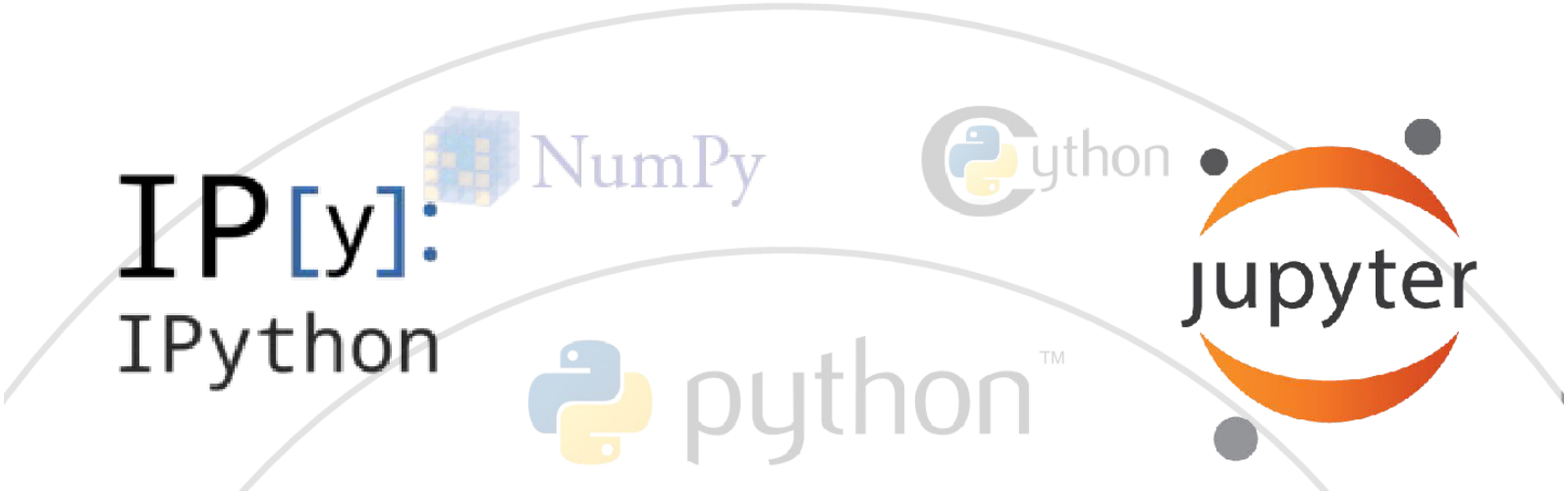




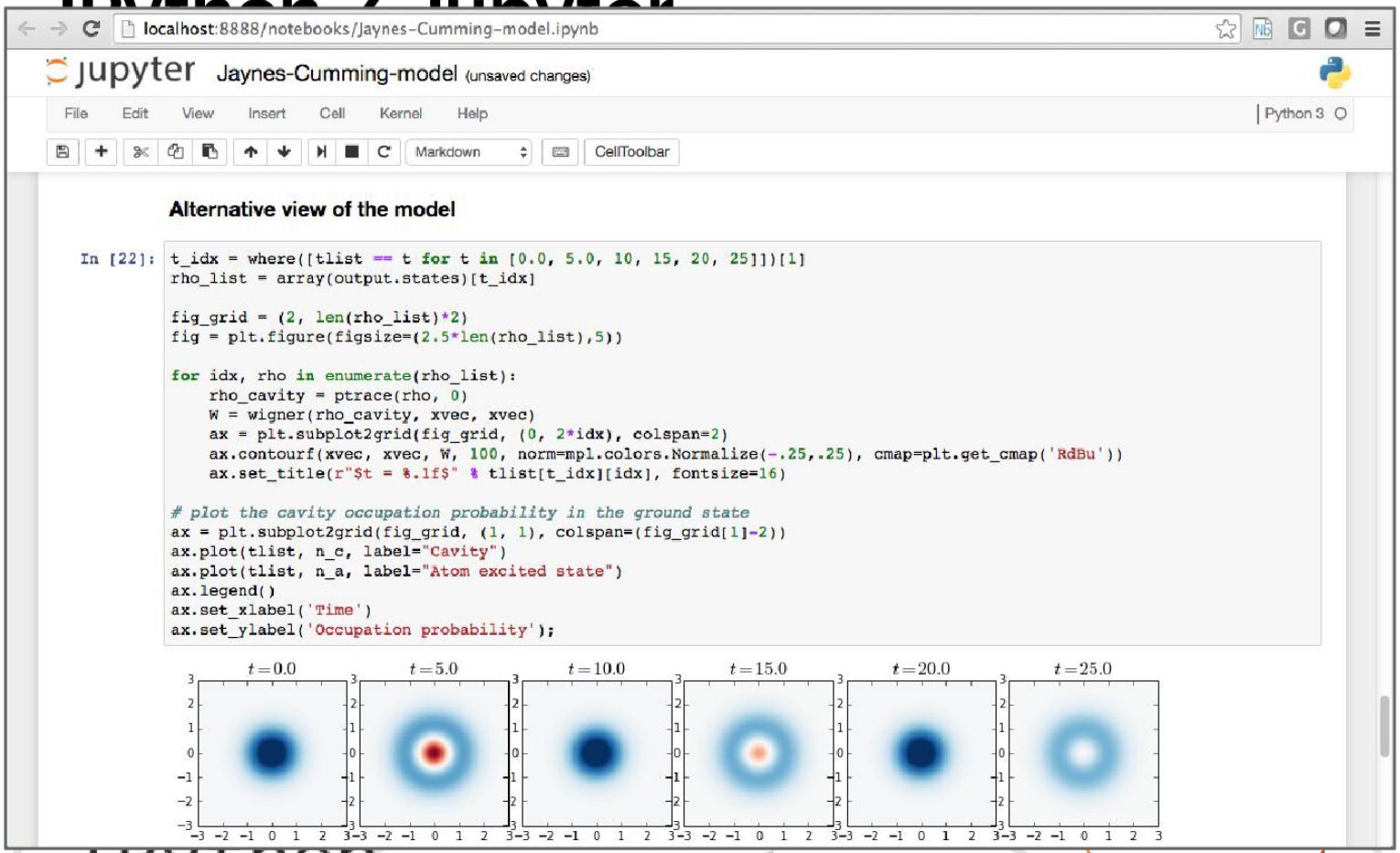
IPython / Jupyter

{Propulsion}

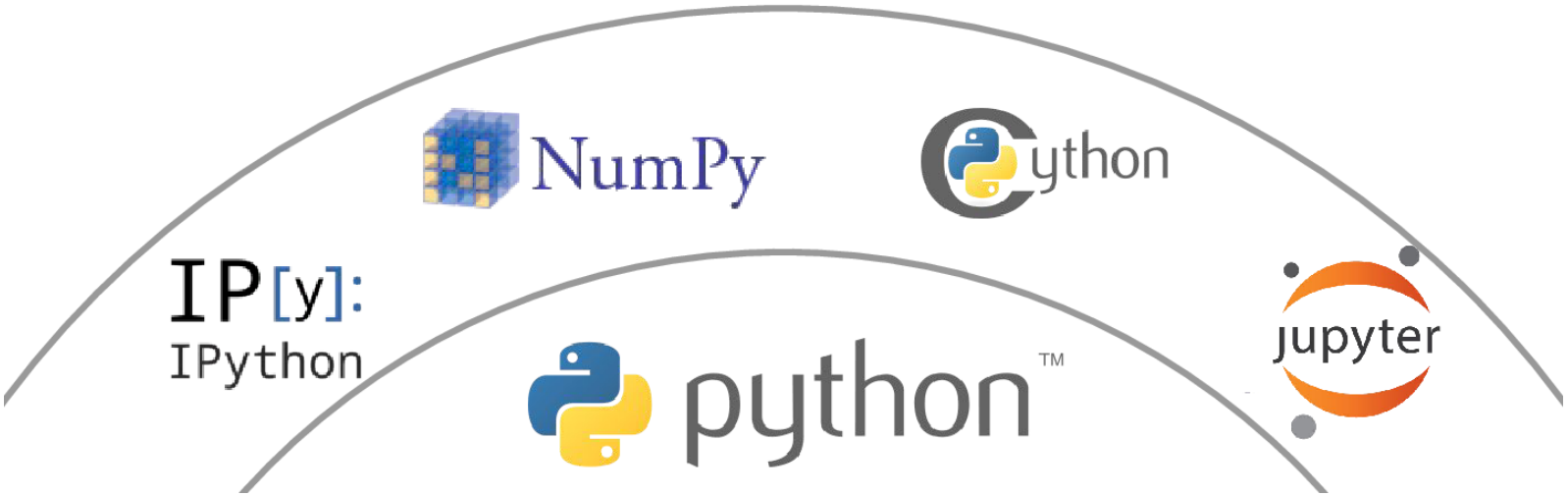
Terminal, development environment, Notebooks, and more for efficient use of Python in day-to-day work

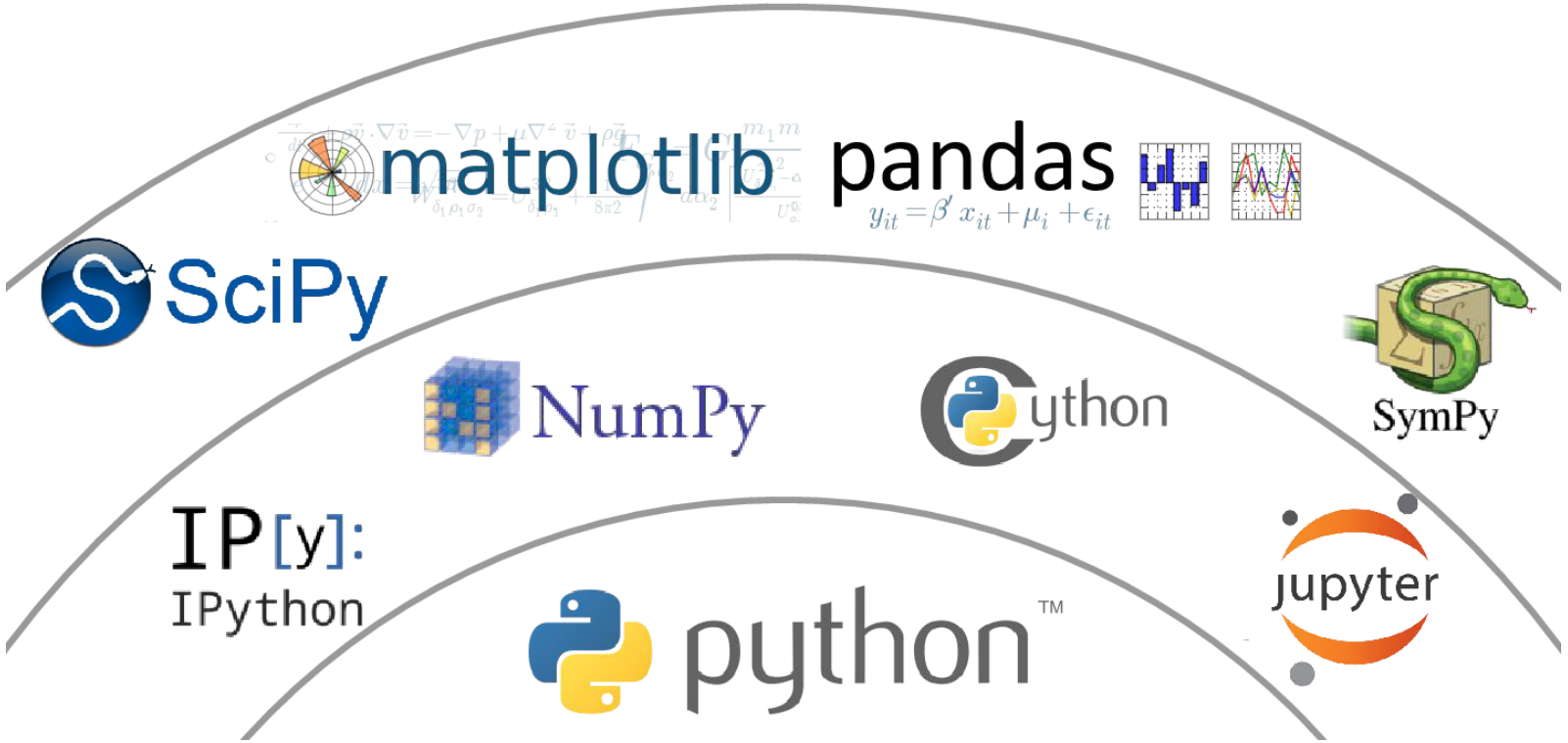


IPython / Jupyter



{Propulsion}

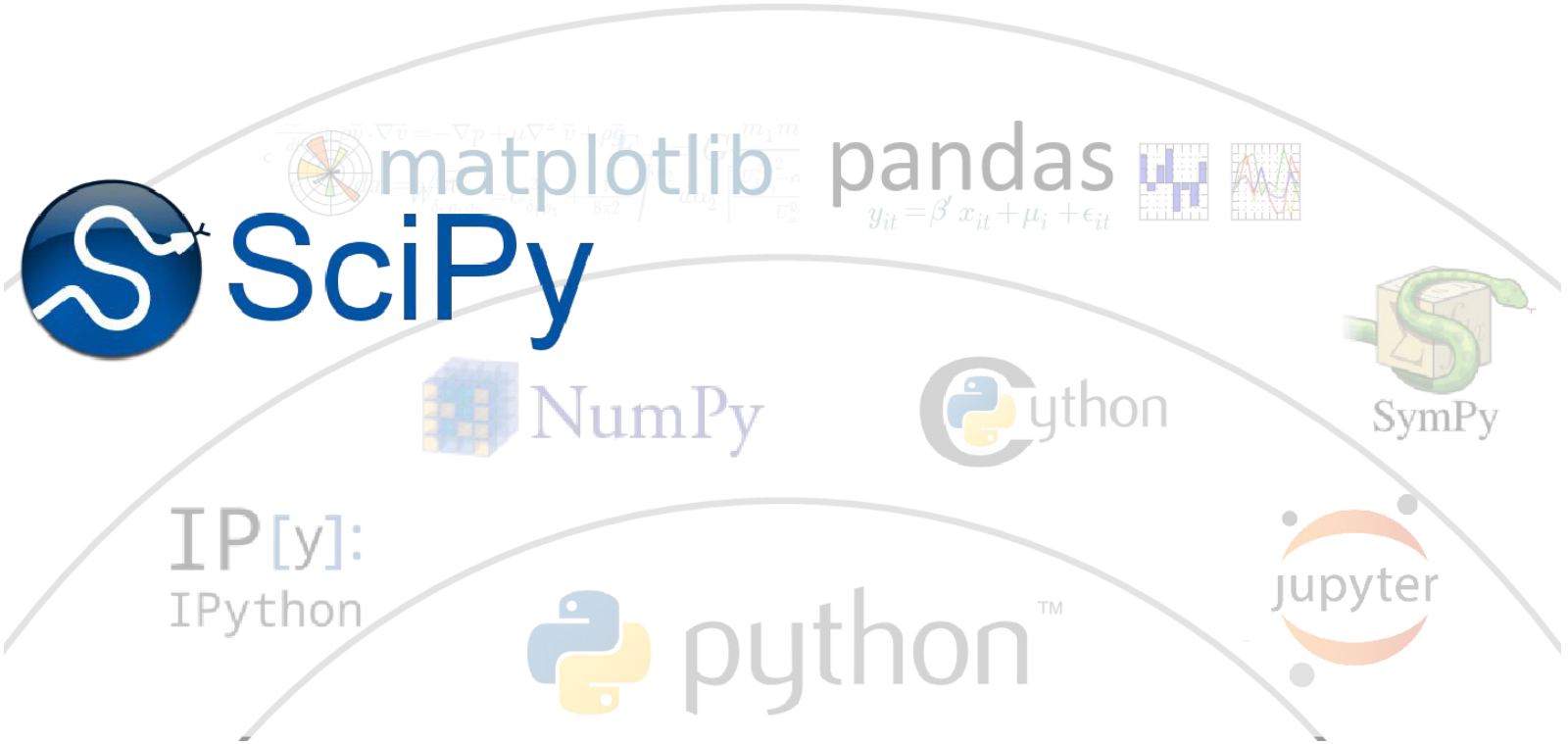




SciPy

{Propulsion}

Provides an interface to common scientific computing Tasks, including wrappers of many NetLib packages.



SciPy

Provi
Tasks

List from http://docs.scipy.org/doc/scipy/reference/

- Special functions (**scipy.special**)
- Integration (**scipy.integrate**)
- Optimization (**scipy.optimize**)
- Interpolation (**scipy.interpolate**)
- Fourier Transforms (**scipy.fftpack**)
- Signal Processing (**scipy.signal**)
- Linear Algebra (**scipy.linalg**)
- Sparse Eigenvalue Problems with ARPACK
- Compressed Sparse Graph Routines (**scipy.sparse.csgraph**)
- Spatial data structures and algorithms (**scipy.spatial**)
- Statistics (**scipy.stats**)
- Multidimensional image processing (**scipy.ndimage**)
- File IO (**scipy.io**)

{Propulsion}

IP[y]: IPython

python™

jupyter

Sympy

{Propulsion}

Library for symbolic computation: algebraic operations, differentiation & integration, optimization, etc.

Sympy

{Propulsion}

Library for symbolic computation: algebraic operations, differentiation & integration, optimization, etc.

Polynomials and rational functions

SymPy does not expand brackets automatically. The function expand is used for this.

In [6]:
```
a=(x+y-z)**6
a
```

Out[6]: $(x + y - z)^6$

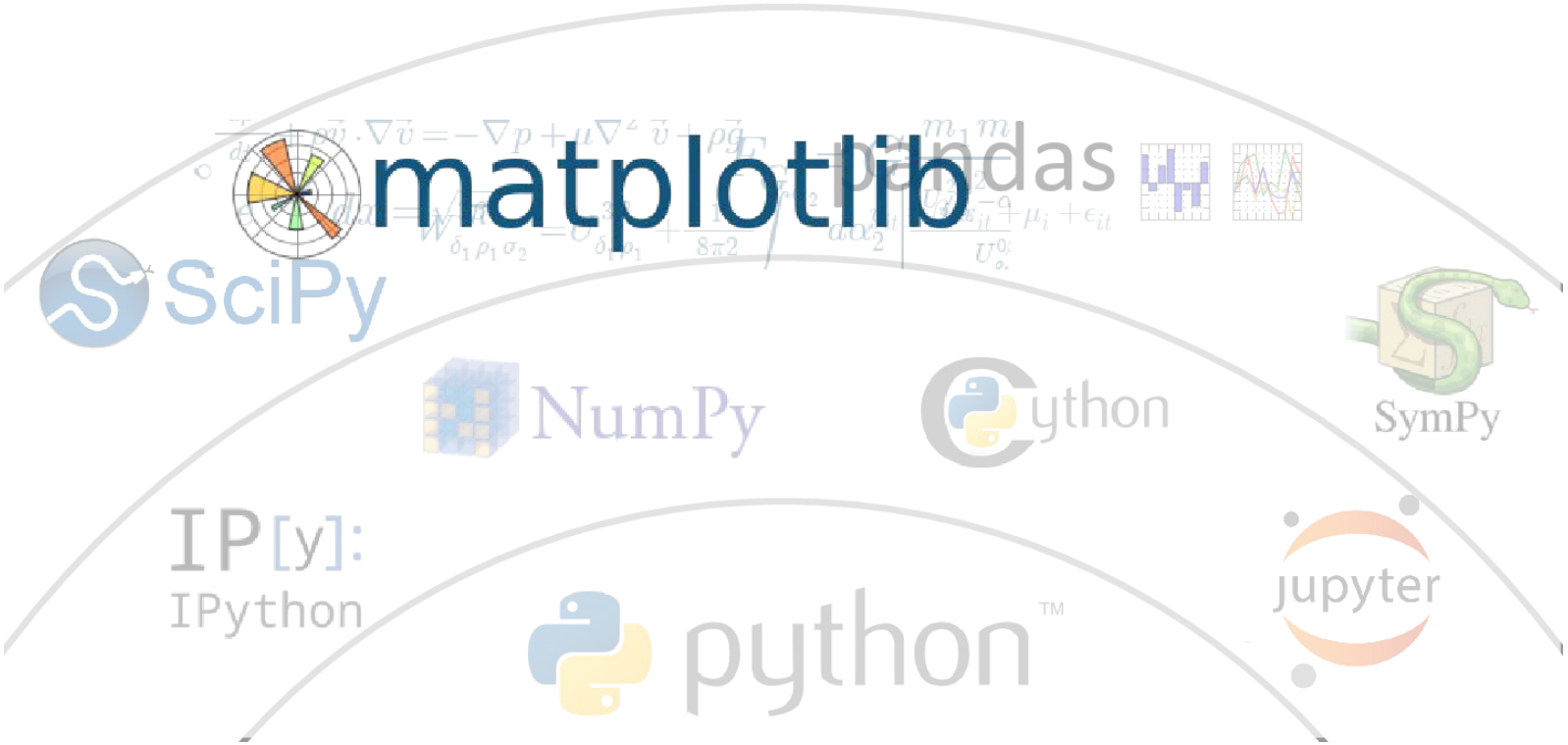
In [7]:
```
a=expand(a)
a
```

Out[7]: $$x^6 + 6x^5y - 6x^5z + 15x^4y^2 - 30x^4yz + 15x^4z^2 + 20x^3y^3 - 60x^3y^2z + 60x^3yz^2 - 20x^3z^3 + 15x^2y^4 - 60x^2y^3z + 90x^2y^2z^2 - 60x^2yz^3 + 15x^2z^4 + 6xy^5 - 30xy^4z + 60xy^3z^2 - 60xy^2z^3 + 30xyz^4 - 6xz^5 + y^6 - 6y^5z + 15y^4z^2 - 20y^3z^3 + 15y^2z^4 - 6yz^5 + z^6$$

matplotlib

{Propulsion}

Matlab-inspired plotting and visualization

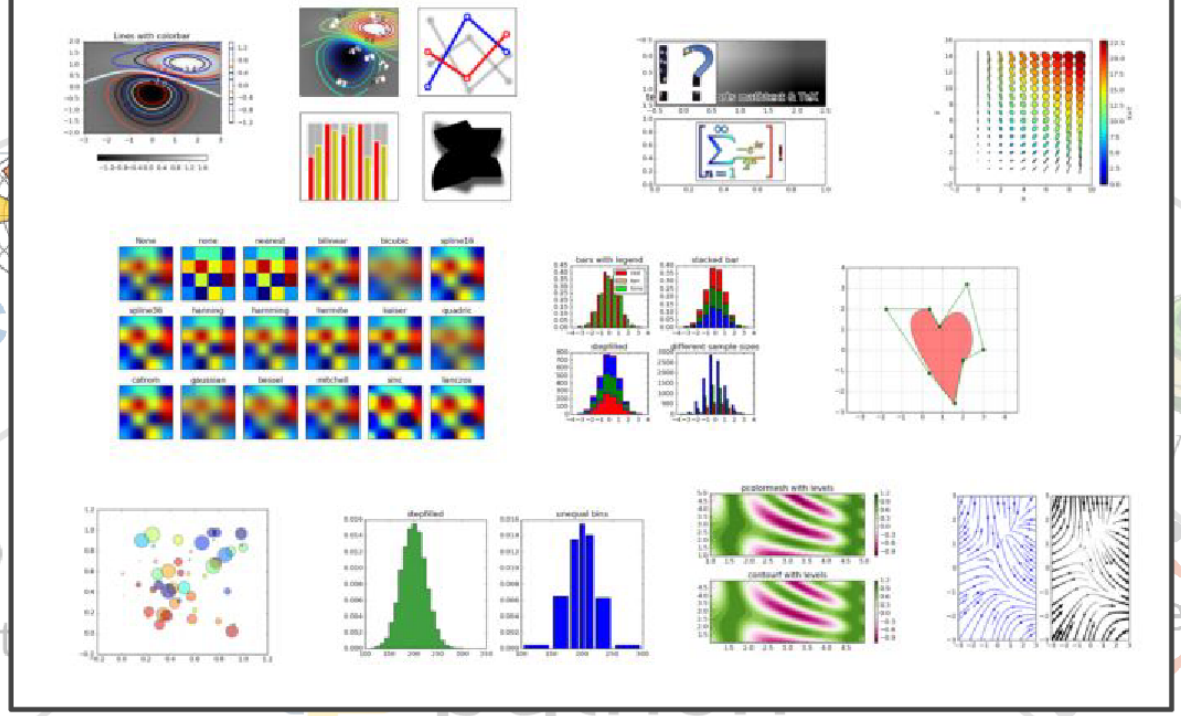


matplotlib

{Propulsion}

Matlab-inspired plotting and visualization

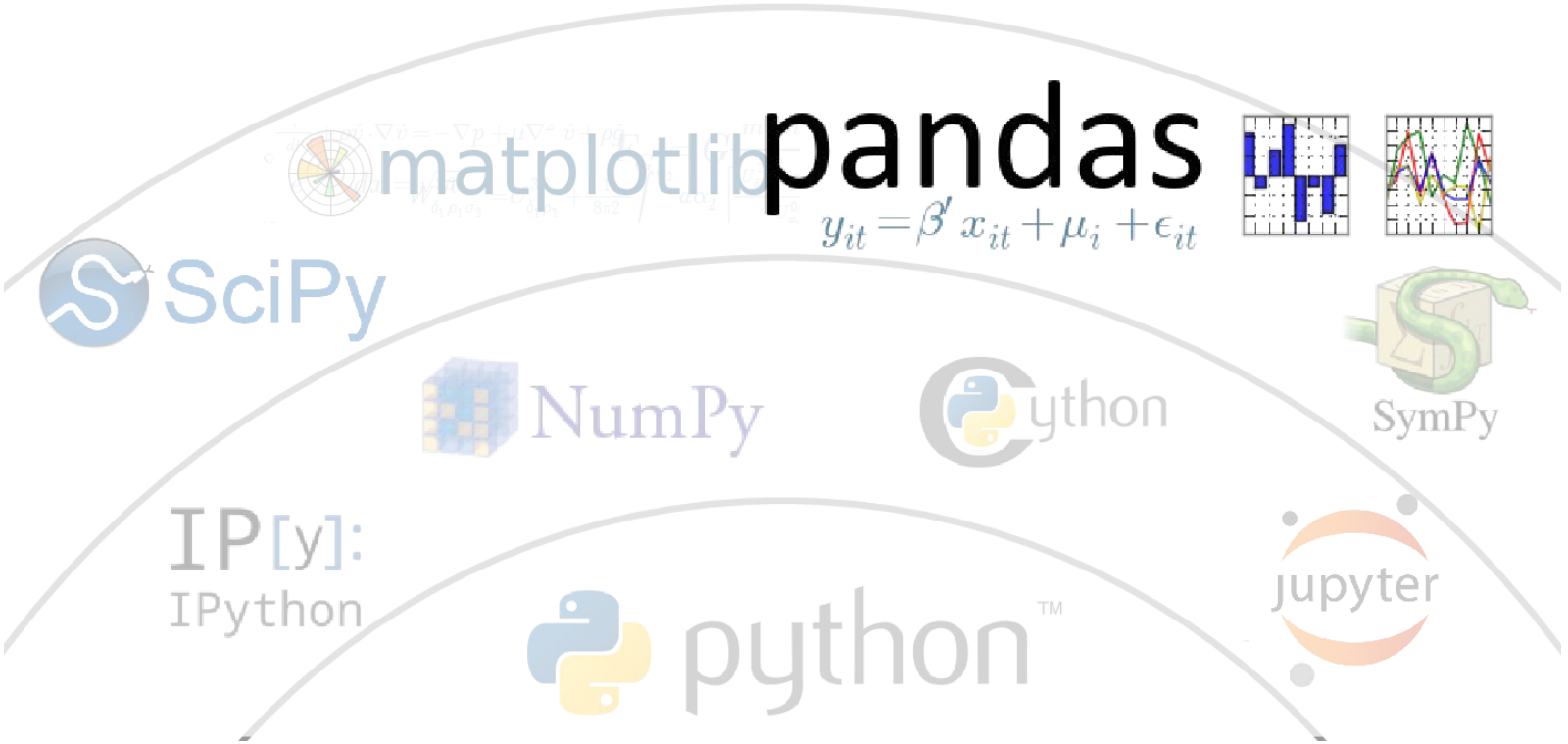
From http://matplotlib.org/gallery.html



Pandas

{Propulsion}

R-inspired DataFrames & associated functionality
(data munging & cleaning, group-by & transformations,
and much more)



Pa

```
In [1]: import pandas as pd
data = pd.read_csv('iris.csv')
data.head()
```

Out[1]:

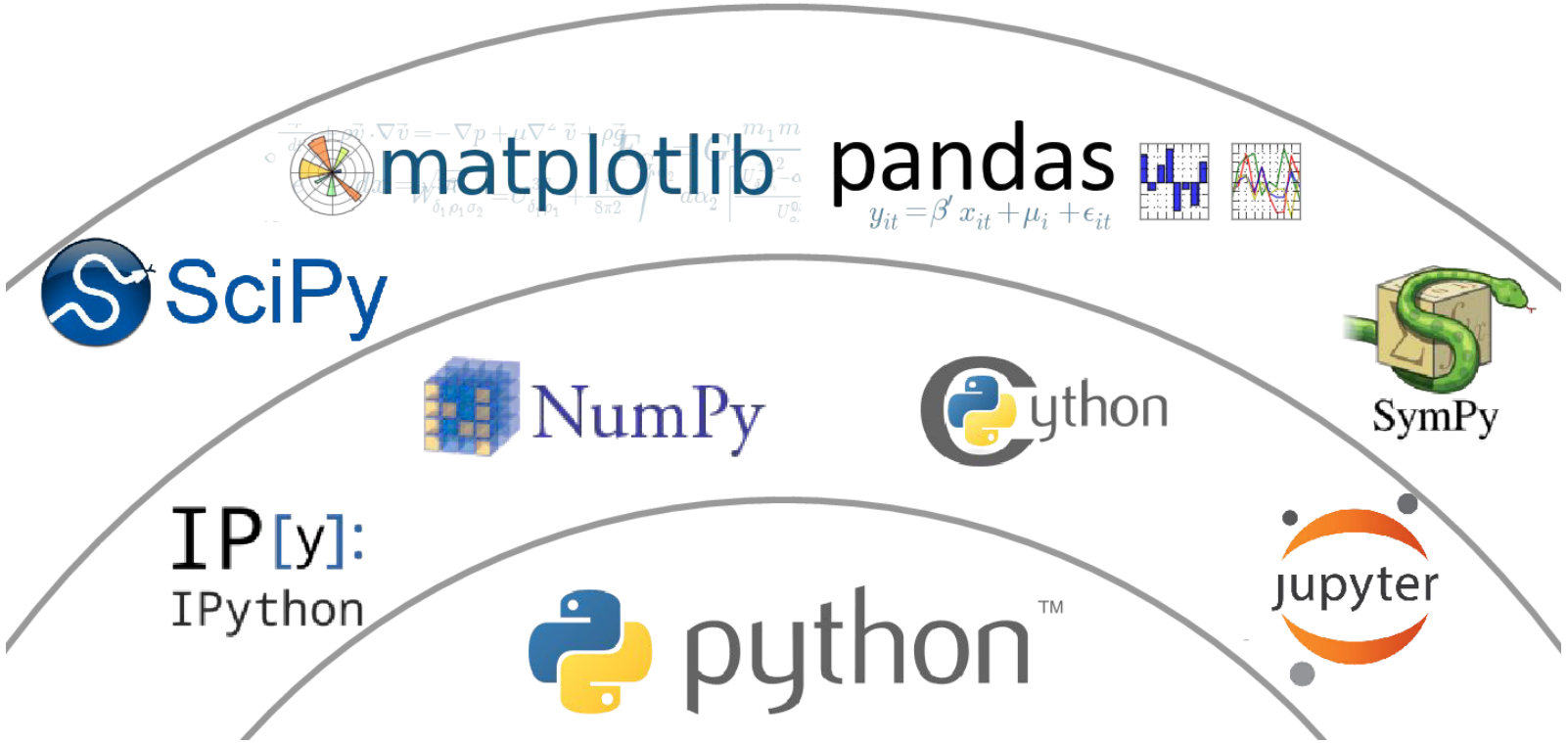
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

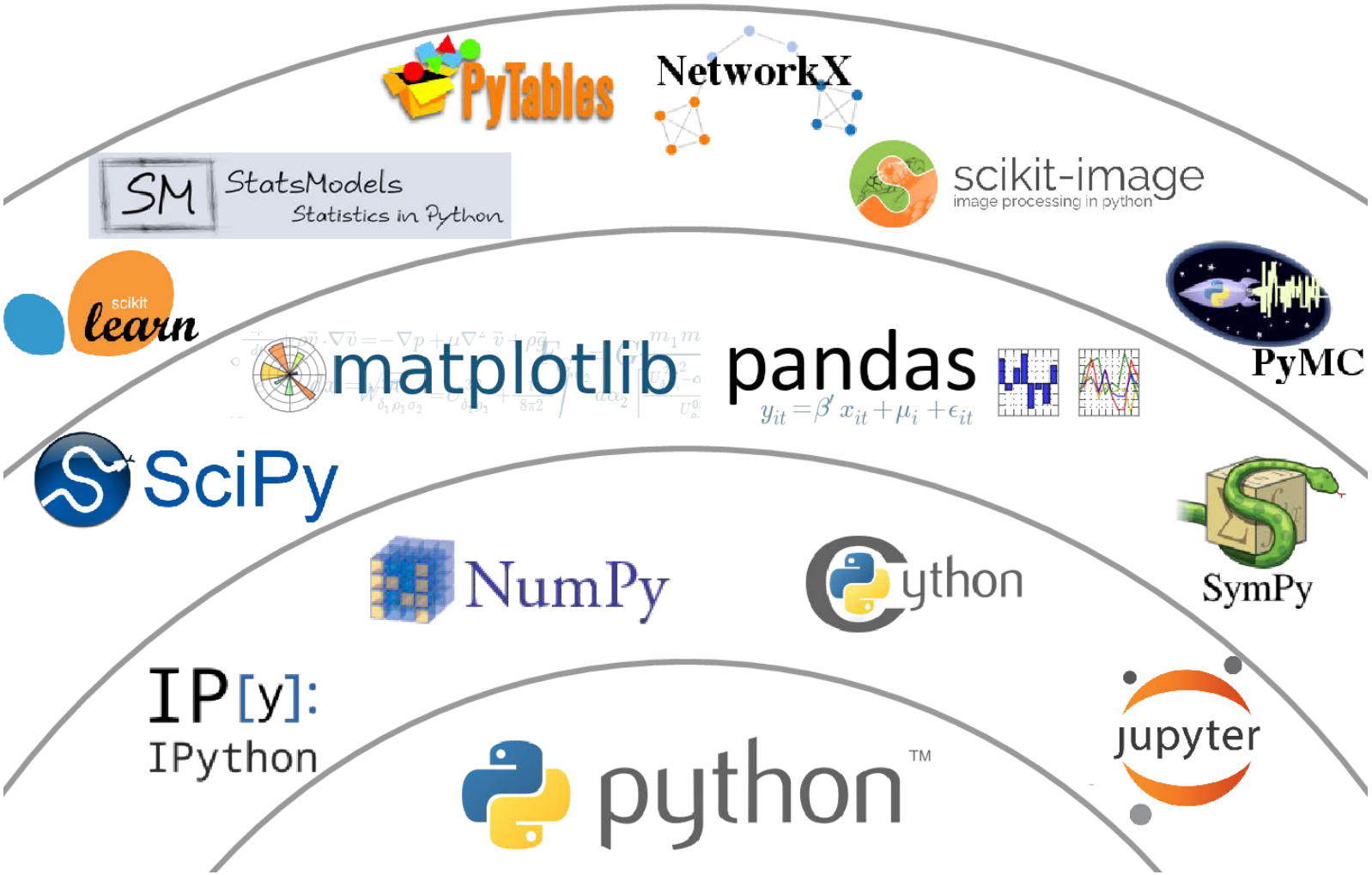
```
In [2]: data.groupby('Species').mean()
```

Out[2]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Species				
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

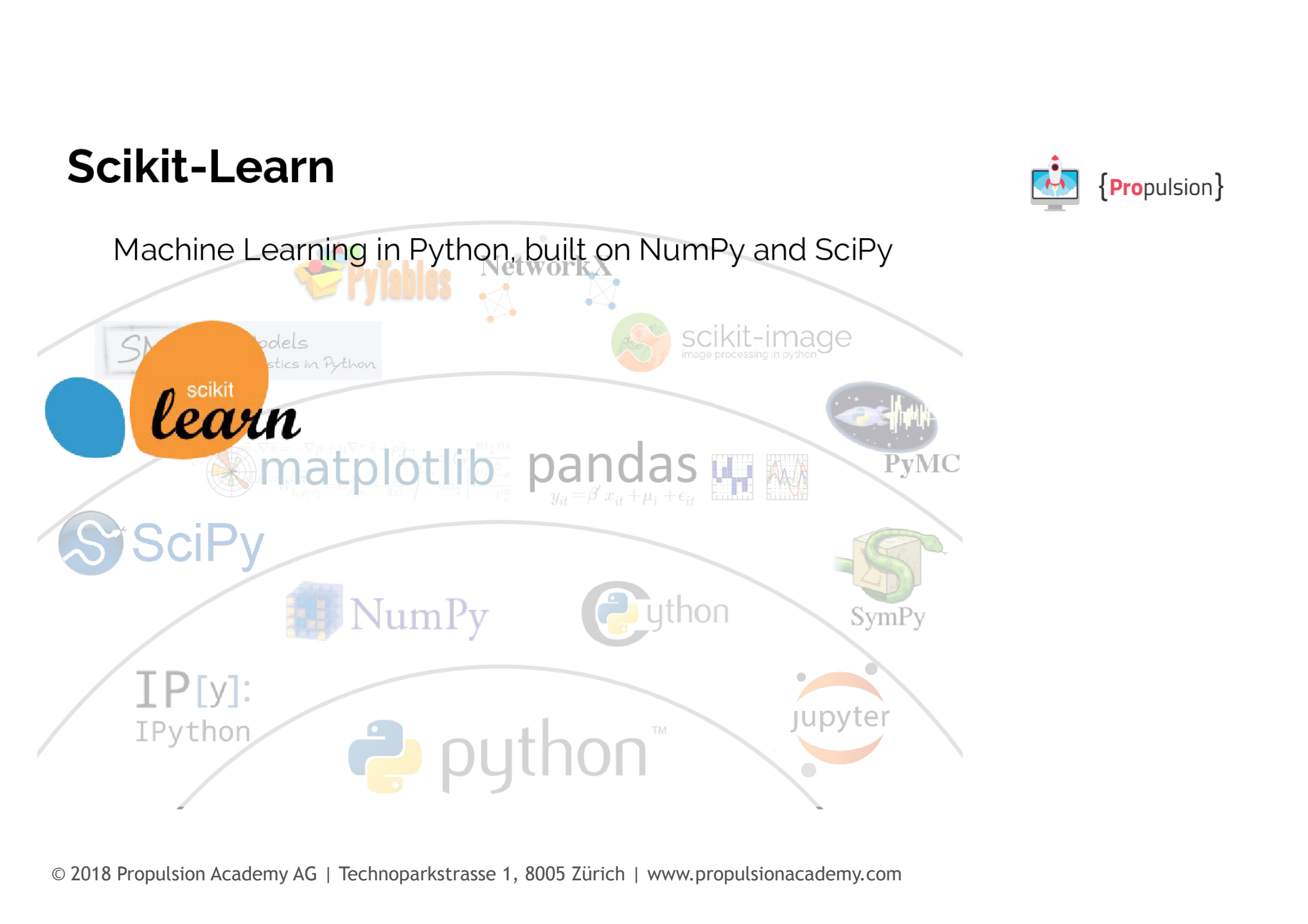
{Propulsion}





{Propulsion}

Scikit-Learn



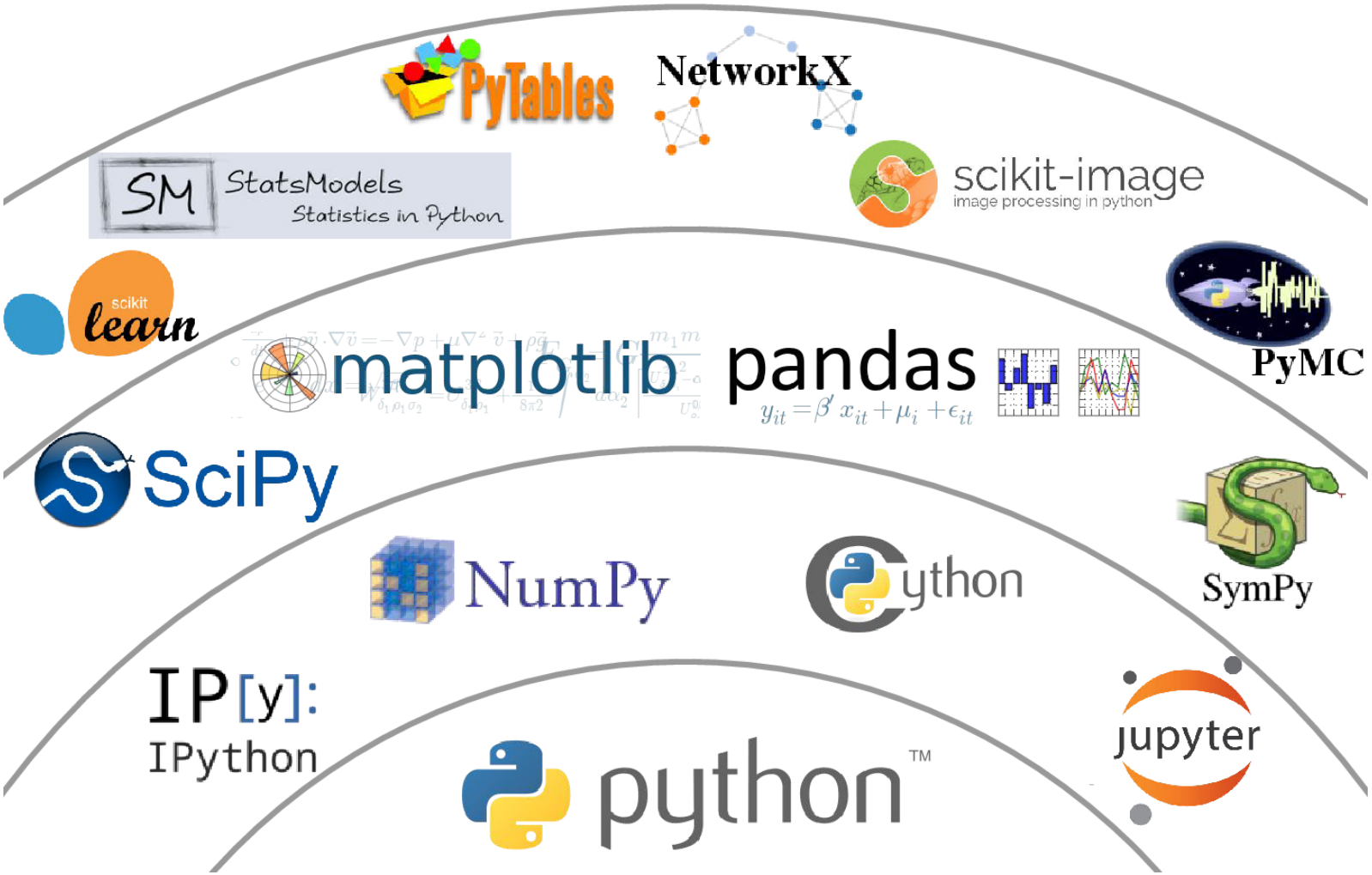
Machine Learning in Python, built on NumPy and SciPy

Scikit-Learn

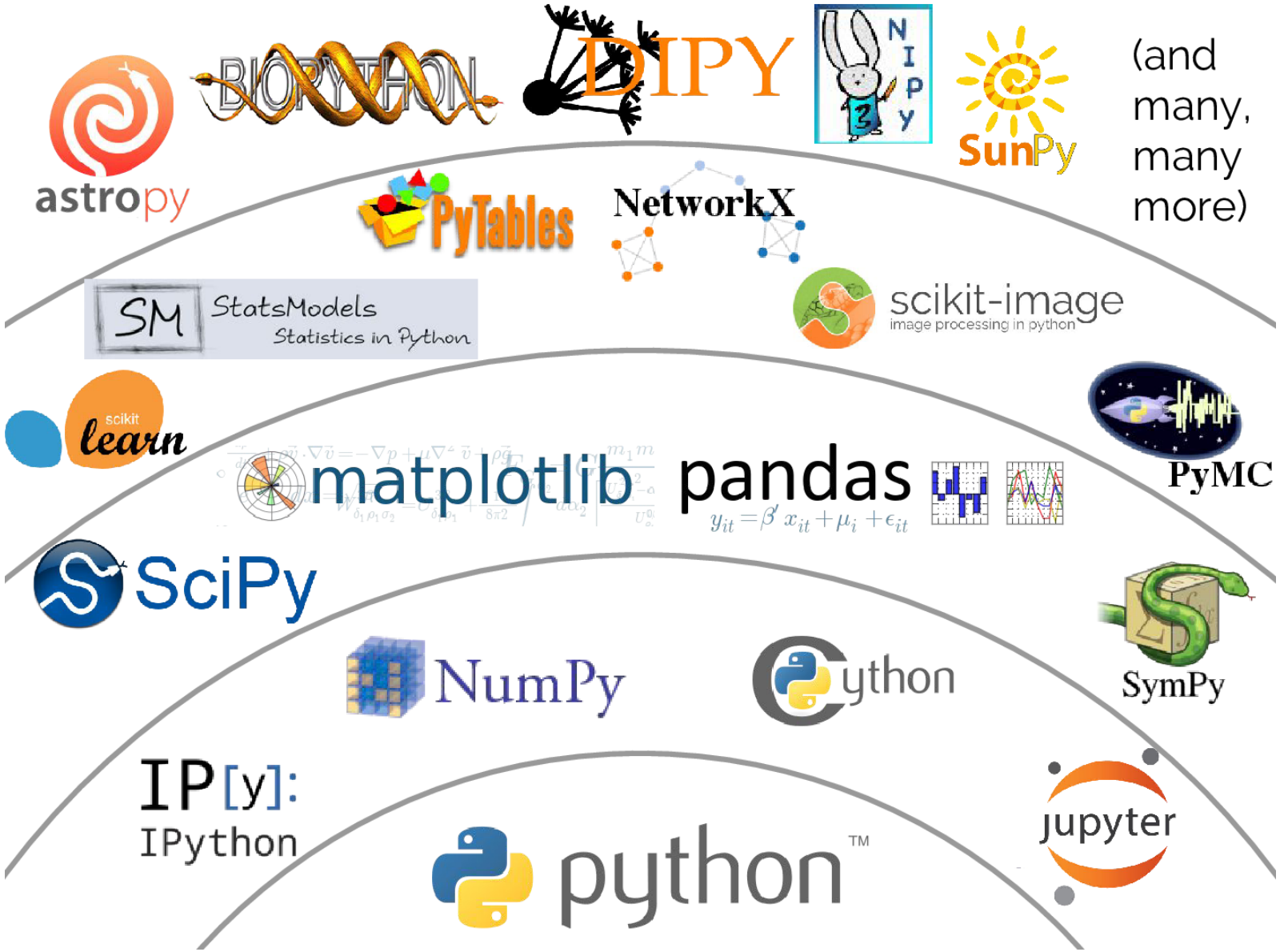
{Propulsion}

Machine Learning in Python, built on NumPy and SciPy

```
In [3]: from sklearn.ensemble import RandomForestClassifier

        features = data.drop('Species', axis=1)
        labels = data['Species']

        model = RandomForestClassifier()
        model.fit(features, labels)

        predicted = model.predict(features.iloc[:5])
        print(predicted)

        ['setosa' 'setosa' 'setosa' 'setosa' 'setosa']
```

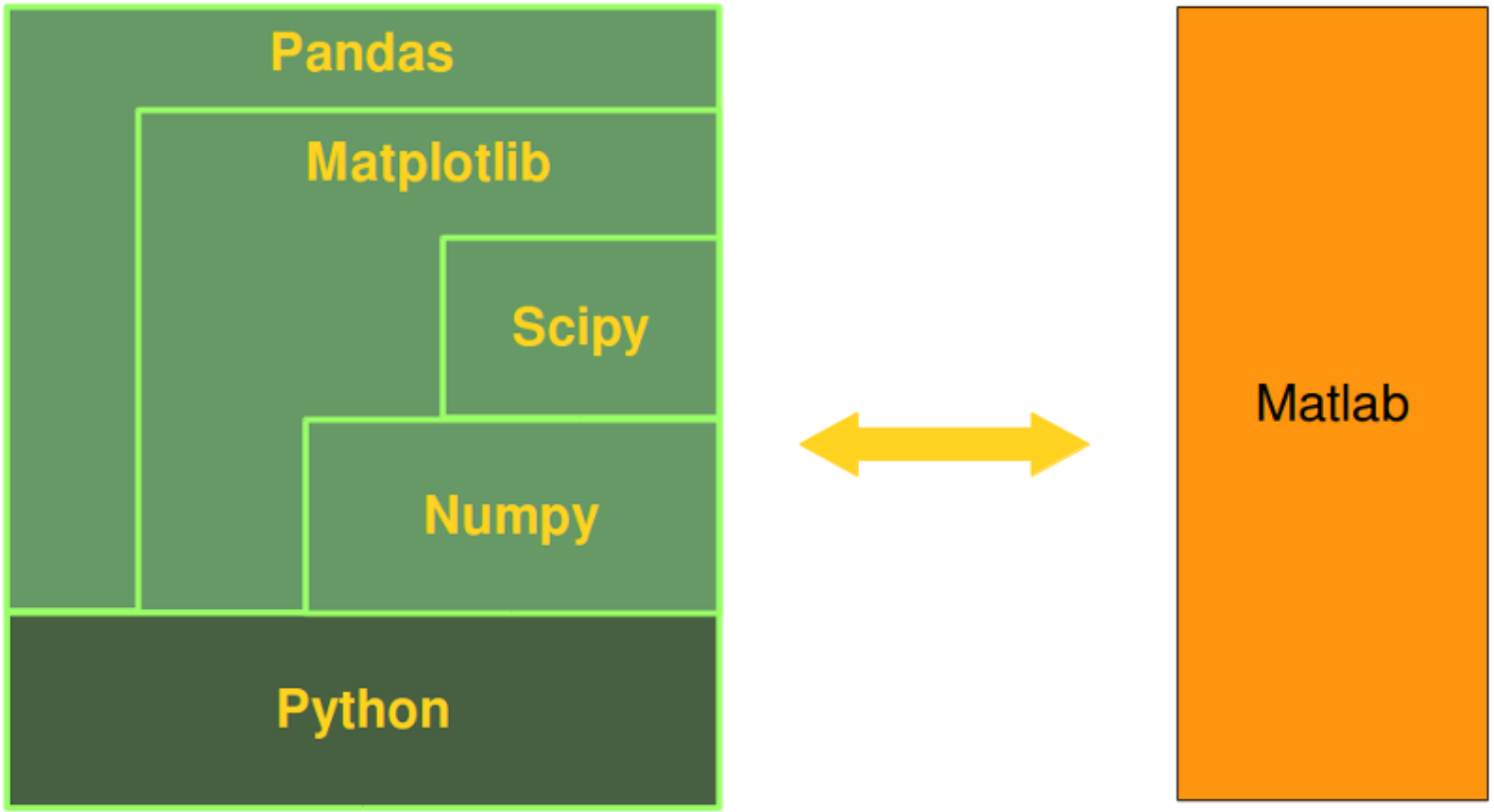


{Propulsion}



{Propulsion}

The Python alternative to MATLAB



{Propulsion}

NumPy vs “Core Python”

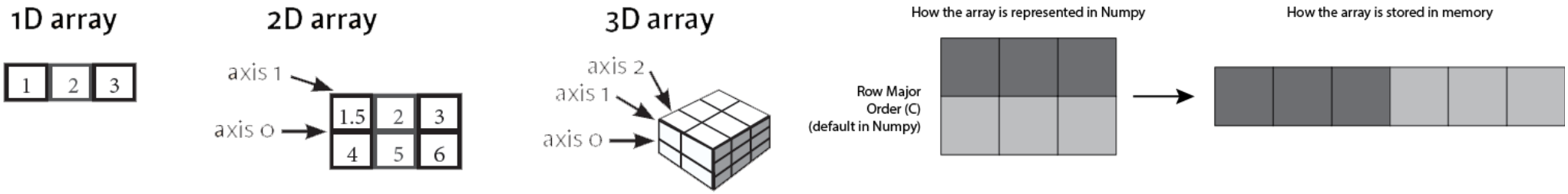
Advantages of Core Python

- High-level data structures: lists with cheap insertion/ appends, dictionaries with fast lookup

Advantages of NumPy

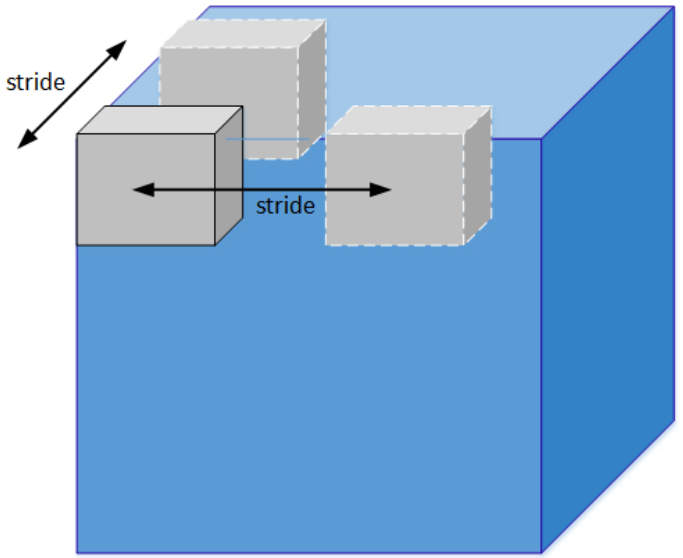
- Array-oriented computing
- Efficiently implemented multi-dimensional arrays
- Designed for scientific computation
- Optimisation during interpretation obtained through **vectorisation**

NumPy arrays



What is a NumPy array?

- Contiguous data structure held in memory
- *data* pointer indicates the memory address of the first byte in the array
- *dtype* (data type) pointer describes the kind of elements that are contained within the array
- *shape* indicates the shape of the array
- *strides* are the number of bytes that should be skipped in memory to go to the next element. If your strides are (10,1), you need to proceed one byte to get to the next column and 10 bytes to locate the next row.



Basic array initialisation and properties

{Propulsion}

```
import numpy as np

a = np.array([1, 2, 3])  # Create a rank 1 array
print type(a)            # Prints "<type 'numpy.ndarray'>"
print a.shape            # Prints "(3,)"
print a[0], a[1], a[2]   # Prints "1 2 3"
a[0] = 5                 # Change an element of the array
print a                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array
print b.shape                     # Prints "(2, 3)"
print b[0, 0], b[0, 1], b[1, 0]   # Prints "1 2 4"
```

{Propulsion}

Creating NumPy arrays

```
import numpy as np

a = np.zeros((2,2))  # Create an array of all zeros
print a              # Prints "[[ 0.  0.]
                     #          [ 0.  0.]]"

b = np.ones((1,2))   # Create an array of all ones
print b              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7) # Create a constant array
print c               # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"

d = np.eye(2)        # Create a 2x2 identity matrix
print d              # Prints "[[ 1.  0.]
                     #          [ 0.  1.]]"

e = np.random.random((2,2)) # Create an array filled with random values
print e                     # Might print "[[ 0.91940167  0.08143941]
                            #              [ 0.68744134  0.87236687]]"
```

{Propulsion}

Array indexing: slicing

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print a[0, 1]   # Prints "2"
b[0, 0] = 77    # b[0, 0] is the same piece of data as a[0, 1]
print a[0, 1]   # Prints "77"
```

{Propulsion}

Integer array indexing

```
import numpy as np

# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print a  # prints "array([[ 1,  2,  3],
         #                [ 4,  5,  6],
         #                [ 7,  8,  9],
         #                [10, 11, 12]])"

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print a[np.arange(4), b]  # Prints "[ 1  6  7 11]"

# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10

print a  # prints "array([[11,  2,  3],
         #                [ 4,  5, 16],
         #                [17,  8,  9],
         #                [10, 21, 12]])
```

{Propulsion}

Boolean array indexing

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;
                    # this returns a numpy array of Booleans of the same
                    # shape as a, where each slot of bool_idx tells
                    # whether that element of a is > 2.

print bool_idx      # Prints "[[False False]
                    #          [ True  True]
                    #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print a[bool_idx]  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print a[a > 2]     # Prints "[3 4 5 6]"
```

Array data types

{Propulsion}

```
import numpy as np

x = np.array([1, 2])  # Let numpy choose the datatype
print x.dtype         # Prints "int64"

x = np.array([1.0, 2.0])  # Let numpy choose the datatype
print x.dtype             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)  # Force a particular datatype
print x.dtype                         # Prints "int64"
```

Array math

{Propulsion}

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print x + y
print np.add(x, y)

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print x - y
print np.subtract(x, y)
```

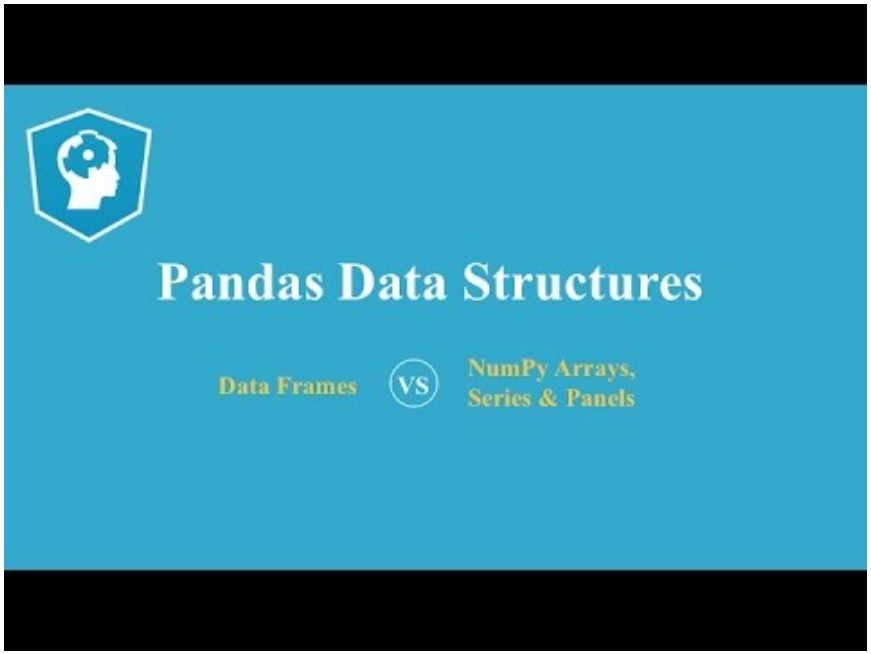
```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print x * y
print np.multiply(x, y)

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print x / y
print np.divide(x, y)

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print np.sqrt(x)
```

- Note: matrix/vector multiplication is accessed with np.dot

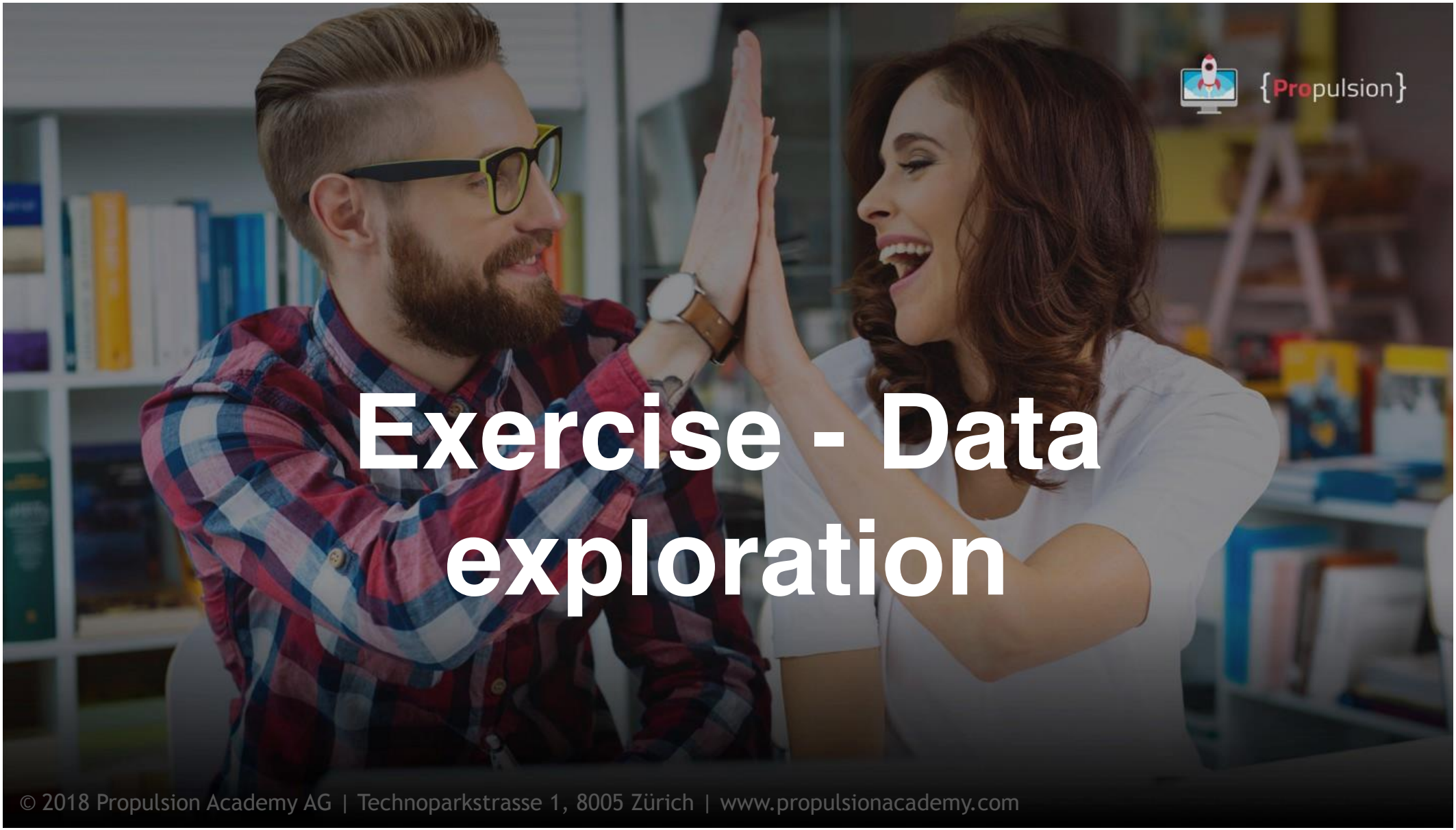
```
import numpy as np

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

print np.sum(x)  # Compute sum of all elements; prints "10"
print np.sum(x, axis=0)  # Compute sum of each column; prints "[4 6]"
print np.sum(x, axis=1)  # Compute sum of each row; prints "[3 7]"
```

```
x = np.array([[1,2], [3,4]])
print x    # Prints "[[1 2]
           #          [3 4]]"
print x.T  # Prints "[[1 3]
           #          [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print v    # Prints "[1 2 3]"
print v.T  # Prints "[1 2 3]"
```

Pandas: NumPy on steroids



- Pandas implements fast and flexible data structures on top of NumPy to work with labelled data
- Usually, rows are observations, and columns are variables
- Comprehensive tutorial: tiny.cc/propacad-ds-pandas-tut
- Reference guide: tiny.cc/propacad-ds-pandas-guide


{Propulsion}
Exercise - Data exploration
© 2018 Propulsion Academy AG | Technoparkstrasse 1, 8005 Zürich | www.propulsionacademy.com

Pandas & CSV: winning combo

(aka: finally some data! Don't forget Jupyter notebook)

- IGN game review data set: tiny.cc/propacad-ds-ign

Questions

- What's the first game ever released?
- Do game scores vary from year to year?
- What's the platform with the highest average reviews?
- Are blockbuster games released at a specific time of the year?

- And what if… We wanted to analyse sales too? tiny.cc/propacad-ds-gamesales