Data Science & ML Course

Lesson #4 [Part #1] Introduction to Numpy

Ivanovitch Silva

October, 2018

Agenda

- Numpy - Overview
- Understanding NumPy ndarrays
- Selecting and slicing
- Arithmetic operations
- Adding rows and columns
- Sorting
- Reading CSV files with NumPy
- Boolean Arrays
- Boolean Index
- Assigning values
- Challenge

nyc_taxis.csv

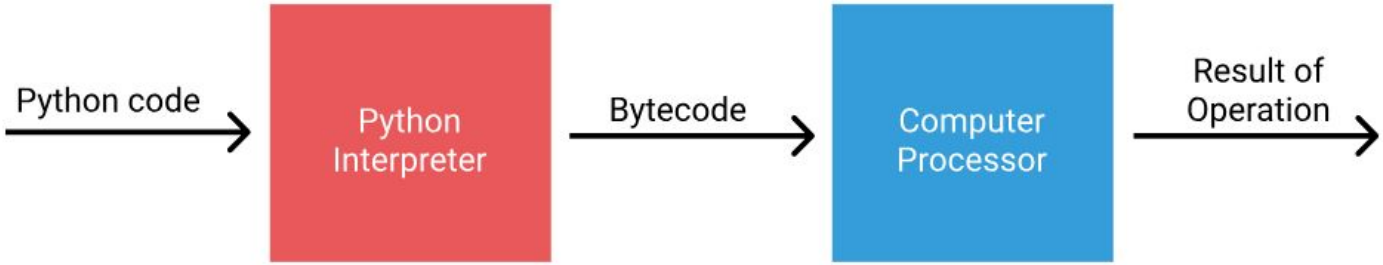
Update from repository

```
git clone https://github.com/ivanovitchm/datascience2machinelearning.git
```

Or ....

```
git pull
```

Understanding Vectorization



Language Type	Example	Time taken to write program	Control over program performance
High-Level	Python	Low	Low
Low-Level	C	High	High

How can Python be so efficient?

Unvectorized code using list of lists

6	5
1	3
5	6
1	4
3	7
5	8
3	5
8	4

Two columns of numbers

```
my_numbers = [
                [6, 5],
                [1, 3],
                [5, 6],
                [1, 4],
                [3, 7],
                [5, 8],
                [3, 5],
                [8, 4]
              ]
```

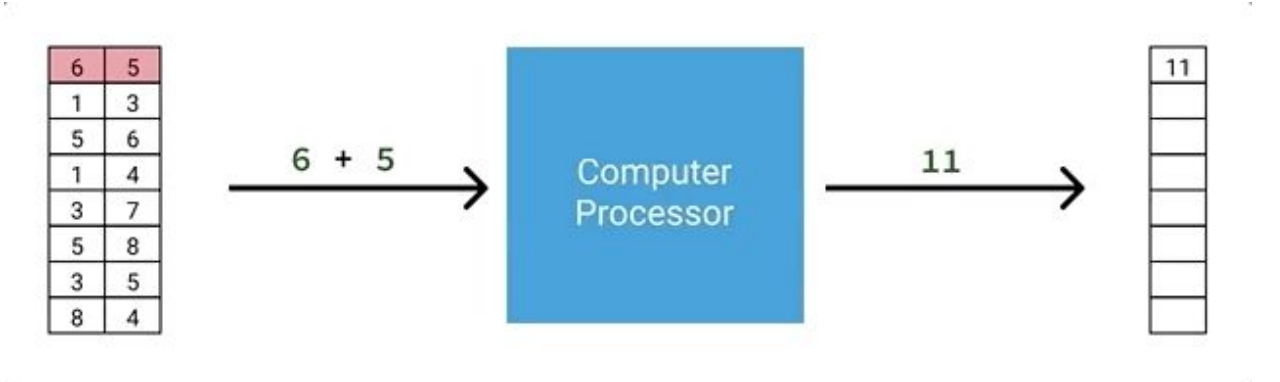
List of lists representation

```
sums = []

for row in my_numbers:
    row_sum = row[0] + row[1]
    sums.append(row_sum)
```

Python code to sum each row

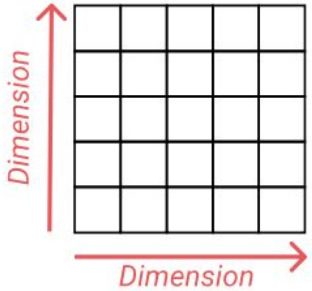
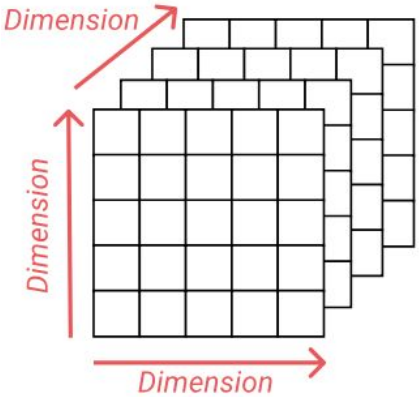
How Vectorization Makes Code Faster

Single Instruction Multiple Data (SIMD)



NumPY

Understanding Numpy ndarray

	Number of Dimensions	Known As
	One	One-dimensional array, array, list, vector, sequence
	Two	Two-dimensional array, matrix, table, list of lists, spreadsheet
	Three	Three-dimensional array, multi-dimensional array, panel

NYC Taxi-Airport Data

There is data on over 1.3 trillion individual trips, reaching back as far as 2009 and is regularly updated

nyc_taxis.csv

NYC Taxi-Airport Data

pickup_year	pickup_month	pickup_day	pickup_dayofweek	pickup_time	pickup_location_code	dropoff_location_code	trip_distance	trip_length	fare_amount	total_amount
2016	1	1	5	0	2	4	21.00	2037	52.0	69.99
2016	1	1	5	0	2	1	16.29	1520	45.0	54.30
2016	1	1	5	0	2	6	12.70	1462	36.5	37.80
2016	1	1	5	0	2	6	8.70	1210	26.0	32.76
2016	1	1	5	0	2	6	5.56	759	17.5	18.80
2016	1	1	5	0	4	2	21.45	2004	52.0	105.60
2016	1	1	5	0	2	6	8.45	927	24.5	32.25
2016	1	1	5	0	2	6	7.30	731	21.5	22.80
2016	1	1	5	0	2	5	36.30	2562	109.5	131.38
2016	1	1	5	0	6	2	12.46	1351	36.0	37.30

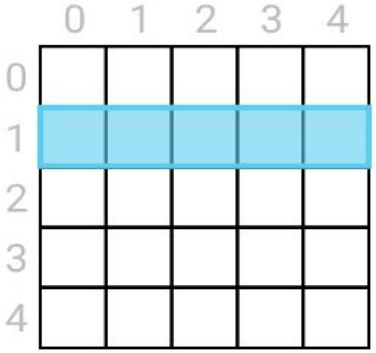
```
1  import csv
2  import numpy as np
3
4  # import nyc_taxi.csv as a list of lists
5  # remove the header row
6  # convert each element to float
7
8  taxi = np.array(
9        [[float(item) for item in row]
10         for row in list(csv.reader(open("nyc_taxis.csv", "r")))[1:]]
11      )
12 print(type(taxi))
13 taxi[:2]
14
15
```

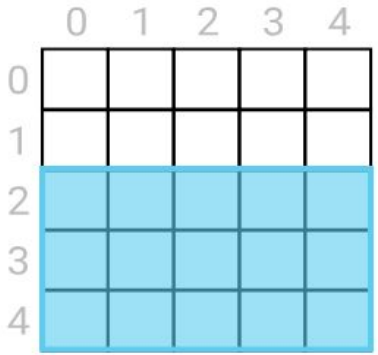
```
<class 'numpy.ndarray'>
array([[2.016e+03, 1.000e+00, 1.000e+00, 5.000e+00, 0.000e+00, 2.000e+00,
        4.000e+00, 2.100e+01, 2.037e+03, 5.200e+01, 8.000e-01, 5.540e+00,
        1.165e+01, 6.999e+01, 1.000e+00],
       [2.016e+03, 1.000e+00, 1.000e+00, 5.000e+00, 0.000e+00, 2.000e+00,
        1.000e+00, 1.629e+01, 1.520e+03, 4.500e+01, 1.300e+00, 0.000e+00,
        8.000e+00, 5.430e+01, 1.000e+00]])
```

Introduction to Numpy

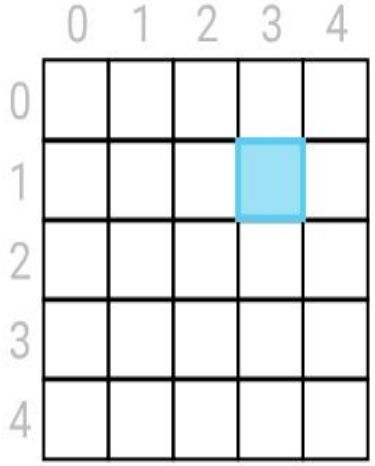
```
>>> print(taxi)

    [[ 2016.  1.   1.  ..., 11.65  69.99   1. ]
     [ 2016.  1.   1.  ...,  8.    54.3    1. ]
     [ 2016.  1.   1.  ...,  0.    37.8    2. ]
     ...,
     [ 2016.  6.  30.  ...,  5.    63.34   1. ]
     [ 2016.  6.  30.  ...,  8.95  44.75   1. ]
     [ 2016.  6.  30.  ...,  0.    54.84   2. ]]

 >>> taxi.shape
     (89560, 15)
```

	List of lists method	*NumPy method*
Selecting a single row	`sel_lol = data_lol[1]`	`sel_np = data_np[1]`
		Same syntax as list of lists. Produces a 1D ndarray.
Selecting multiple rows	`sel_lol = data_lol[2:]`	`sel_np = data_np[2:]`
		Same syntax as list of lists. Produces a 2D ndarray.





Selecting a single item



List of lists method

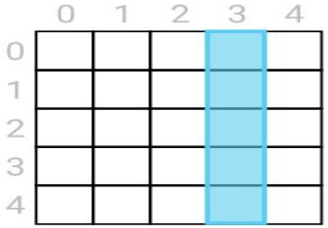
```
sel_lol = data_lol[1][3]
```

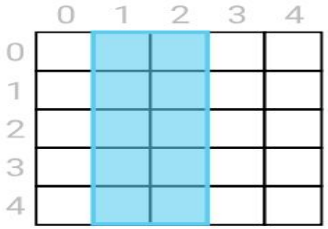
NumPy method

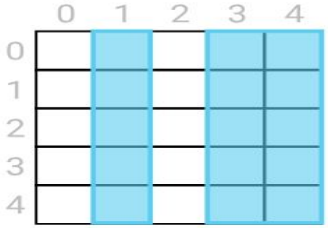
```
sel_np = data_np[1,3]
```

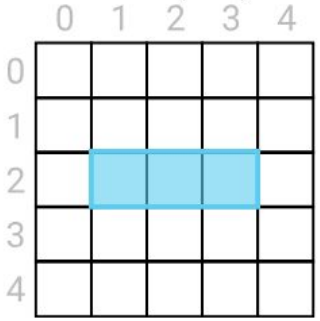
Comma separated row/column locations. Produces a single Python object.

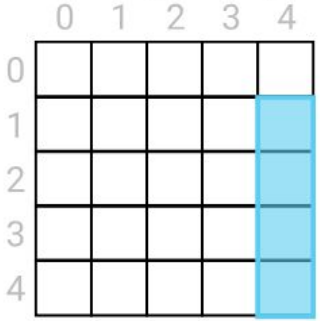
	List of lists method	*NumPy method*

Selecting a single column 	`sel_lol = []`
	
	`for row in data_lol:`
	`  col4 = row[3]`
	`  sel_lol.append(col4)`
		`sel_np = data_np[:,3]`
		
		Comma separated row wildcard and column location. Produces a 1D ndarray

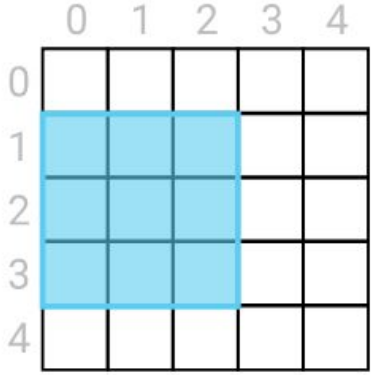
Selecting multiple columns 	`sel_lol = []`
	
	`for row in data_lol:`
	`  col23 = row[1:3]`
	`  sel_lol.append(col23)`
		`sel_np = data_np[:,1:3]`
		
		Comma separated row wildcard and column slice location. Produces a 2D ndarray

Selecting multiple, specific columns 	`sel_lol = []`
	
	`for row in data_lol:`
	`  cols = [row[1],`
	`  row[3],row[4]]`
	`  sel_lol.append(cols)`
		`cols = [1,3,4]`
		`sel_np = data_np[:,cols]`
		
		Comma separated row wildcard and list of column locations. Produces a 2D ndarray

	List of lists method	*NumPy method*
Selecting a 1D slice (row) 	`sel_lol = data_lol[2][1:4]`	`sel_np = data_np[2,1:4]` *Comma separated row location and column slice. Produces a 1D ndarray*

Selecting a 1D slice (column)
 
	`sel_lol = []`
	
	`rows = data_lol[1:]`
	`for r in rows:`
	`  col5 = r[4]`
	`  sel_lol.append(col5)`
		`sel_np = data_np[1:,4]`
		Comma separated row slice and column location. Produces a 1D ndarray

Selecting a 2D slice



List of lists method

```
sel_lol = []

rows = data_lol[1:4]
for r in rows:
    new_row = r[:3]
    sel_lol.append(new_row)
```

NumPy method

```
sel_np = data_np[1:4,:3]
```

Comma separated row/column slice locations. Returns a 2D ndarray

Vectorized Operations (list of lists vs numpy)

```
import numpy as np

# create random (5000000,5) numpy arrays and
# list of lists
np_array = np.random.rand(5000000,5)
list_array = np_array.tolist()

def python_subset():
    filtered_cols = []
    for row in list_array:
        filtered_cols.append([row[1],row[2]])
    return filtered_cols

def numpy_subset():
    return np_array[:,1:3]
```

Vectorized Operations (list of lists vs numpy)

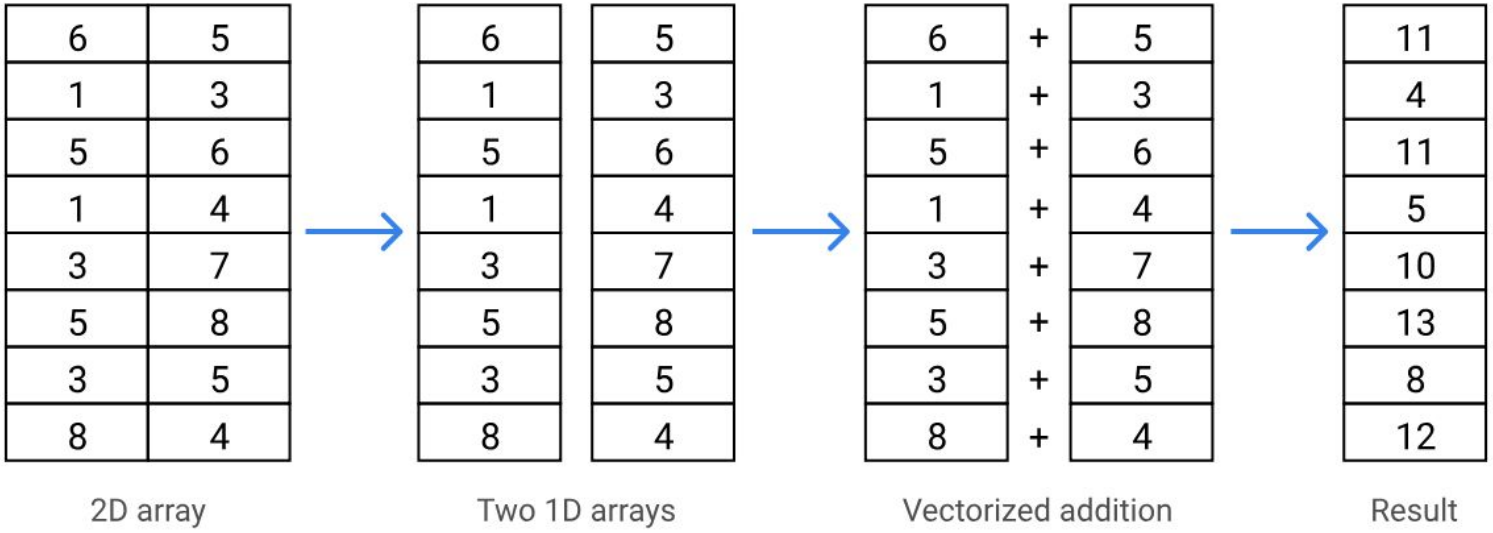
```
%%timeit -r 2 -n 10
# the number of executions will be n * r

list_of_list = python_subset()
```

```
1.32 s ± 22.2 ms per loop (mean ± std. dev. of 2 runs, 10 loops each)
```

```
%%timeit -r 2 -n 10
# the number of executions will be n * r

numpy_array = numpy_subset()
```

```
724 ns ± 294 ns per loop (mean ± std. dev. of 2 runs, 10 loops each)
```

Mathematical Operations (numpy)



Calculating Statistics for 1D ndarrays

Calculation	Function Representation	Method Representation
Calculate the minimum value of **trip_mph**	np.min(trip_mph)	trip_mph.min()
Calculate the maximum value of **trip_mph**	np.max(trip_mph)	trip_mph.max()
Calculate the mean average value of **trip_mph**	np.mean(trip_mph)	trip_mph.mean()
Calculate the median average value of **trip_mph**	np.median(trip_mph)	There is no ndarray median method

Calculating Statistics for 2D ndarrays

2D ndarray

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

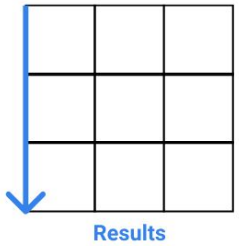
`ndarray.max(axis=0)`

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

3	1	4	3

Result

`ndarray.method(axis=0)`
Calculates along the **row** axis



Calculates result for each **column**.

2D ndarray

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

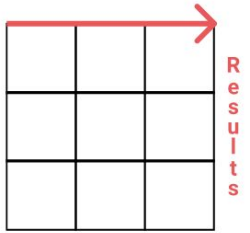
`ndarray.max(axis=1)`

1	0	1	1
0	1	4	3
0	1	0	2
3	0	1	3

Result

1
4
2
3

`ndarray.method(axis=1)`
Calculates along the **column** axis



Calculates result for each **row**.

Adding Rows and Columns to ndarrays (concatenate)

```
>>> print(ones)

    [[ 1  1  1]
     [ 1  1  1]]

>>> print(zeros)

    [ 0  0  0]

>>> print(ones.shape)

    (2, 3)

>>> print(zeros.shape)

    (3,)
```

```
>>> combined = np.concatenate([ones,zeros],axis=0)

    Traceback (most recent call last):
      File "stdin", line 1, in module
    ValueError: all the input arrays must have same number of dimensions
```

Object	Current shape	Desired Shape
ones	(2, 3)	(2, 3)
zeros	(3,)	(1, 3)

Adding Rows and Columns to ndarrays (concatenate)

```
>>> zeros_2d = np.expand_dims(zeros,axis=0)

>>> print(zeros_2d)

    [[ 0  0  0]]

>>> print(zeros_2d.shape)

    (1, 3)
```

Adding Rows and Columns to ndarrays (concatenate)

```
>>> combined = np.concatenate([ones,zeros_2d],axis=0)

>>> print(combined)

    [[ 1  1  1]
     [ 1  1  1]
     [ 0  0  0]]
```

Sorting ndarrays

```
fruit = np.array(['orange', 'banana',
                  'apple', 'grape',
                  'cherry'])
```

orange	0
banana	1
apple	2
grape	3
cherry	4

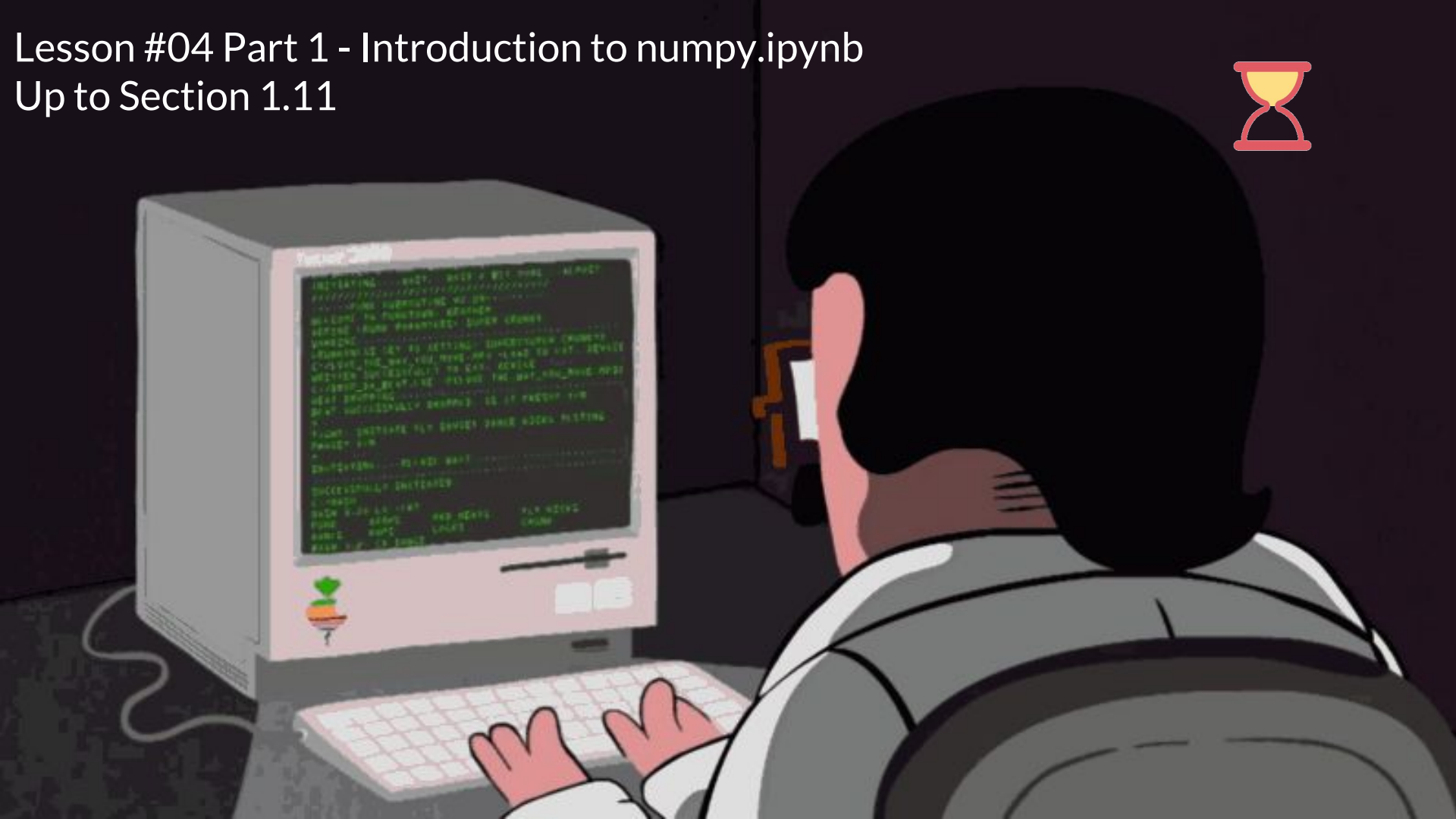
```
sorted_fruit = fruit[sorted_order]
```

apple
banana
cherry
grape
orange

```
sorted_order = np.argsort(fruit)
```

2	1	4	3	0

Lesson #04 Part 1 - Introduction to numpy.ipynb
Up to Section 1.11

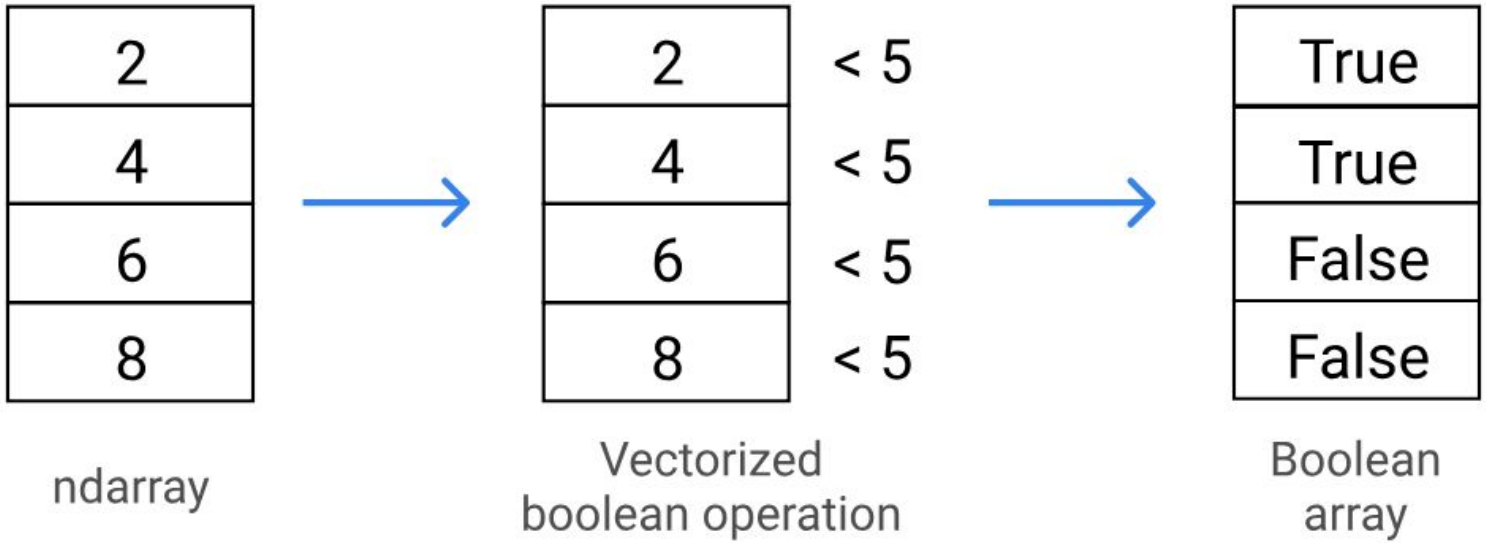


Reading CSV files from Numpy

```
taxi = np.genfromtxt('nyc_taxis.csv', delimiter=',')
print(taxi)
```

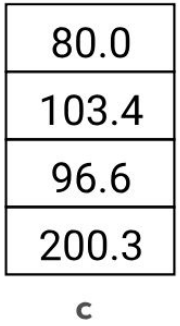
```
[[   nan    nan    nan ...,    nan    nan    nan]
 [  2016      1      1 ...,  11.65  69.99      1]
 [  2016      1      1 ...,      8   54.3      1]
 ...,
 [  2016      6     30 ...,      5  63.34      1]
 [  2016      6     30 ...,   8.95  44.75      1]
 [  2016      6     30 ...,      0  54.84      2]]
```

Slicing from boolean arrays



Boolean indexing with 1D ndarrays

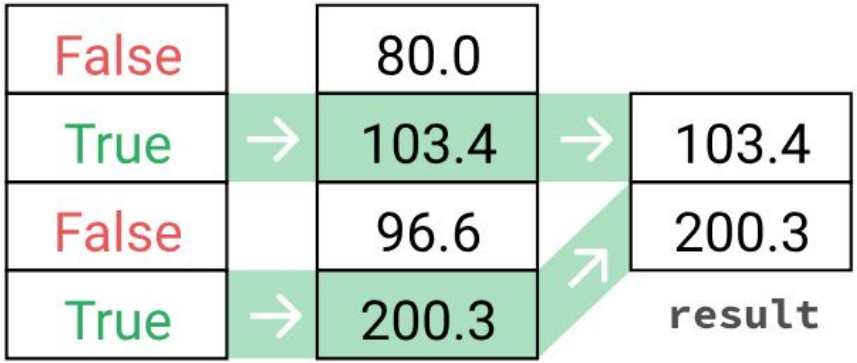
```
c = np.array([80.0, 103.4,
              96.9, 200.3])
```



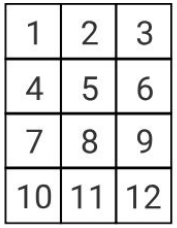
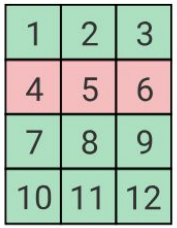
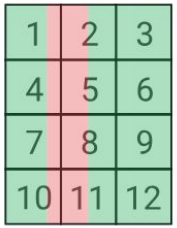
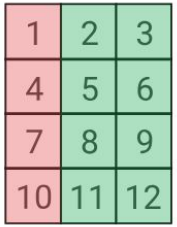

```
c_bool = c > 100
```

Boolean indexing with 1D ndarrays

```
result = c[c_bool]
```



Boolean Indexing with 2D ndarrays

Code	Visualization	Explanation
`arr = np.array([ [ 1, 2, 3], [ 4, 5, 6], [ 7, 8, 9], [10, 11, 12] ]) print(arr)`		The original array
`bool_1 = [True, False, True, True] print(arr[bool_1])`		`bool_1`'s shape (4) is the same as the shape of `arr`'s first axis (4), so this selects the 1st, 3rd, and 4th rows.
`print(arr[:,bool_1])`		`bool_1`'s shape (4) is not the same as the shape of `arr`'s second axis (3), so it can't be used to index and produces an `error`
`bool_2 = [False, True, True] print(arr[:,bool_2])`		`bool_2`'s shape (3) is the same as the shape of `arr`'s second axis (3), so this selects the 2nd and 3rd columns.

Assigning values in 1D ndarray

```
a = np.array(['red','blue','black','blue','purple'])
a[0] = 'orange'
print(a)

['orange', 'blue', 'black', 'blue', 'purple']
```

```
a[3:] = 'pink'
print(a)

['orange', 'blue', 'black', 'pink', 'pink']
```

Assigning values in 2D ndarray

```
ones = np.array([[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]])
ones[1,2] = 99
print(ones)
```

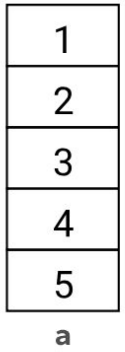
```
[[ 1,  1,  1,  1,  1],
 [ 1,  1, 99,  1,  1],
 [ 1,  1,  1,  1,  1]]
```

```
ones[0] = 42
print(ones)
```

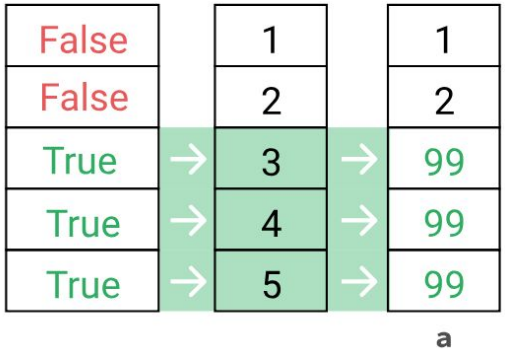
```
[[42, 42, 42, 42, 42],
 [ 1,  1, 99,  1,  1],
 [ 1,  1,  1,  1,  1]]
```

Assignment Using Boolean Arrays

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




```
a[a > 2] = 99
```



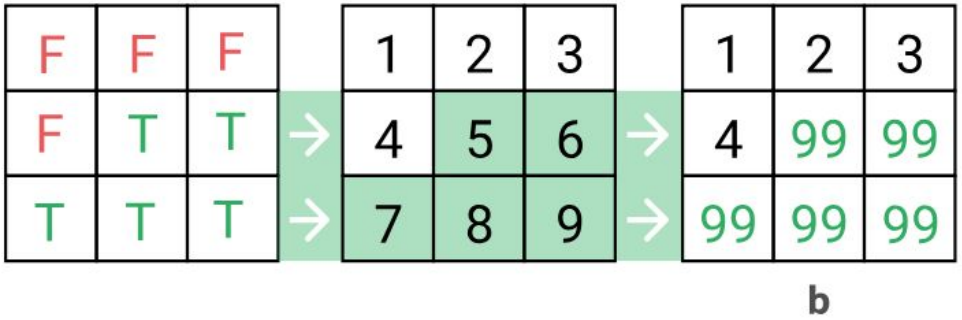
Assignment Using Boolean Arrays

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

1	2	3
4	5	6
7	8	9

b

```
b[b > 4] = 99
```



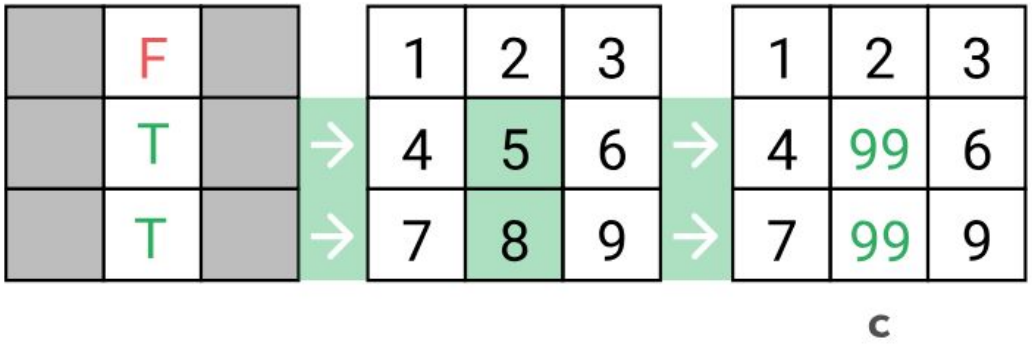
Assignment Using Boolean Arrays

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

1	2	3
4	5	6
7	8	9

c

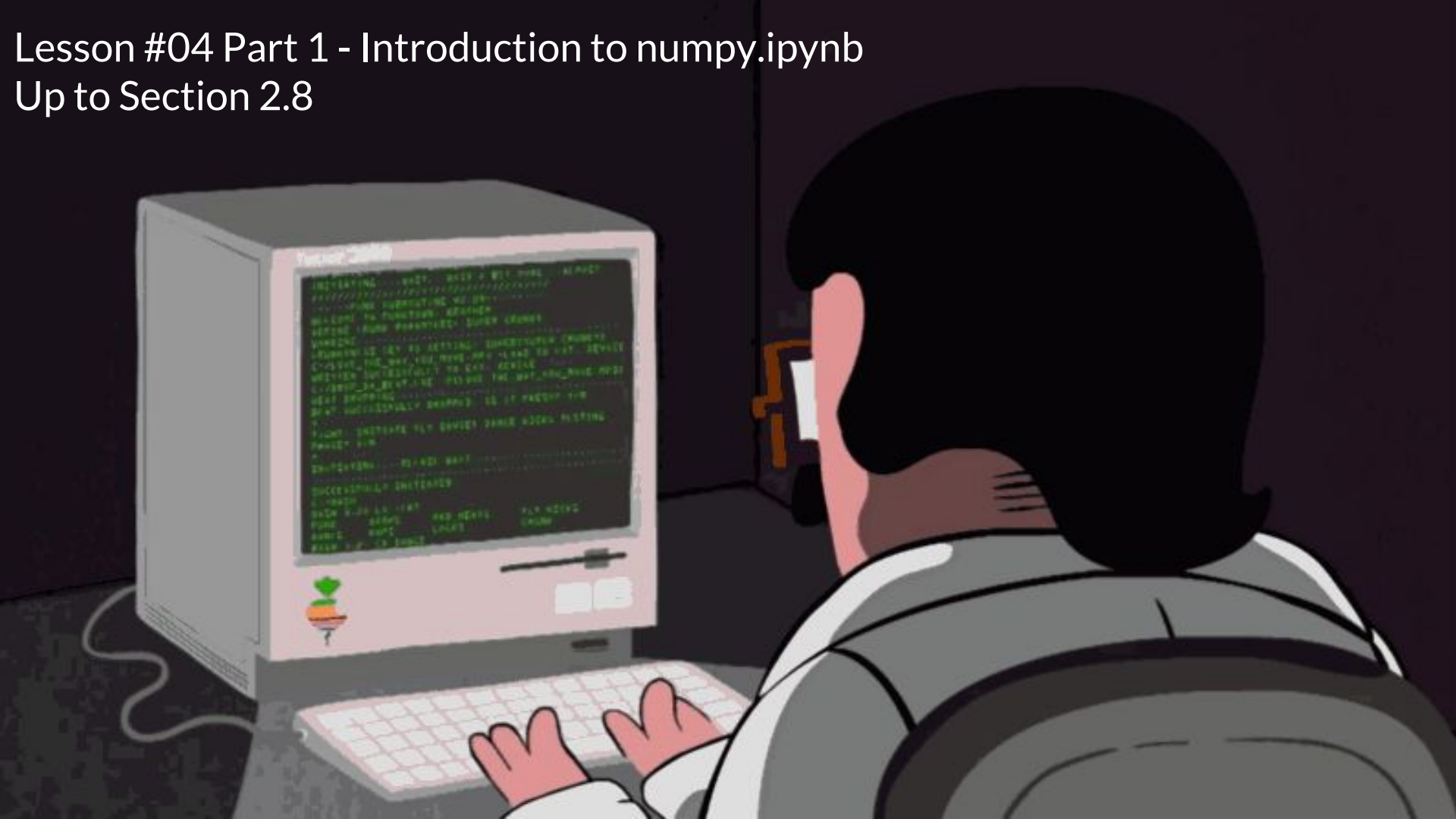
```
c[c[:, 1] > 2, 1] = 99
```



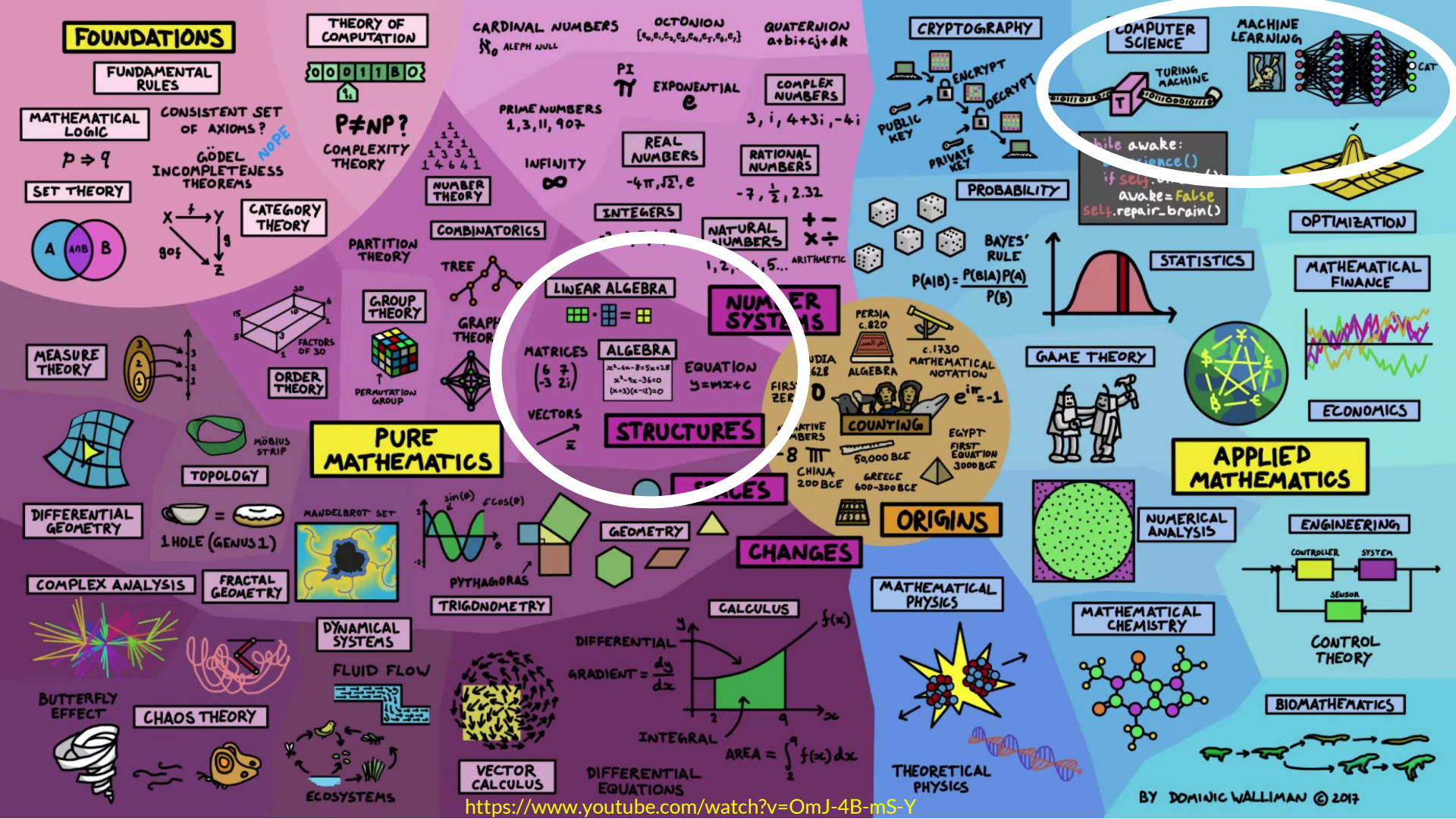
Challenges

Which is the most popular airport?
Calculating statistics for trip?

Lesson #04 Part 1 - Introduction to numpy.ipynb
Up to Section 2.8

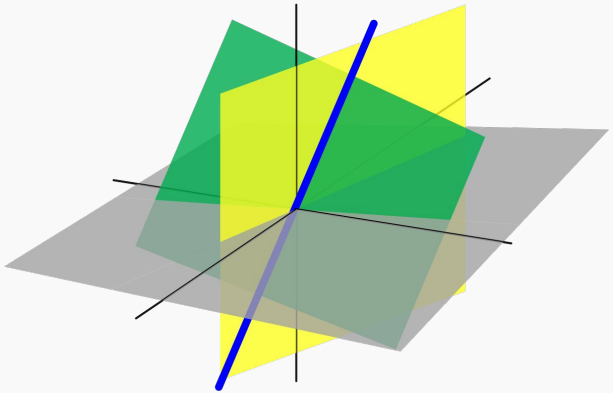


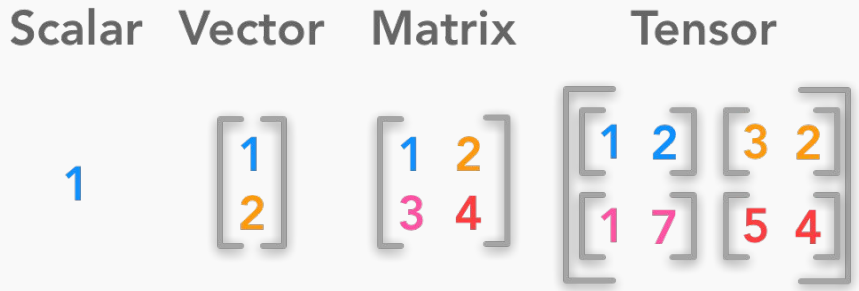
Basic of Algebra Linear for ML



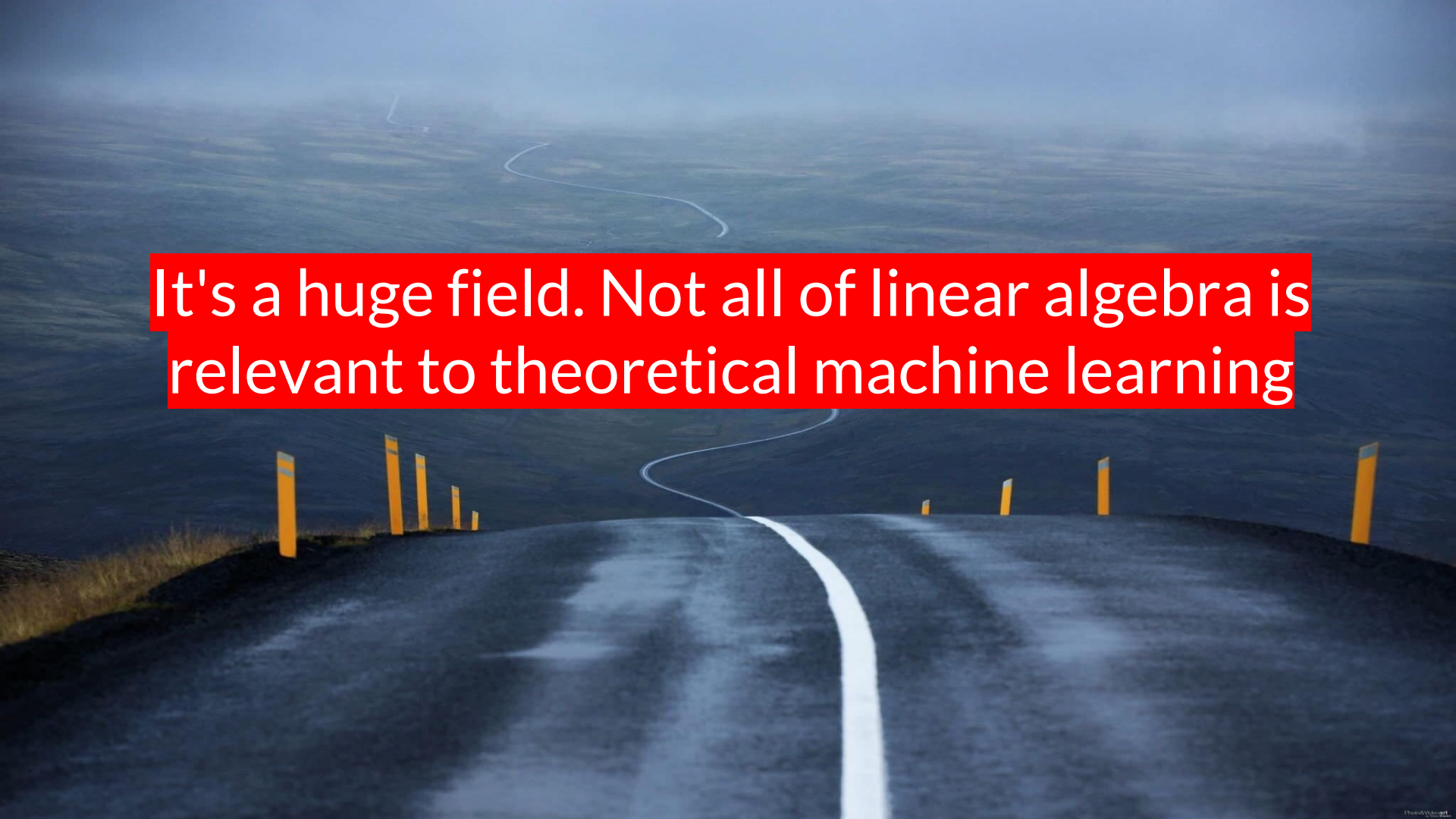
https://www.youtube.com/watch?v=OmJ-4B-mS-Y

Linear Algebra & Machine Learning





Linear algebra is a field of mathematics that could be called the mathematics of data.


It's a huge field. Not all of linear algebra is relevant to theoretical machine learning

Learn Linear Algebra Notation & Arithmetic

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

Gr Liv Area	SalePrice
2480	205000
1829	237000
2673	249000
1005	133500
1768	224900

Export to plot.ly »

$$hypothesis = \begin{bmatrix} 1 & 2480 \\ 1 & 1829 \\ 1 & 2679 \\ 1 & 1005 \\ 1 & 1768 \end{bmatrix} \times \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 2480\,\theta_1 + \theta_0 \\ 1829\,\theta_1 + \theta_0 \\ 2679\,\theta_1 + \theta_0 \\ 1005\,\theta_1 + \theta_0 \\ 1768\,\theta_1 + \theta_0 \end{bmatrix}$$

Examples of Linear Algebra in ML

1. Dataset and Data Files
2. Images and Photographs
3. One Hot Encoding
4. Linear Regression
5. Regularization
6. Principal Component Analysis
7. Singular-Value Decomposition
8. Latent Semantic Analysis
9. Recommender Systems
10. Deep Learning

Dataset and Data Files

Notation:

- m - number of training examples
- X's - input variable/features
- y's - output variable/ target variable

m = 1465

X	y
Lot Area	SalePrice
31770	215000
11622	105000
14267	172000
11160	244000
13830	189900

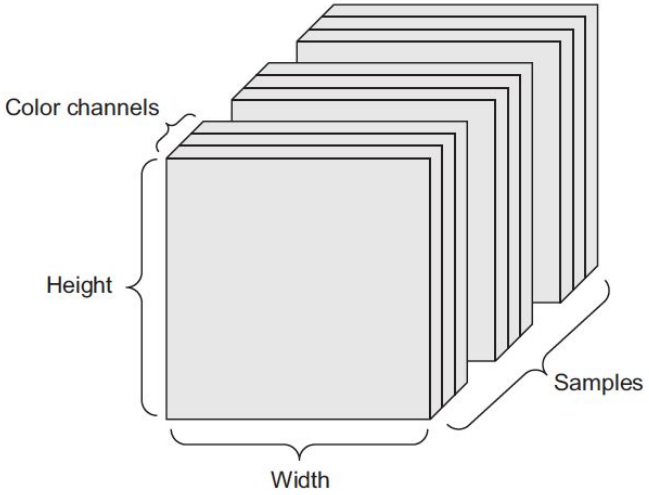
$X^{(1)} = 31770 \quad y^{(1)} = 215000$
$X^{(2)} = 11622 \quad y^{(2)} = 105000$
$X^{(3)} = 14267 \quad y^{(3)} = 172000$

$(X^{(i)}, y^{(i)})$ = i^{th} training example

Images and Photographs

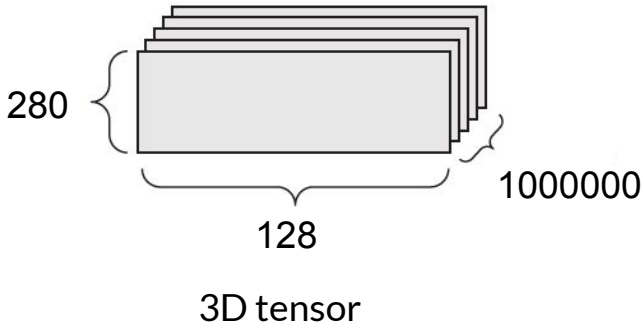
151	121	1	93	165	204	14	214	28	235
62	67	17	234	27	1	221	37	189	141
20	168	155	113	178	228	25	130	139	221
236	136	158	230	10	5	165	17	30	155
174	148	93	70	95	106	151	10	160	214
103	126	58	16	138	136	98	202	42	233
235	103	52	37	94	104	173	86	223	113
212	15	179	139	48	232	194	46	174	37
119	81	241	172	95	170	29	210	22	194
129	19	33	253	229	5	152	233	52	44
88	200	194	185	140	200	223	190	164	102
113	16	220	215	143	104	247	29	97	203
9	210	102	246	75	9	158	104	184	129
124	52	76	148	249	107	65	216	187	181
6	251	52	208	46	65	185	38	77	240
150	194	28	206	148	197	208	28	74	93
33	183	248	153	168	205	146	100	254	218
130	53	128	212	61	226	201	110	140	183
165	246	22	102	151	213	40	138	8	93
152	251	101	230	23	162	70	238	75	24
187	105	152	83	167	98	125	180	136	121
139	197	55	209	28	124	208	208	104	40
123	19	144	223	62	253	202	108	47	242
220	144	31	16	136	123	227	62	183	163

29	142	142	75	22	109	111	28	6	5
137	168	41	206	100	70	219	127	114	191
205	154	226	14	89	86	242	67	203	15
247	47	128	123	253	229	181	251	232	28
68	75	24	99	93	63	215	222	102	180
206	246	85	103	215	3	62	64	77	216
126	80	165	149	196	75	186	60	179	193
44	253	164	253	14	216	175	30	46	254
137	23	33	203	241	21	144	63	244	188
32	214	142	121	249	109	99	232	183	71
45	36	152	27	190	137	61	1	237	247
1	14	241	70	2	30	151	67	169	205
32	80	102	32	99	169	91	166	73	214
186	219	9	203	209	240	40	249	119	122
177	252	38	203	119	0	217	139	139	157
154	145	49	251	150	185	235	23	230	156
157	168	223	60	247	118	5	180	16	206
102	208	195	246	140	138	54	191	139	79
17	233	85	169	166	24	49	40	160	97
84	242	247	144	203	3	19	24	198	88
67	67	185	98	123	106	168	105	127	153
37	113	214	252	203	80	146	211	7	16
142	241	66	86	214	133	146	253	189	200
67	215	174	111	189	54	144	56	59	163



4D tensor

How to model a Twitter?

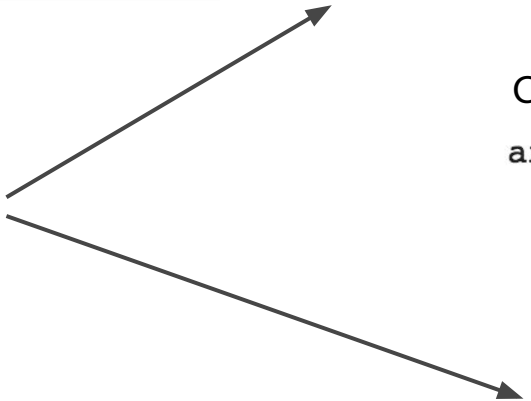


Suppose a dataset of 1 million tweets. Each tweet can be encoded as a 2D tensor of shape (280,128)

One hot encoding

```
train_X.ocean_proximity.head(10)
```

```
1380        NEAR BAY
12294         INLAND
7387        <1H OCEAN
14454     NEAR OCEAN
2927          INLAND
12462         INLAND
19813         INLAND
11229       <1H OCEAN
16696       <1H OCEAN
13564         INLAND
Name: ocean_proximity, dtype: object
```



maps each category to a different integer

```
array([0, 1, 2, 3, 1, 1, 1, 2, 2, 1])
```

Create a binary attribute per category

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

OneHotEnconder

Linear Regression

Lot Area	Overall Qual	Year Built	Yr Sold	SalePrice
31770	6	1960	2010	215000
11622	5	1961	2010	105000
14267	6	1958	2010	172000
11160	7	1968	2010	244000
13830	5	1997	2010	189900

$$X\theta = y$$
$$X^T X\theta = X^T y$$
$$\theta = (X^T X)^{-1} X^T y$$

$$\begin{bmatrix} 1 & 31770 & 6 & 1960 & 2010 \\ 1 & 11622 & 5 & 1961 & 2010 \\ 1 & 14267 & 6 & 1958 & 2010 \\ 1 & 11160 & 7 & 1968 & 2010 \\ 1 & 13830 & 5 & 1997 & 2010 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} \theta_0 + 31770\theta_1 + 6\theta_2 + 1960\theta_3 + 2010\theta_4 \\ \theta_0 + 11622\theta_1 + 5\theta_2 + 1961\theta_3 + 2010\theta_4 \\ \theta_0 + 14267\theta_1 + 6\theta_2 + 1958\theta_3 + 2010\theta_4 \\ \theta_0 + 11160\theta_1 + 7\theta_2 + 1968\theta_3 + 2010\theta_4 \\ \theta_0 + 13830\theta_1 + 5\theta_2 + 1997\theta_3 + 2010\theta_4 \end{bmatrix}$$