

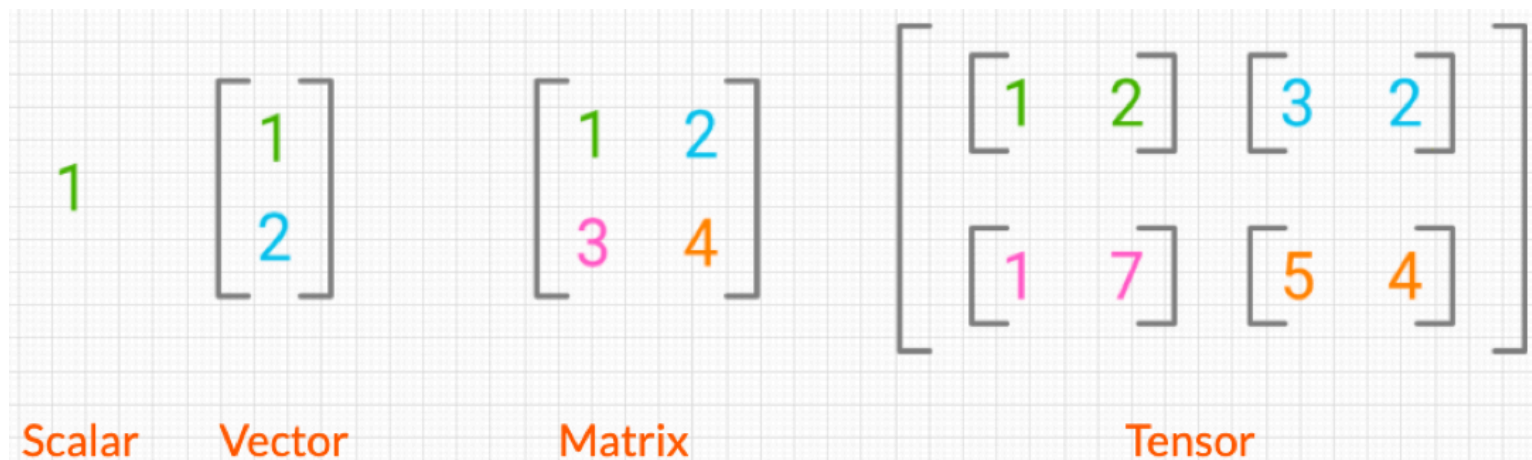


Introduction to library numpy



numpy

The library for working with **arrays**





numpy

Examples of 3D-arrays

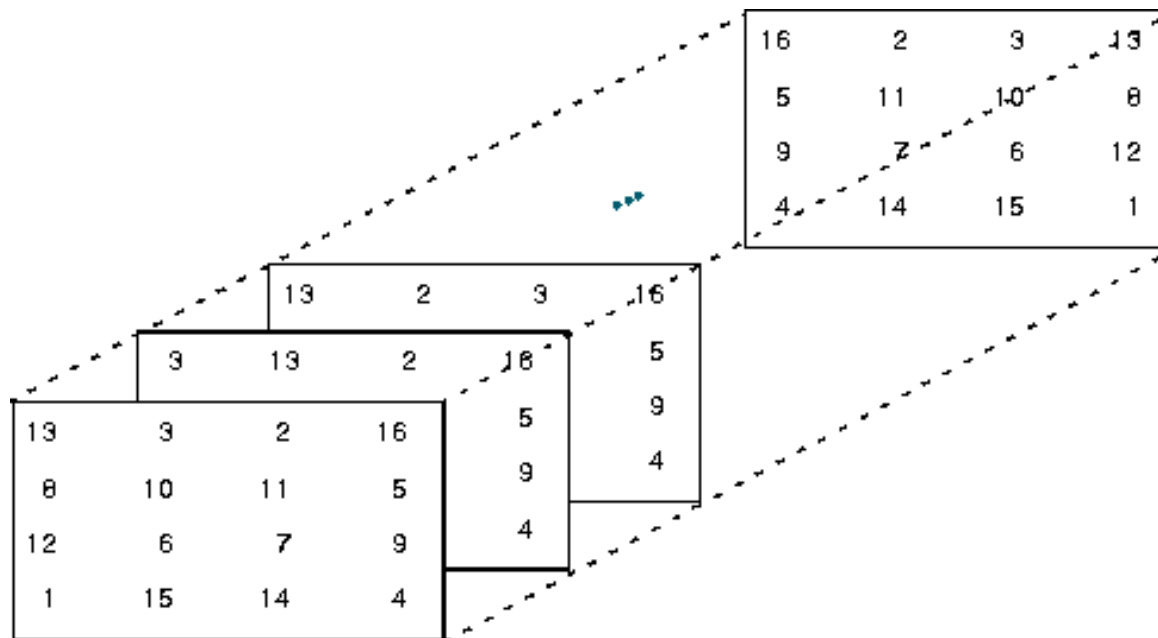
13	2	3	16
3	13	2	16
13	3	2	16
8	10	11	5
12	6	7	9
1	15	14	4

```
array([[[5, 8, 9, 5, 0],
        [0, 1, 7, 6, 9],
        [2, 4, 5, 2, 4],
        [2, 4, 7, 7, 9]],
       [[1, 7, 0, 6, 9],
        [9, 7, 6, 9, 1],
        [0, 1, 8, 8, 3],
        [9, 8, 7, 3, 6]],
       [[5, 1, 9, 3, 4],
        [8, 1, 4, 0, 3],
        [9, 2, 0, 4, 9],
        [2, 7, 7, 9, 8]]])
```



numpy

A tensor is a high-dimensional array





numpy

Used to perform math operations
(product, inverse, transpose),
transformations,
on arrays



numpy array

The main data structure is the array (**ndarray**)

It is a grid of numeric values of the same data type

- vectors 1D arrays
- matrices 2D arrays
- high-dimensional arrays nD arrays



numpy array

We will review

- how to create arrays
- array attributes
- how to transform arrays
- matrix operations



shape of 1D array

```
import numpy as np
```

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

```
array([5, 1, 2, 4, 5, 1, 5])
```

```
x.shape
```

```
(7,)  
  ↑
```

convert a list
to a 1D array



Creating a 2D array

```
import numpy as np
```

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

x

```
array([[1, 3, 3],  
       [1, 4, 3],  
       [1, 3, 4]])
```



shape of a 2D array

```
import numpy as np
```

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

x

↑

↑

```
array([[1, 3, 3],  
       [1, 4, 3],  
       [1, 3, 4]])
```

```
x.shape
```

```
(3, 3)
```



np functions to create ndarrays

Function	Description
<code>np.array</code>	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
<code>np.arange</code>	Like the built-in <code>range</code> but returns an ndarray
<code>np.ones</code>	Produce an array of all 1s
<code>np.zeros</code>	Produce an array of all 0s
<code>np.empty</code>	Create new arrays by allocating new memory, but do not populate with any values
<code>np.full</code>	Produce an array of the given shape and dtype with all values set to the indicated "fill value"
<code>np.eye</code> , <code>identity</code>	Create a square $N \times N$ identity matrix (1s on the diagonal and 0s elsewhere)



INTRODUCTION – Creating ndarrays

```
# create a list
```

```
x = [5,1,2,4,5,1,5]  
x
```

```
[5, 1, 2, 4, 5, 1, 5]
```

```
len(x)
```

```
7
```

```
# convert it to an 1D-array
```

```
x = np.array(x)  
x
```

```
array([5, 1, 2, 4, 5, 1, 5])
```

```
x.shape
```

```
(7,)
```



INTRODUCTION – Creating ndarrays

```
# create a list
```

```
x = [5,1,2,4,5,1,5]
```

```
x
```

```
[5, 1, 2, 4, 5, 1, 5]
```

```
len(x)
```

```
7
```

```
# convert it to an 1D-array
```

```
x = np.array(x)
```

```
x
```

```
array([5, 1, 2, 4, 5, 1, 5])
```

```
x.shape
```

```
(7,)
```

```
# nested list (list of lists)
```

```
x = [[1,3,3],[1,4,3],[1,3,4]]
```

```
x
```

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

```
len(x)
```

```
3
```

```
# convert it to an nD-array
```

```
x = np.array(x)
```

```
x
```

```
array([[1, 3, 3],  
       [1, 4, 3],  
       [1, 3, 4]])
```

```
x.shape
```

```
(3, 3)
```



INTRODUCTION – Creating ndarrays

```
x = list(range(1,10))  
x
```

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

```
x = np.array(x)  
x
```

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



INTRODUCTION – Creating ndarrays

```
x = list(range(1,10))  
x
```

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

```
x = np.array(x)  
x
```

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

```
x = np.arange(1,10)  
x
```

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



INTRODUCTION – Creating ndarrays

```
x = list(range(1,10))
```

```
x
```

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

```
x = np.array(x)
```

```
x
```

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

```
x = np.arange(1,10)
```

```
x
```

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

```
x = np.arange(0,10,step=2)
```

```
x
```

```
array([0, 2, 4, 6, 8])
```

```
np.linspace(0,10,5)      # five values evenly spaced in (0,10)
```

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```



INTRODUCTION – Commonly used arrays

```
np.zeros(10)
```

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

```
np.zeros(10, dtype=int)
```

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

```
np.eye(3)
```

3x3 identity matrix

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



INTRODUCTION – Commonly used arrays

```
np.eye(3) # 3x3 identity matrix
```

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

```
x = np.empty((3,3))
```

Think of a matrix with NAs

```
x
```

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



INTRODUCTION – Commonly used arrays

```
np.ones((3,4),dtype=float)    # 3x4 array of ones
```

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

```
np.full((3,4),2.1)    # 3x4 array filled with same number
```

```
array([[2.1, 2.1, 2.1, 2.1],  
       [2.1, 2.1, 2.1, 2.1],  
       [2.1, 2.1, 2.1, 2.1]])
```



INTRODUCTION – Creating arrays with random values

Values from the uniform distribution

- UNIF(0,1) `np.random.random(shape)`
- UNIF(a,b) `np.random.uniform(a,b,shape)`
- DUNIF(a,b) `np.random.randint(a,b,shape)`

Values from the normal distribution

- N(0,1) `np.random.randn(shape)`
- N(μ, σ) `np.random.normal($\mu, \sigma, shape$)`



INTRODUCTION – Creating arrays with Uniform values

```
np.random.seed(9)
```

```
# U(0,1) and U(a,b)
```

```
x = np.random.random(size = (2,3))
```

```
x
```

fill the 2x3 array with U(0,1) values

```
array([[0.01037415, 0.50187459, 0.49577329],  
       [0.13382953, 0.14211109, 0.21855868]])
```

```
x = np.random.uniform(low=100,high=200,size = (2,3))
```

```
x
```

fill the 2x3 array with U(100,200) values

```
array([[141.85081805, 124.81011684, 108.40596512],  
       [134.54986401, 116.67763465, 187.85590855]])
```



INTRODUCTION – Array with Discrete Uniform values

```
# DUNIF(5,10,(3,4))
```

fill with integers from 5 to 9

```
np.random.randint(low=5,high=10,size=(3,4))
```

```
array([[8, 5, 7, 5],  
       [7, 9, 5, 9],  
       [7, 5, 6, 5]])
```



3x3 Array with Standard Normal values

```
# N(0,1)
```

```
x = np.random.randn(3,3)
```

```
x
```

```
array([[ 0.63589108,  1.74011731,  0.29668222],  
       [ 0.70750366,  1.82281576,  0.43076903],  
       [ 1.54272963, -0.90072117, -0.13712501]])
```



3x2 Array with Non-standard Normal values

```
# N(mean=10, s.deviation = 3)
```

```
np.random.normal(loc=10, scale=3, size=(3, 2))
```

```
array([[13.89273704, 12.0258135 ],  
       [10.09587435, 12.75443769],  
       [11.1415284 , 11.54910246]])
```



array attributes



INTRODUCTION – array attributes

```
np.random.seed(1)
x = np.random.randint(10,size=(3,4,5))
x
```

```
array([[[5, 8, 9, 5, 0],
        [0, 1, 7, 6, 9],
        [2, 4, 5, 2, 4],
        [2, 4, 7, 7, 9]],

       [[1, 7, 0, 6, 9],
        [9, 7, 6, 9, 1],
        [0, 1, 8, 8, 3],
        [9, 8, 7, 3, 6]],

       [[5, 1, 9, 3, 4],
        [8, 1, 4, 0, 3],
        [9, 2, 0, 4, 9],
        [2, 7, 7, 9, 8]]])
```

```
np.random.randint(low=5,high=10,size=(3,4))
```

```
array([[8, 5, 7, 5],
       [7, 9, 5, 9],
       [7, 5, 6, 5]])
```



INTRODUCTION – ndarrays attributes

```
np.random.seed(1)
x = np.random.randint(10,size=(3,4,5))
x
```

```
array([[[5, 8, 9, 5, 0],
        [0, 1, 7, 6, 9],
        [2, 4, 5, 2, 4],
        [2, 4, 7, 7, 9]],

       [[1, 7, 0, 6, 9],
        [9, 7, 6, 9, 1],
        [0, 1, 8, 8, 3],
        [9, 8, 7, 3, 6]],

       [[5, 1, 9, 3, 4],
        [8, 1, 4, 0, 3],
        [9, 2, 0, 4, 9],
        [2, 7, 7, 9, 8]]])
```

x.ndim

3

x.shape

(3, 4, 5)

x.size

60



INTRODUCTION – ndarrays attributes

```
np.random.seed(1)
x = np.random.randint(10,size=(3,4,5))
x
```

```
array([[[5, 8, 9, 5, 0],
        [0, 1, 7, 6, 9],
        [2, 4, 5, 2, 4],
        [2, 4, 7, 7, 9]],

       [[1, 7, 0, 6, 9],
        [9, 7, 6, 9, 1],
        [0, 1, 8, 8, 3],
        [9, 8, 7, 3, 6]],

       [[5, 1, 9, 3, 4],
        [8, 1, 4, 0, 3],
        [9, 2, 0, 4, 9],
        [2, 7, 7, 9, 8]]])
```

x.ndim

3

x.shape

(3, 4, 5)

x.size

60

x[0,0,2]

9

matrix 0, row 0, col 2



INTRODUCTION – Changing the shape of arrays

1D array

```
x = list(range(1,10))
```

```
x
```

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

```
array1D = np.array(x)
```

```
array1D
```

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

```
array1D.shape
```

```
(9,)
```

```
array1D.ndim
```

```
1
```



INTRODUCTION – reshaping arrays

1D array

```
x = list(range(1,10))  
x
```

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

```
array1D = np.array(x)  
array1D
```

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

```
array1D.shape
```

```
(9,)
```

```
array1D.ndim
```

```
1
```

Convert this 1D array to 2D array

```
# reshape with -1 to keep dimension (9,)
```

```
array2D = array1D.reshape((-1,1))  
array2D
```

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

```
array2D.shape
```

```
(9, 1)
```

```
array2D.ndim
```

```
2
```



INTRODUCTION – reshaping arrays

1D array

```
x = list(range(1,10))
```

```
x
```

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

```
array1D = np.array(x)
```

```
array1D
```

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

```
array1D.shape
```

```
(9,)
```

different
shapes

```
array1D.ndim
```

```
1
```

Convert this 1D array to 2D array

```
# reshape with -1 to keep dimension (9,)
```

```
array2D = array1D.reshape((-1,1))
```

```
array2D
```

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

```
array2D.shape
```

```
(9, 1)
```

```
array2D.ndim
```

```
2
```



INTRODUCTION – concatenate arrays

```
array2
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
# concatenate
```

```
np.concatenate([array2,array2],axis = 0)
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8],  
       [1, 2, 3, 4],  
       [5, 6, 7, 8]])
```



INTRODUCTION – concatenate arrays

```
array2
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
# concatenate
```

```
np.concatenate([array2,array2],axis = 0)
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8],  
       [1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
np.concatenate([array2,array2],axis = 1)
```

```
array([[1, 2, 3, 4, 1, 2, 3, 4],  
       [5, 6, 7, 8, 5, 6, 7, 8]])
```



INTRODUCTION – concatenate arrays

```
array2
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
# concatenate
```

```
np.concatenate([array2,array2],axis = 0)
```

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8],  
       [1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
np.concatenate([array2,array2],axis = 1)
```

```
array([[1, 2, 3, 4, 1, 2, 3, 4],  
       [5, 6, 7, 8, 5, 6, 7, 8]])
```



matrix operations



INTRODUCTION – matrix operations

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

```
array([[4, 3],  
       [2, 1]])
```

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

```
array([[1, 2],  
       [3, 4]])
```



INTRODUCTION – matrix operations

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

```
array([[4, 3],  
       [2, 1]])
```

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

```
array([[1, 2],  
       [3, 4]])
```

```
c*d
```

```
array([[4, 6],  
       [6, 4]])
```



INTRODUCTION – matrix operations

Elementwise multiplication

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

```
array([[4, 3],  
       [2, 1]])
```

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

```
array([[1, 2],  
       [3, 4]])
```

```
c*d
```

```
array([[4, 6],  
       [6, 4]])
```



INTRODUCTION – matrix operations

Elementwise multiplication

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

```
array([[4, 3],  
       [2, 1]])
```

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

```
array([[1, 2],  
       [3, 4]])
```

```
c*d
```

```
array([[4, 6],  
       [6, 4]])
```



INTRODUCTION – matrix operations

Elementwise multiplication

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

```
array([[4, 3],  
       [2, 1]])
```

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

```
array([[1, 2],  
       [3, 4]])
```

```
c*d
```

```
array([[4, 6],  
       [6, 4]])
```



INTRODUCTION – matrix operations

Elementwise multiplication

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

```
array([[4, 3],  
       [2, 1]])
```

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

```
array([[1, 2],  
       [3, 4]])
```

```
c*d
```

```
array([[4, 6],  
       [6, 4]])
```



INTRODUCTION – Create two arrays, a and b

```
x = np.arange(1,7)
```

```
x
```

```
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
```

```
a
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

```
b = x.reshape(3,2)
```

```
b
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```



INTRODUCTION – matrix multiplication

```
x = np.arange(1,7)
x
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
a
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b = x.reshape(3,2)
b
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.dot(b)
array([[22, 28],
       [49, 64]])
```



INTRODUCTION – matrix multiplication

```
x = np.arange(1,7)
x
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
a
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b = x.reshape(3,2)
b
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.dot(b)
array([[22, 28],
       [49, 64]])
```

$$1(1) + 2(3) + 3(5) = 22$$



INTRODUCTION – matrix multiplication

```
x = np.arange(1,7)
x
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
a
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b = x.reshape(3,2)
b
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.dot(b)
```

```
array([[22, 28],
       [49, 64]])
```

$$1(2) + 2(4) + 3(6) = 28$$



INTRODUCTION – matrix multiplication

```
x = np.arange(1,7)
x
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
a
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b = x.reshape(3,2)
b
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.dot(b)
array([[22, 28],
       [49, 64]])
```

$$4(1) + 5(3) + 6(5) = 49$$



INTRODUCTION – matrix multiplication notation

```
x = np.arange(1,7)
x
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
a
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b = x.reshape(3,2)
b
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
a.dot(b)
array([[22, 28],
       [49, 64]])
```

```
np.dot(a,b)
array([[22, 28],
       [49, 64]])
```



INTRODUCTION – Transpose of a matrix

```
x = np.arange(1,7)
x
```

```
array([1, 2, 3, 4, 5, 6])
```

```
a = x.reshape(2,3)
a
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
a.T
```

Transpose of matrix a

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```



INTRODUCTION

```
# x'x is the sum of squares of x
```

```
x
```

```
array([1, 2, 3, 4, 5, 6])
```

```
np.dot(x.T, x)
```

```
91
```

1D array



$$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 = 91$$



INTRODUCTION – Create a square matrix from a vector

```
x = np.arange(1,10)
x[2] = 0
x
```

```
array([1, 2, 0, 4, 5, 6, 7, 8, 9])
```

```
x = x.reshape(3,3)
x
```

```
array([[1, 2, 0],
       [4, 5, 6],
       [7, 8, 9]])
```



INTRODUCTION – operations on square matrices

```
x = np.arange(1,10)
```

```
x[2] = 0
```

```
x
```

```
array([1, 2, 0, 4, 5, 6, 7, 8, 9])
```

```
x = x.reshape(3,3)
```

```
x
```

```
array([[1, 2, 0],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
np.diag(x)
```

```
array([1, 5, 9])
```

```
np.trace(x)
```

```
15
```

```
np.linalg.det(x)
```

```
9.0000000000000002
```

```
from numpy import linalg
```

```
linalg.det(x)
```

```
9.0000000000000002
```



INTRODUCTION – operations on square matrices

```
x = np.arange(1,10)
x[2] = 0
x
```

```
array([1, 2, 0, 4, 5, 6, 7, 8, 9])
```

```
x = x.reshape(3,3)
x
```

```
array([[1, 2, 0],
       [4, 5, 6],
       [7, 8, 9]])
```

```
y = linalg.inv(x)
y
```

inverse of matrix x

```
array([[ -0.33333333, -2.          ,  1.33333333],
       [ 0.66666667,  1.          , -0.66666667],
       [-0.33333333,  0.66666667, -0.33333333]])
```

```
np.diag(x)
```

```
array([1, 5, 9])
```

```
np.trace(x)
```

```
15
```

```
np.linalg.det(x)
```

```
9.0000000000000002
```

```
from numpy import linalg
```

```
linalg.det(x)
```

```
9.0000000000000002
```



Example

Linear Regression



Linear Regression

Use numpy, sklearn, and matplotlib to

- Display the relation between two variables x , y
- Find the regression line
- Predict y given X
- Draw a scatterplot with the regression line



Linear Regression

- Car dealers use the *Red Book* to estimate the price of used cars
- This book (published monthly) lists the trade-in prices for all basic models of cars
- However, the Red Book does not indicate the value determined by the odometer reading, despite the fact that a critical factor for used-car buyers is **how far the car has been driven**



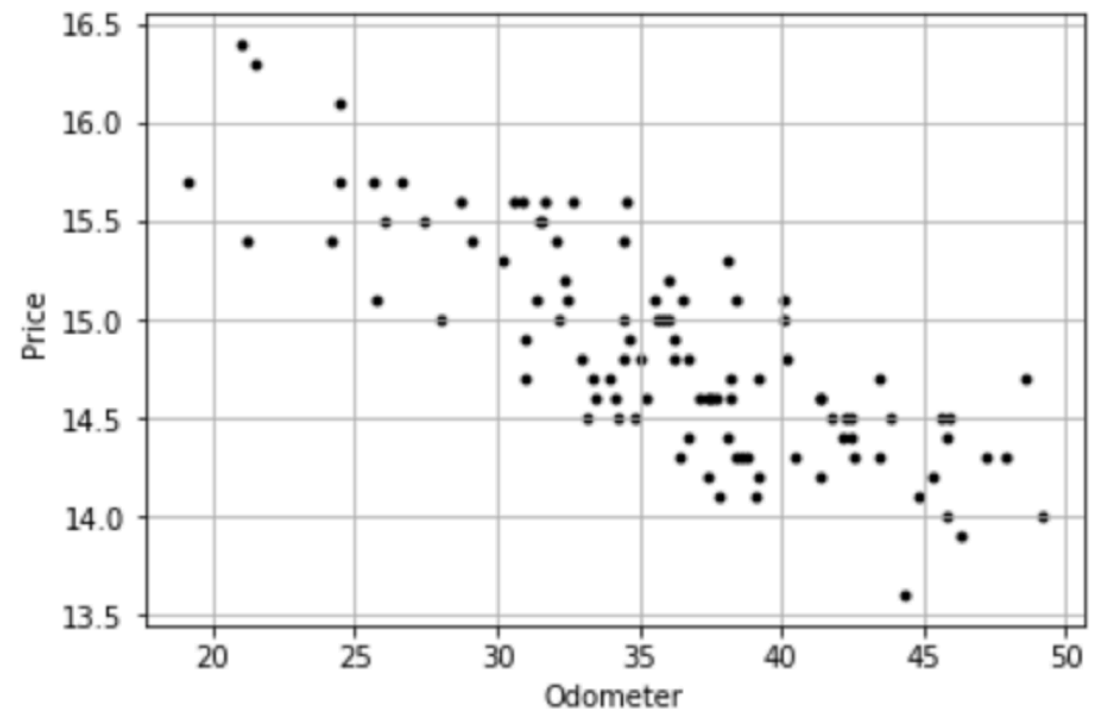
Linear Regression

- To examine this issue, a used-car dealer randomly selected 100 three-year old Toyota Camrys that were sold at an auction during the past month.
- The dealer recorded the price (000s) and the number of miles (000s) on the odometer (Odometer.csv)
- How much does the number of miles affect the price of a used-car?



Odometer.csv

	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	30.9	15.6
6	31.7	15.6
7	34	14.7
8	45.9	14.5
9	19.1	15.7
10	40.1	15.1
11	40.2	14.8

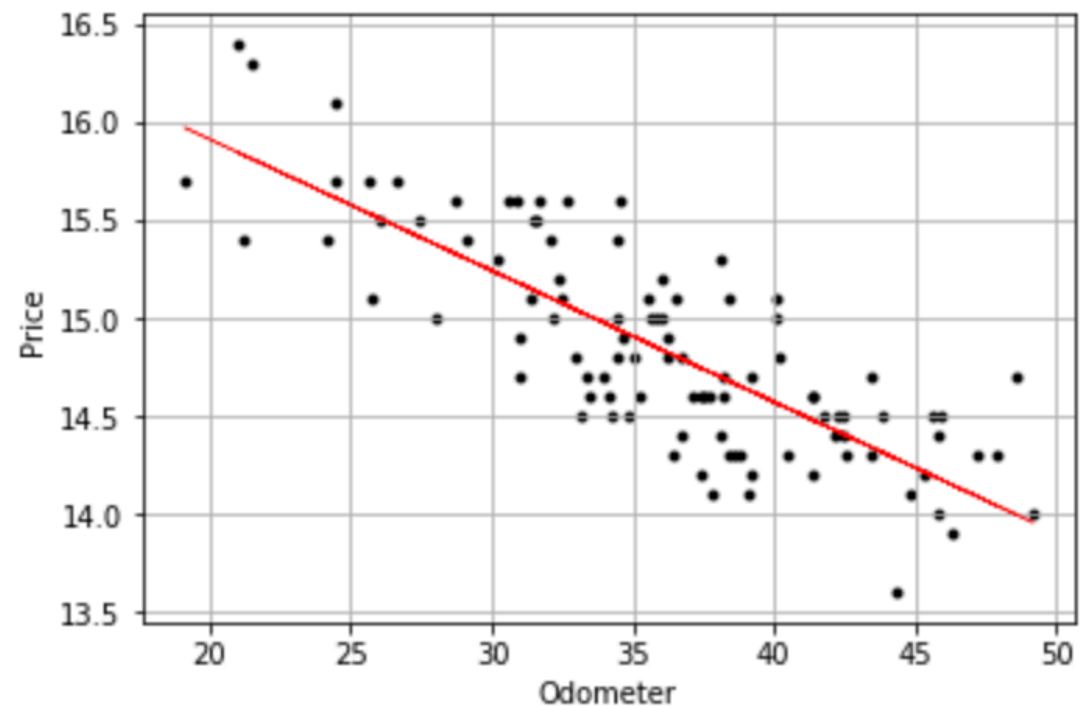


100 x 2



Odometer.csv

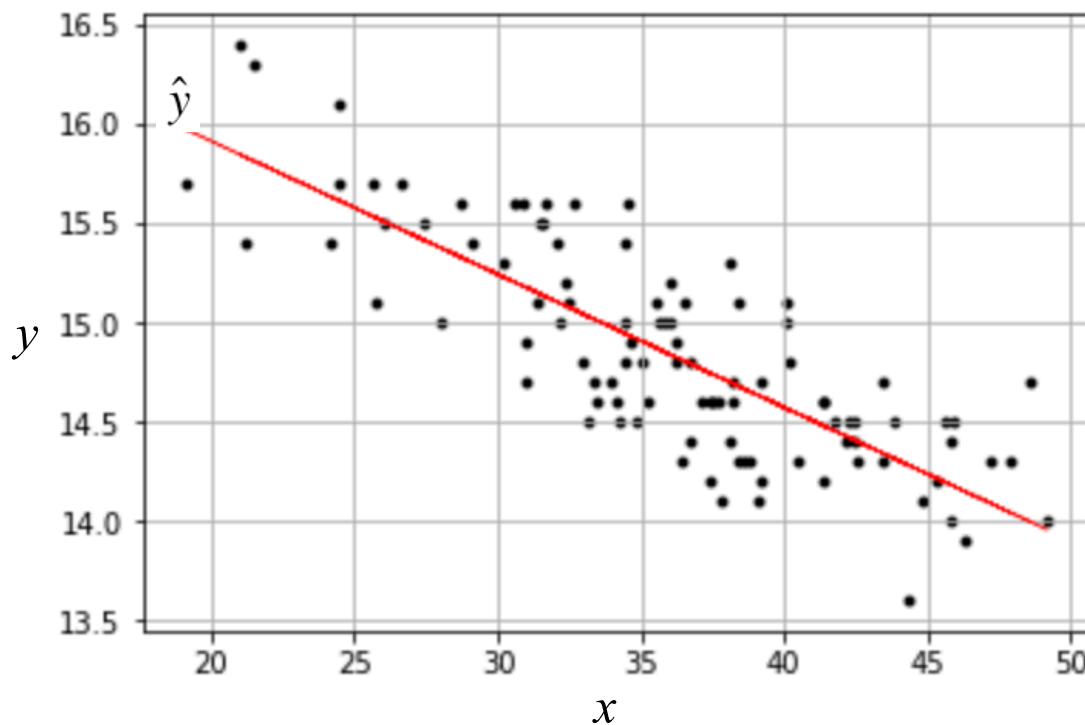
	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	30.9	15.6
6	31.7	15.6
7	34	14.7
8	45.9	14.5
9	19.1	15.7
10	40.1	15.1
11	40.2	14.8



100 x 2



Linear Regression



$$\hat{y} = b_0 + b_1 x = \overset{\text{intercept}}{17.250} - \overset{\text{slope}}{0.0669}x$$

$$b = [b_0, b_1] \\ = [17.25, -0.0669]$$



Odometer.csv

	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	X 30.9	y 15.6
6	31.7	15.6
7	34	14.7
8	45.9	14.5
9	19.1	15.7
10	40.1	15.1
11	40.2	14.8

headers



Reading csv file into an array

```
import numpy as np
import matplotlib.pyplot as plt
```

```
array1 = np.genfromtxt('Odometer.csv', skip_header=1, delimiter=',')
array1[:5]
```

```
array([[37.4, 14.6],
       [44.8, 14.1],
       [45.8, 14. ],
       [30.9, 15.6],
       [31.7, 15.6]])
```

```
array1.shape
```

```
(100, 2)
```

	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	30.9	15.6
6	31.7	15.6

x

y



Reading csv file with numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
array1 = np.genfromtxt('Odometer.csv', skip_header=1, delimiter=',')
array1[:5]
```

x y

```
array([[37.4, 14.6],
       [44.8, 14.1],
       [45.8, 14. ],
       [30.9, 15.6],
       [31.7, 15.6]])
```

```
array1.shape
```

```
(100, 2)
```



Reading csv file with numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
array1 = np.genfromtxt('Odometer.csv', skip_header=1, delimiter=',')
array1[:5]
```

```
      x      y
array([[37.4, 14.6],
       [44.8, 14.1],
       [45.8, 14. ],
       [30.9, 15.6],
       [31.7, 15.6]])
```

```
array1.shape
```

```
(100, 2)
```

```
x, y = array1[:,0], array1[:,1]
x[:5]
```

```
array([37.4, 44.8, 45.8, 30.9, 31.7])
```

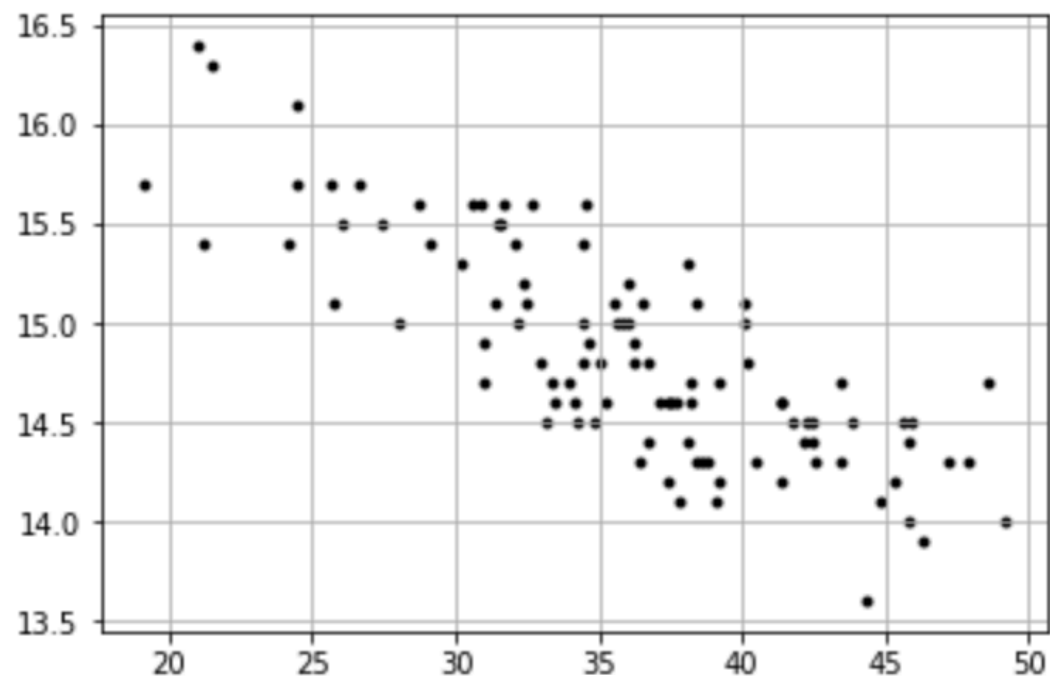
1D arrays



Scatterplot with matplotlib

```
plt.figure()  
plt.scatter(x,y,c='k',s=9)
```

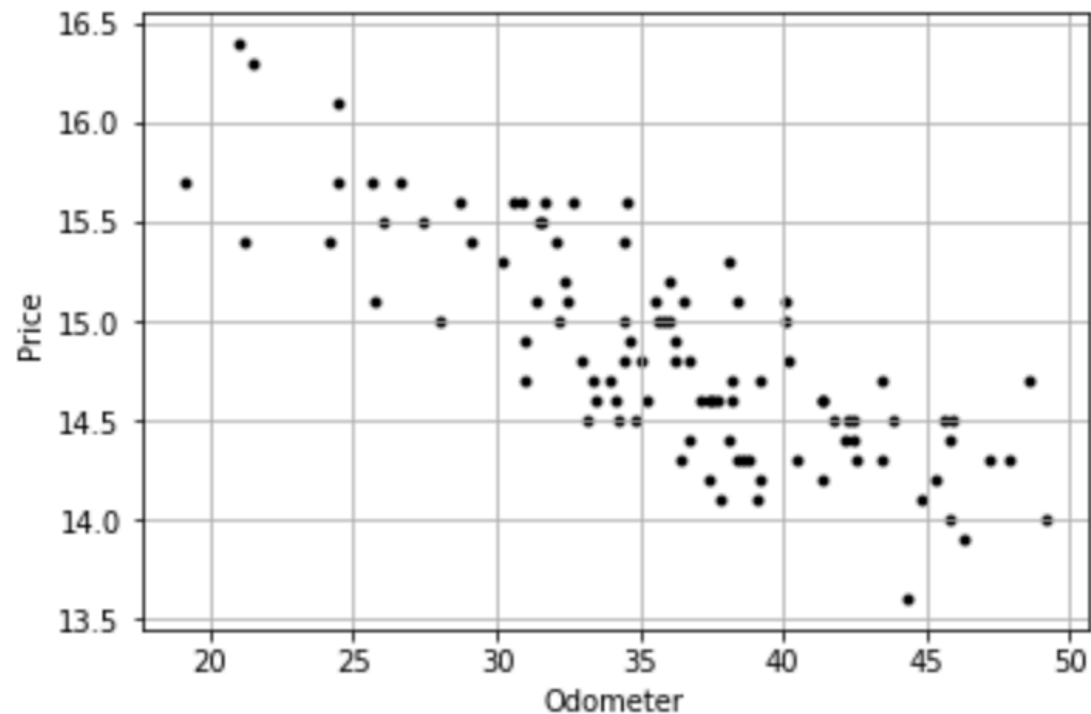
```
plt.grid()
```





Scatterplot with matplotlib

```
plt.figure()  
plt.scatter(x,y,c='k',s=9)  
plt.ylabel('Price')  
plt.xlabel('Odometer')  
plt.grid()
```





Linear Regression with sklearn



Linear regression with sklearn

```
from sklearn.linear_model import LinearRegression
```

```
x.shape
```

```
(100,)
```

for sklearn
we need x
to be 2D array



Linear regression with sklearn

```
from sklearn.linear_model import LinearRegression
```

```
x.shape
```

```
(100,)
```

1D array

```
x_2d = x.reshape(-1, 1)  
x_2d.shape
```

```
(100, 1)
```

2D array

```
m1 = LinearRegression().fit(x_2d,y)
```



Linear regression with sklearn

```
from sklearn.linear_model import LinearRegression
```

```
x.shape
```

```
(100,)
```

1D array

```
x_2d = x.reshape(-1, 1)  
x_2d.shape
```

```
(100, 1)
```

2D array

```
m1 = LinearRegression().fit(x_2d,y)
```

for multiple regression
with p predictors,
this array should be of
(100, p) shape



Linear regression with sklearn

```
from sklearn.linear_model import LinearRegression
```

```
x.shape
```

```
(100,)
```

```
x_2d = x.reshape(-1, 1)  
x_2d.shape
```

```
(100, 1)
```

```
m1 = LinearRegression().fit(x_2d,y)
```

```
m1.intercept_
```

```
17.24872734291551
```

```
m1.coef_
```

```
array([-0.06686089])
```

$$\begin{aligned} b &= [b_0, b_1] \\ &= [17.25, -0.0669] \end{aligned}$$



Linear regression with sklearn

Predict the price of a car with Odometer reading 40

```
newval = np.array([[40]])  
newval
```

```
array([[40]])
```

```
newval.shape
```

```
(1, 1)
```

newval must be a
2D numpy array



Predicting with model m1

Predict the price of a car with Odometer reading 40

```
newval = np.array([[40]])  
newval
```

```
array([[40]])
```

```
newval.shape
```

```
(1, 1)
```

```
m1.predict(newval)
```

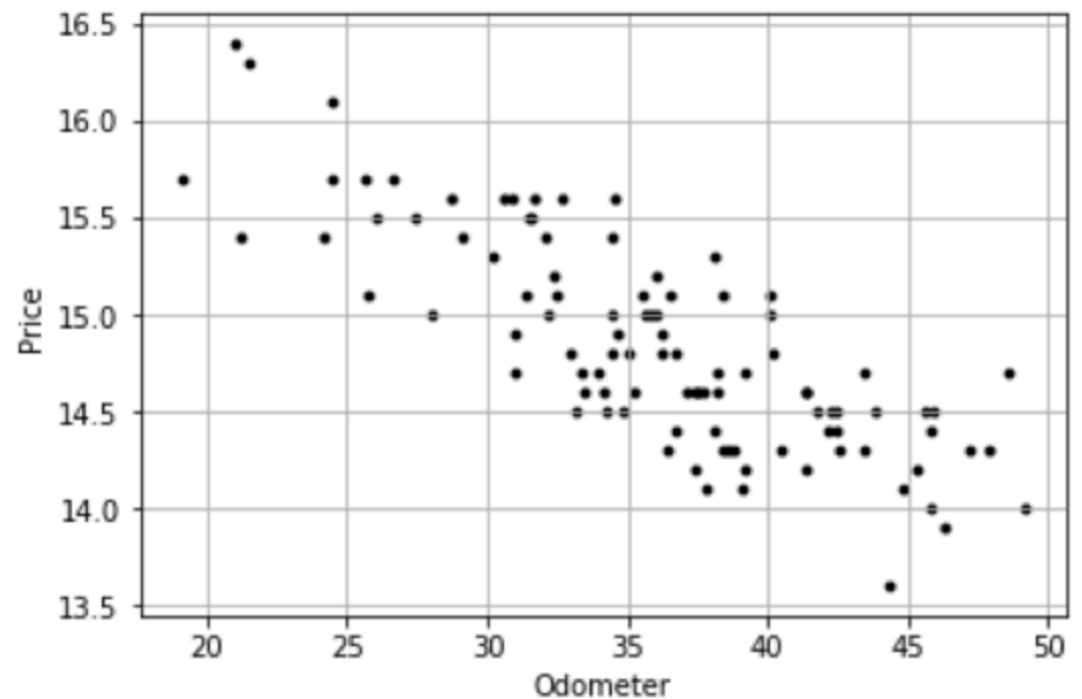
```
array([14.57429193])
```

car price is predicted
to be 14,574 dollars



Scatterplot

```
plt.figure()  
plt.scatter(x,y,c='k',s=9)
```



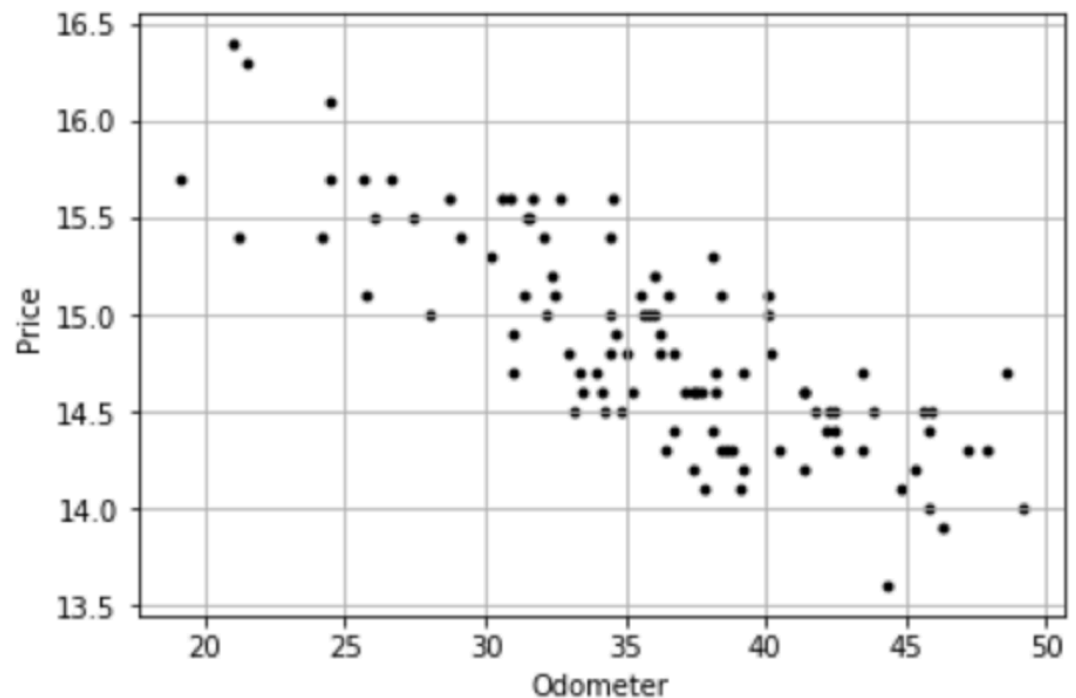


Scatterplot with regression line

```
# predict price  
# for all Odometer values in x
```

```
yhat = ml.predict(x_2d)
```

```
plt.figure()  
plt.scatter(x,y,c='k',s=9)
```



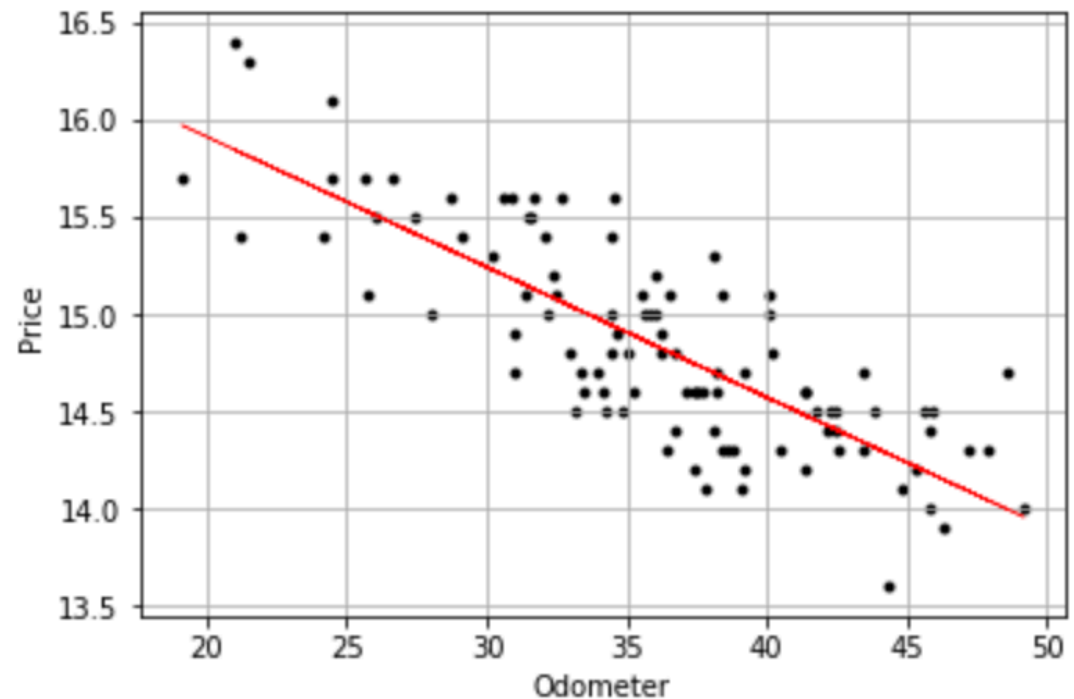


Scatterplot with regression line

```
# predict price  
# for all Odometer values in x
```

```
yhat = ml.predict(x_2d)
```

```
plt.figure()  
plt.scatter(x,y,c='k',s=9)  
plt.plot(x,yhat,color = 'r',  
         linewidth = 0.5)
```





Linear Regression with **numpy**



Odometer.csv

	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	X 30.9	y 15.6
6	31.7	15.6
7	34	14.7
8	45.9	14.5
9	19.1	15.7
10	40.1	15.1
11	40.2	14.8



Insert a column of ones

	A	B	C
1		Odometer	Price
2	1	37.4	14.6
3	1	44.8	14.1
4	1	45.8	14
5	1	30.9	15.6
6	1	31.7	15.6
7	1	34	14.7
8	1	45.9	14.5
9	1	19.1	15.7
10	1	40.1	15.1
11	1	40.2	14.8



Formula for Linear Regression coefficients

	A	B	C
1		Odometer	Price
2	1	37.4	14.6
3	1	44.8	14.1
4	1	45.8	14
5	1	30.9	15.6
6	1	31.7	15.6
7	1	34	14.7
8	1	45.9	14.5
9	1	19.1	15.7
10	1	40.1	15.1
11	1	40.2	14.8

regression formula

$$b = (X'X)^{-1} X'y$$

$$b = [b_0, b_1]$$

$$= [17.25, -0.0669]$$



Reading csv file with numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
array1 = np.genfromtxt('Odometer.csv', skip_header=1, delimiter=',')
array1[:5]
```

```
array([[37.4, 14.6],
       [44.8, 14.1],
       [45.8, 14. ],
       [30.9, 15.6],
       [31.7, 15.6]])
```

```
array1.shape
```

```
(100, 2)
```

	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	30.9	15.6
6	31.7	15.6

x

y



Reading csv file with numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
array1 = np.genfromtxt('Odometer.csv', skip_header=1, delimiter=',')
array1[:5]
```

x **y**

```
array([[37.4, 14.6],
       [44.8, 14.1],
       [45.8, 14. ],
       [30.9, 15.6],
       [31.7, 15.6]])
```

```
array1.shape
```

```
(100, 2)
```

	A	B
1	Odometer	Price
2	37.4	14.6
3	44.8	14.1
4	45.8	14
5	30.9	15.6
6	31.7	15.6



Reading csv file with numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
array1 = np.genfromtxt('Odometer.csv', skip_header=1, delimiter=',')
array1[:5]
```

```
array([[37.4, 14.6],
       [44.8, 14.1],
       [45.8, 14. ],
       [30.9, 15.6],
       [31.7, 15.6]])
```

The image shows the first five rows of a 2D array. The first column is labeled 'x' and the second column is labeled 'y'. Each column is enclosed in a red box.

```
array1.shape
```

```
(100, 2)
```

```
x, y = array1[:,0], array1[:,1]
x[:5]
```

```
array([37.4, 44.8, 45.8, 30.9, 31.7])
```

1D arrays




Linear regression with numpy

```
x, y = array1[:,0], array1[:,1]
x[:5]

array([37.4, 44.8, 45.8, 30.9, 31.7])
```

```
# add column of ones to x
```

```
x1 = np.c_[np.ones(100),x]
x1[:5]
```

A diagram with two red arrows. The first arrow starts from the 'np.ones(100)' part of the line 'x1 = np.c_[np.ones(100),x]' and points down to the first column of the resulting array. The second arrow starts from the 'x' part of the same line and points down to the second column of the resulting array.

```
array([[ 1. , 37.4],
       [ 1. , 44.8],
       [ 1. , 45.8],
       [ 1. , 30.9],
       [ 1. , 31.7]])
```



Linear regression with numpy

```
x, y = array1[:,0], array1[:,1]
x[:5]

array([37.4, 44.8, 45.8, 30.9, 31.7])
```

```
# add column of ones to x
```

```
x1 = np.c_[np.ones(100),x]
x1[:5]
```

```
array([[ 1. , 37.4],
       [ 1. , 44.8],
       [ 1. , 45.8],
       [ 1. , 30.9],
       [ 1. , 31.7]])
```

	A	B	C
1		Odometer	Price
2	1	37.4	14.6
3	1	44.8	14.1
4	1	45.8	14
5	1	30.9	15.6
6	1	31.7	15.6

X1

y



Linear regression with numpy

```
x, y = array1[:,0], array1[:,1]
x[:5]

array([37.4, 44.8, 45.8, 30.9, 31.7])
```

```
# add column of ones to x
```

```
x1 = np.c_[np.ones(100),x]
x1[:5]
```

```
array([[ 1. , 37.4],
       [ 1. , 44.8],
       [ 1. , 45.8],
       [ 1. , 30.9],
       [ 1. , 31.7]])
```

	A	B	C
1		Odometer	Price
2	1	37.4	14.6
3	1	44.8	14.1
4	1	45.8	14
5	1	30.9	15.6
6	1	31.7	15.6

$$b = (X'X)^{-1} X'y$$





Linear regression with numpy

```
x, y = array1[:,0], array1[:,1]
x[:5]

array([37.4, 44.8, 45.8, 30.9, 31.7])
```

```
# add column of ones to x
```

```
x1 = np.c_[np.ones(100),x]
x1[:5]
```

```
array([[ 1. , 37.4],
       [ 1. , 44.8],
       [ 1. , 45.8],
       [ 1. , 30.9],
       [ 1. , 31.7]])
```

$$b = \underbrace{(X'X)^{-1}}_{b1} \underbrace{X'y}_{b2}$$

```
b1 = np.dot(x1.T,x1)
b1 = np.linalg.inv(b1)
b1
```

 $(X^T X)^{-1}$

```
array([[ 3.11063002e-01, -8.36030663e-03],
       [-8.36030663e-03,  2.32159802e-04]])
```

```
b2 = np.dot(x1.T,y)
b2
```

 $X^T y$

```
array([ 1484.1 , 53155.93])
```



Linear regression with numpy

```
x, y = array1[:,0], array1[:,1]
x[:5]

array([37.4, 44.8, 45.8, 30.9, 31.7])
```

```
# add column of ones to x
```

```
x1 = np.c_[np.ones(100),x]
x1[:5]
```

```
array([[ 1. , 37.4],
       [ 1. , 44.8],
       [ 1. , 45.8],
       [ 1. , 30.9],
       [ 1. , 31.7]])
```

$$b = (X'X)^{-1} X'y$$

```
b1 = np.dot(x1.T,x1)
b1 = np.linalg.inv(b1)
b1
```

$$(X^T X)^{-1}$$

```
array([[ 3.11063002e-01, -8.36030663e-03],
       [-8.36030663e-03,  2.32159802e-04]])
```

```
b2 = np.dot(x1.T,y)
b2
```

$$X^T y$$

```
array([ 1484.1 , 53155.93])
```

```
coeffs = np.dot(b1,b2)
coeffs
```

$$(X^T X)^{-1} X^T y$$

```
array([17.24872734, -0.06686089])
```

intercept

slope



Linear regression with numpy

```
x, y = array1[:,0], array1[:,1]
x[:5]

array([37.4, 44.8, 45.8, 30.9, 31.7])
```

```
# add column of ones to x
```

```
x1 = np.c_[np.ones(100),x]
x1[:5]
```

```
array([[ 1. , 37.4],
       [ 1. , 44.8],
       [ 1. , 45.8],
       [ 1. , 30.9],
       [ 1. , 31.7]])
```

$$b = (X'X)^{-1} X'y$$

```
b1 = np.dot(x1.T,x1)
b1 = np.linalg.inv(b1)
b1
```

$$(X^T X)^{-1}$$

```
array([[ 3.11063002e-01, -8.36030663e-03],
       [-8.36030663e-03,  2.32159802e-04]])
```

```
b2 = np.dot(x1.T,y)
b2
```

$$X^T y$$

```
array([ 1484.1 , 53155.93])
```

```
coeffs = np.dot(b1,b2)
coeffs
```

$$(X^T X)^{-1} X^T y$$

```
array([17.24872734, -0.06686089])
```

INTERPRET THE SLOPE

Each additional mile
decreases the average price
by 6.69 cents



Predicting with numpy

Predicting with numpy
requires
coeffs to be a 2D array

```
coeffs.shape
```

```
(2,)
```

1D array

```
coeffs2 = coeffs.reshape(-1,1)  
coeffs2
```

```
array([[17.24872734],  
       [-0.06686089]])
```

```
coeffs2.shape
```

```
(2, 1)
```

2D array



Predict the price of a used car with 40 miles

$$\text{Prediction} = X b$$

```
coeffs.shape
```

```
(2,)
```

```
b coeffs2 = coeffs.reshape(-1,1)  
coeffs2
```

```
array([[17.24872734],  
       [-0.06686089]])
```

```
coeffs2.shape
```

```
(2, 1)
```

```
newval = np.array([[1,40]])  
newval
```

X

```
array([[ 1, 40]])
```

```
newval.shape
```

```
(1, 2)
```

```
np.dot(X, b)  
newval, coeffs2
```

```
array([[14.57429193]])
```

car price is predicted to be 14,574 dollars



Scatterplot with regression line

```
# predict price  
# for all Odometer values in x
```

```
yhat = np.dot(x1,coeffs)
```

```
plt.figure()  
plt.scatter(x,y,c='k',s=9)  
plt.plot(x,yhat,color = 'r',  
         linewidth = 0.5)
```

