setup

NOTE TO GRADER: I built this notebook using Marimo as I manage a Git repository of all Python notebooks for personal and academic purposes. Jupyter and Git mix like oil and water. This notebook is near identical to the Jupyter notebook with a few minor changes (that should not be of consequence but I have noted them in case they are).

- `from sympy import *` has been transformed into `import sympy as sp` due to Marimo not supporting wildcard imports.
- All imports are done in a singular setup cell instead of throughout the code. (this also pleases the CS major side of me where all imports should always be done at the top of your file)
- Some formatting regarding printing is slightly different than what would be expected from Jupyter Lab.

1.1 Motivating Example: Penguin Data

To begin our lab section, we will discuss a specific task that you may encounter as a data scientist: classifying penguin species from different features. Without knowing what species each pengunin is in a given row, we can identify different clusters in the data.

bill_length_mm float64	bill_depth_mm float64	flipper_length_mm float64	body_mass_g float64
39.1	18.7	181	3,750
39.5	17.4	186	3,800
40.3	18	195	3,250
NaN	NaN	NaN	NaN
36.7	19.3	193	3,450

5 rows, 4 columns

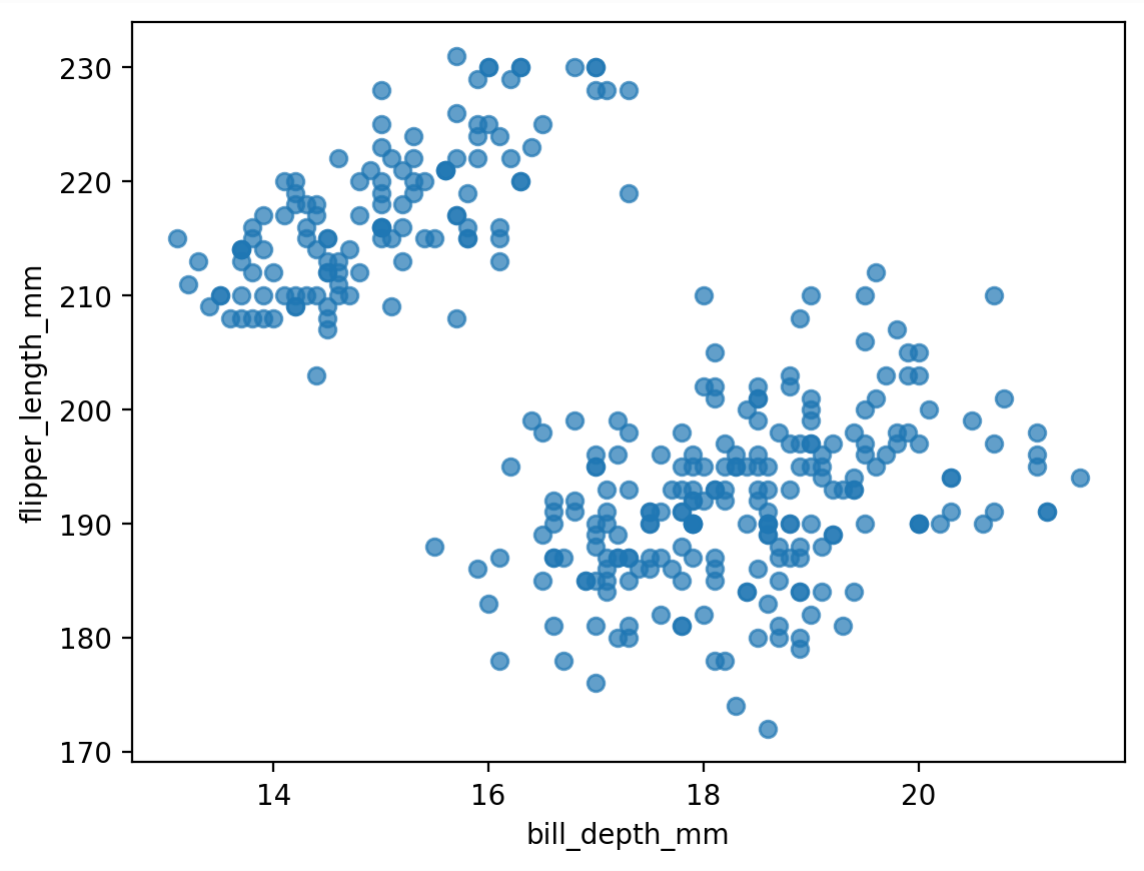
Page 1 of 1

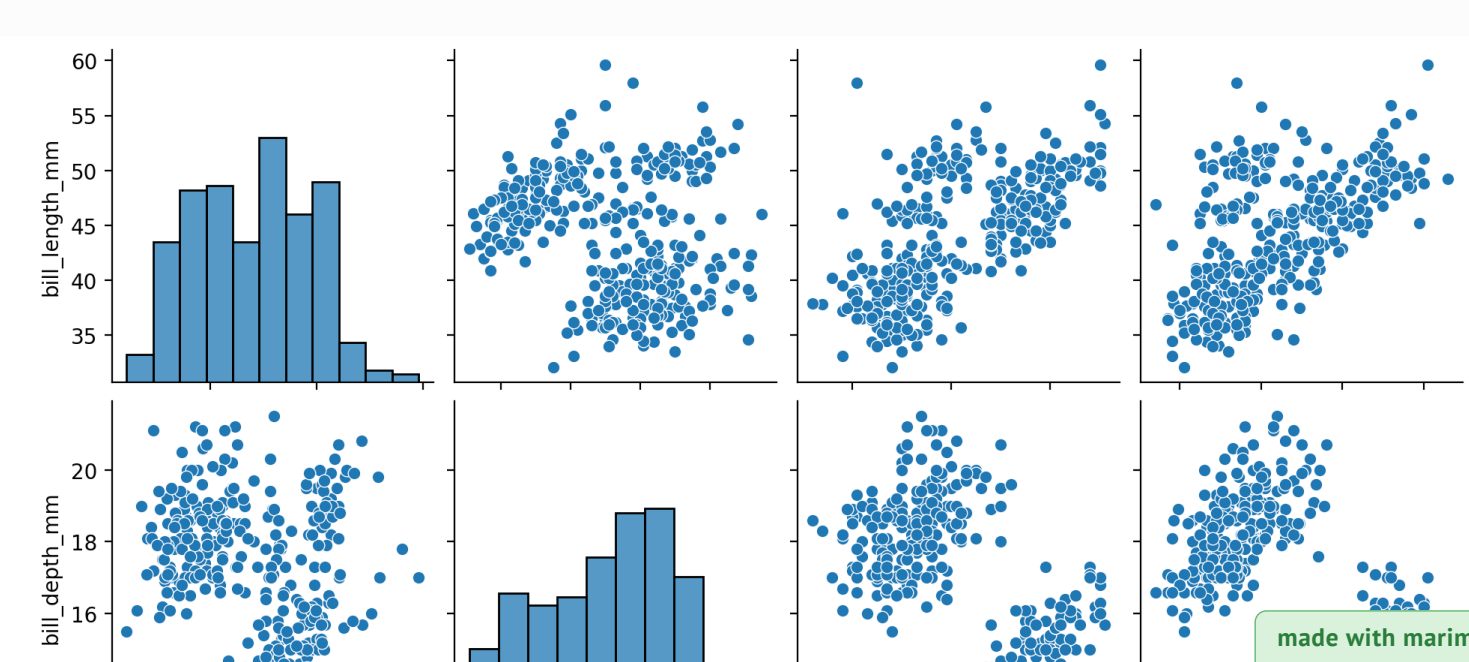
Download

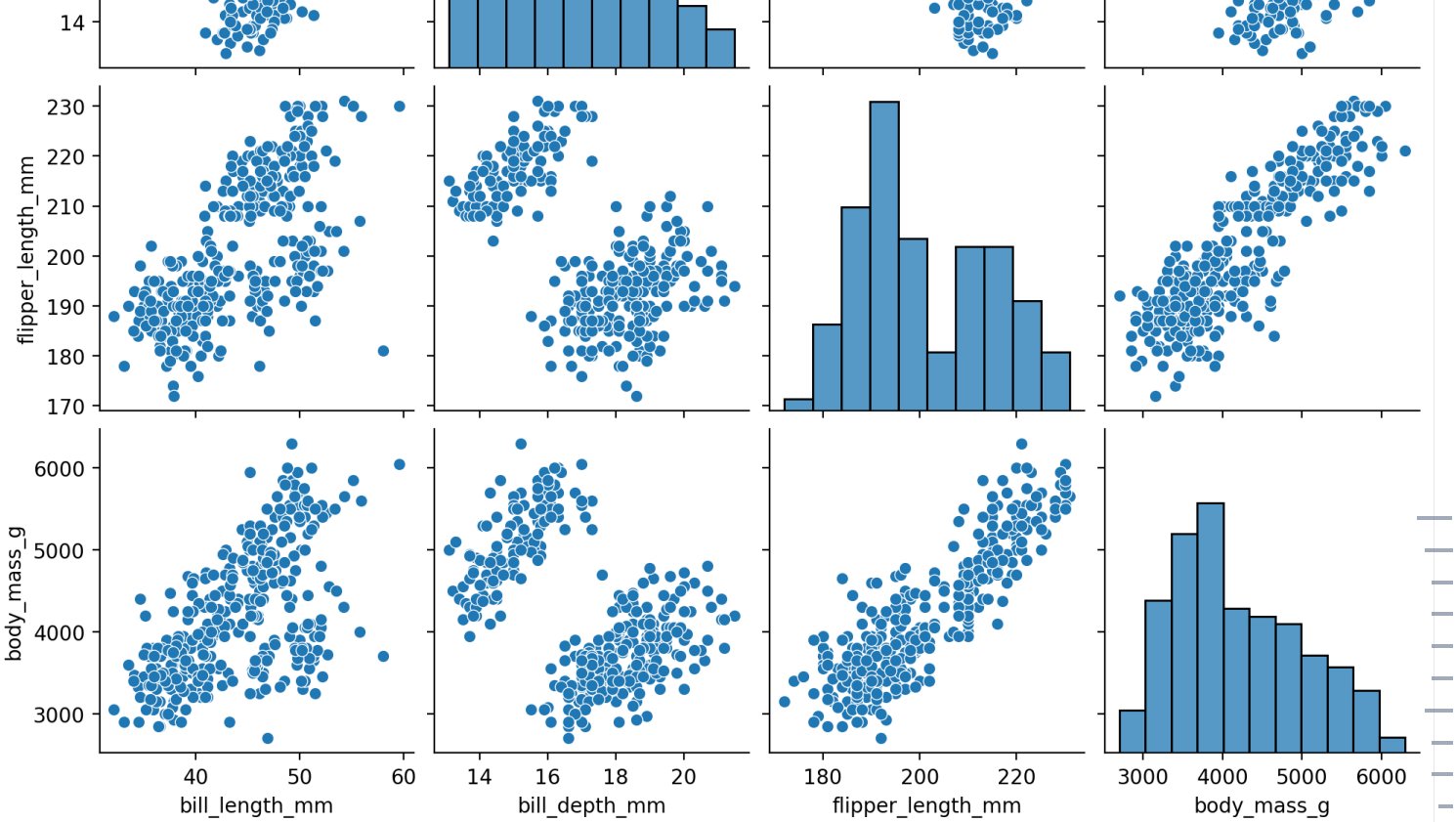
```
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
dtype: int64
```

```
Shape: (344, 4)
There are 344 rows (samples) and 4 columns (features).
```

```
X shape: (342, 4)
[[  39.1   18.7  181.  3750. ]
 [  39.5   17.4  186.  3800. ]
 [  40.3   18.   195.  3250. ]
 ...
 [  50.4   15.7  222.  5750. ]
 [  45.2   14.8  212.  5200. ]
 [  49.9   16.1  213.  5400. ]]
```







1.2.1 Vectors and Matrices (Linear Algebra)

```
[1. 3. 5. 7.]
```

```
First element: 1.0
Second element: 3.0
```

```
L2 Norm of u: 9.16515138991168
```

Sanity Check: Does this answer match with what you get using the formula for the Norm?

Hint: $||u||_2 = \sum_{i=1}^{d} u_i^2$

```
Manual calculation: 9.16515138991168
```

ℓ_1 vs ℓ_2 Norm

```
V L1 Norm: 10.0
W L1 Norm: 2.0

V L2 Norm: 5.477225575051661
W L2 Norm: 1.4142135623730951
```

P Norm Investigation

How does the P norm value change as P changes?

The p-norm of a vector

For a vector $u = (u_1, u_2, \ldots, u_d)$ and $p \geq 1$, the p-norm is defined as

$$\|u\|_p = \left(\sum_{i=1}^{d} |u_i|^p \right)^{1/p}.$$

As $p \to \infty$, the p-norm converges to the infinity norm:

$$\|u\|_\infty = \max_{1 \leq i \leq d} |u_i|.$$

Visualizing the p-norm as "circles"

A **norm** $\|\cdot\|$ defines a notion of length. Once we have a norm, we can define a distance between two points $x, c \in \mathbb{R}^d$ by:

$$d_p(x, c) = \|x - c\|_p.$$

In ordinary Euclidean geometry, the set of points at distance r from the origin is a **circle**:

$$\{x \in \mathbb{R}^2 : \|x\|_2 = r\}.$$

But if we change the norm, the "circle" changes shape. For $p \geq 1$:

$$\|x\|_p = \left(|x_1|^p + |x_2|^p\right)^{1/p}.$$

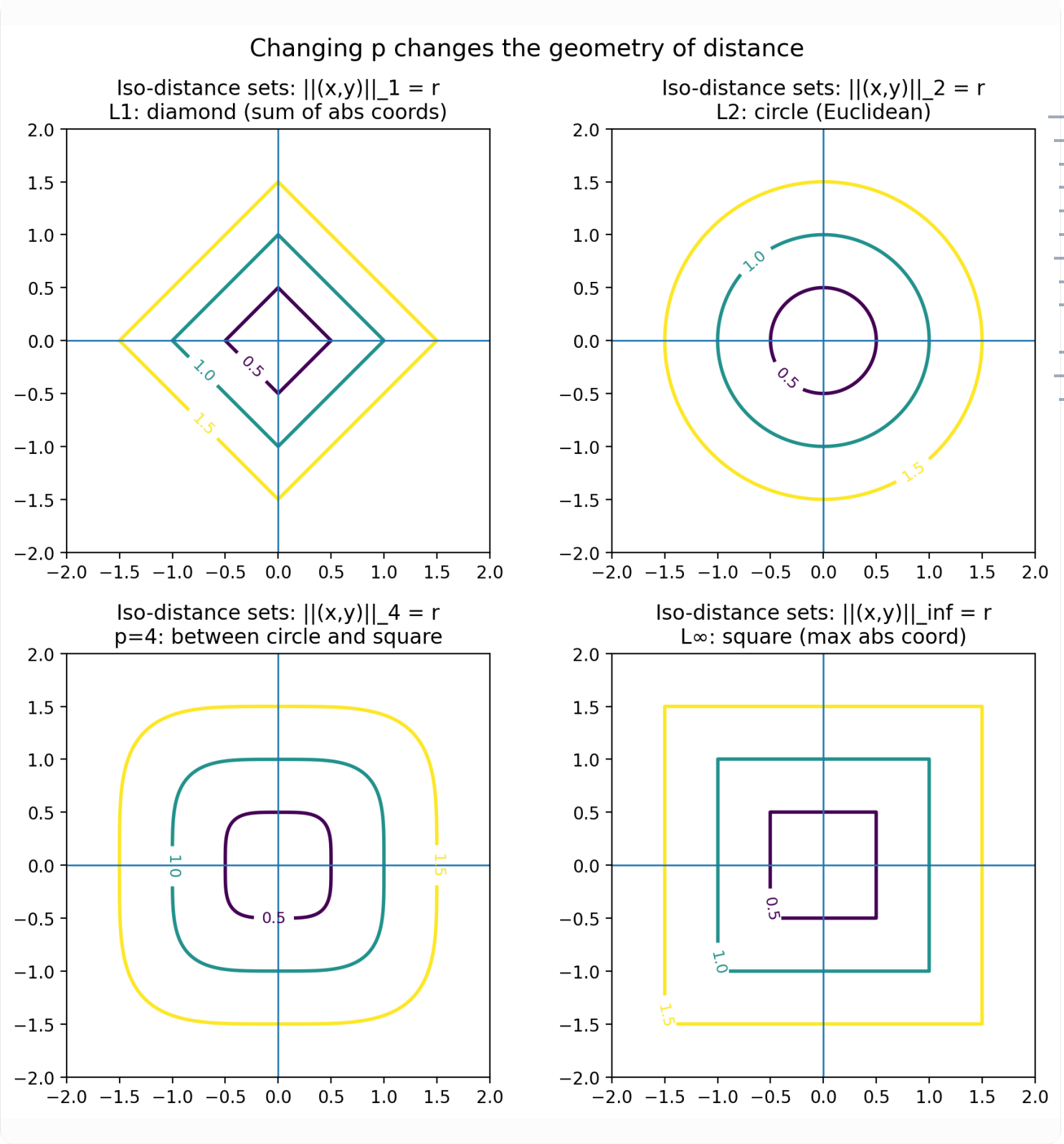
We will plot **iso-distance contours** (level sets) for several p values:

$$\{(x_1, x_2) : \|(x_1, x_2)\|_p = r\}.$$

Key idea: changing p changes what "equally far" means.

- $p = 2$ (**L2 / Euclidean**) → round circles
- $p = 1$ (**L1**) → diamond-shaped contours
- $p \to \infty$ (L_∞) → square-shaped contours (distance is dominated by the largest coordinate)

This matters for the rest of the notebook because "closeness" between data points (and later, model penalties) depends on the choice of norm.



Exercise prompt:

"Which norm would treat outliers along one feature axis as most influential? (hint: L_∞)"

"Which norm would make axis-aligned differences count more than diagonal ones? (hint: L_1)"

Key takeaway: The value of a norm depends on how it weights large vs small components. As p increases, the largest component dominates the norm. This is why different norms lead to different notions of distance and different regularization behavior in machine learning.

A norm measures size, a p-norm is a specific way to measure size, and regularization works by penalizing the norm of the model parameters—so choosing p directly determines what kinds of solutions the model prefers.

Vector addition, scalar multiplication, euclidean distance, and transpose and indexing of matrices

```
v + w: [1 3 3 5]
```

```
[0 2 0 2]
```

```
Numpy Distance: 2.23606797749979
```

```
Manual Distance: 2.23606797749979
Match: True
```

Matrices

```
[[1. 3. 5. 7.]
 [2. 4. 6. 8.]]

Vertical Stack:
 [[1. 3. 5. 7.]
 [2. 4. 6. 8.]]
Horizontal Stack:
 [1. 3. 5. 7. 2. 4. 6. 8.]
Shape of X_1: (2, 4)
```

```
[[1. 3. 5. 7.]
 [2. 4. 6. 8.]]
```

```
1.0
3.0
```

```
Shape of V: (2, 3)

V^{T}:
 [[1 4]
 [2 5]
 [3 6]]

V^{T}:
 [[1 4]
 [2 5]
 [3 6]]

(3, 2)
```

```
[[np.int64(1), np.int64(4)], [np.int64(2), np.int64(5)], [np.int64(3), np.int64(6)]]
```

Matrix Multiplication

Note: Order matters! The inside dimensions of the multiplication must match.

i.e. We can multiply a 2x3 matrix with a 3x4, but we cannot multiply a 3x4 matrix with a 2x3 matrix.

A valid matrix multiplication:

$$A_{2x3}B_{3x4}$$

A not valid matrix mutlilication:

$$B_{3x4}A_{2x3}$$

```
v1 dot V: [ 9 12 15]
V dot v2: [14 32]
```

```
U Matrix
 [[ 5  6]
 [ 2  7]
 [10  3]]

U dot V
 [[29 40 51]
 [30 39 48]
 [22 35 48]]
```

```
V dot U
 [[39 29]
 [90 77]]
```

Frobenious

This norm is applied to an mxn matrix. The Frobenious norm is defined as:

$$||A||_F = (\sum_{i=1}^{n}\sum_{j=1}^{m} A_{i,j}^2)^{\frac{1}{2}}$$

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

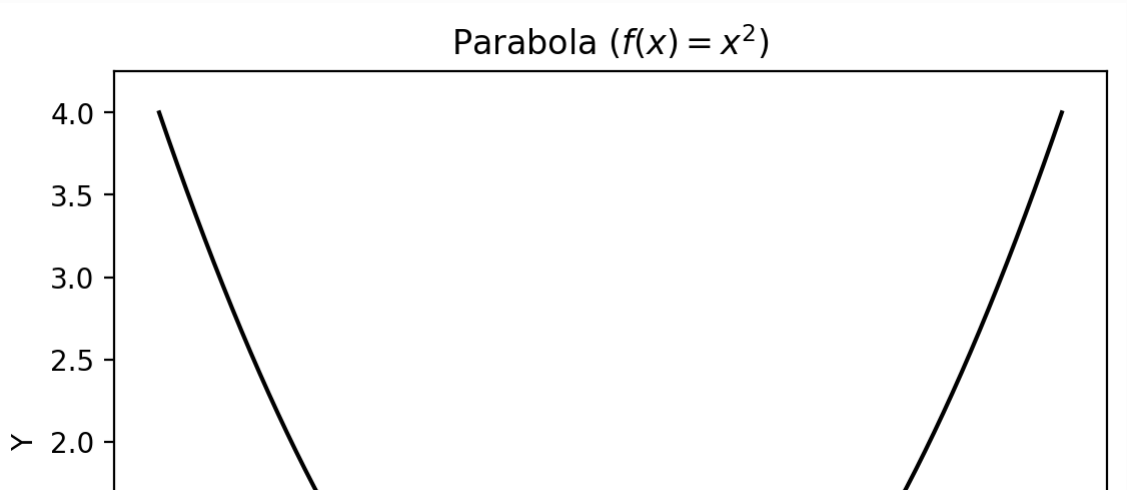
```
Frobenius Norm: 1.4142135623730951
```

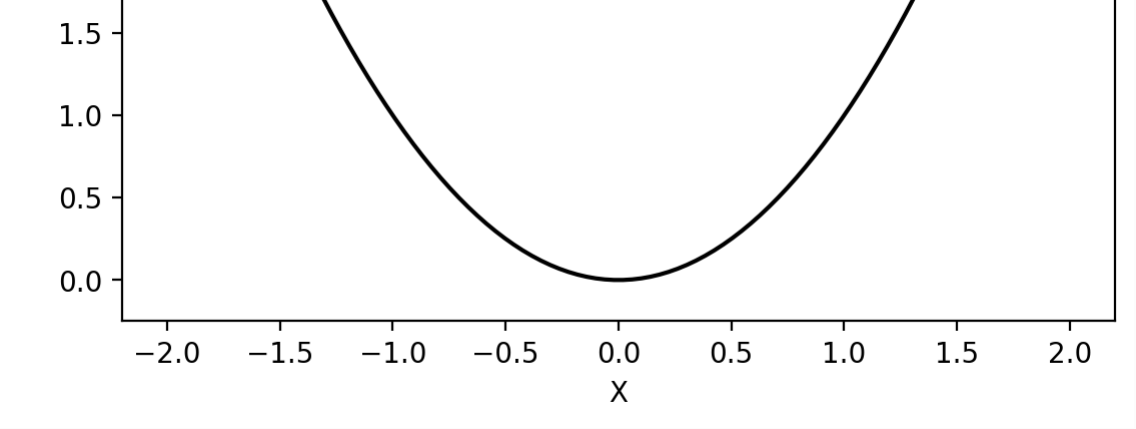
We expect this value to be $(1^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2)^{\frac{1}{2}} = (2)^{\frac{1}{2}} = \quad 2$. Does our answer match?

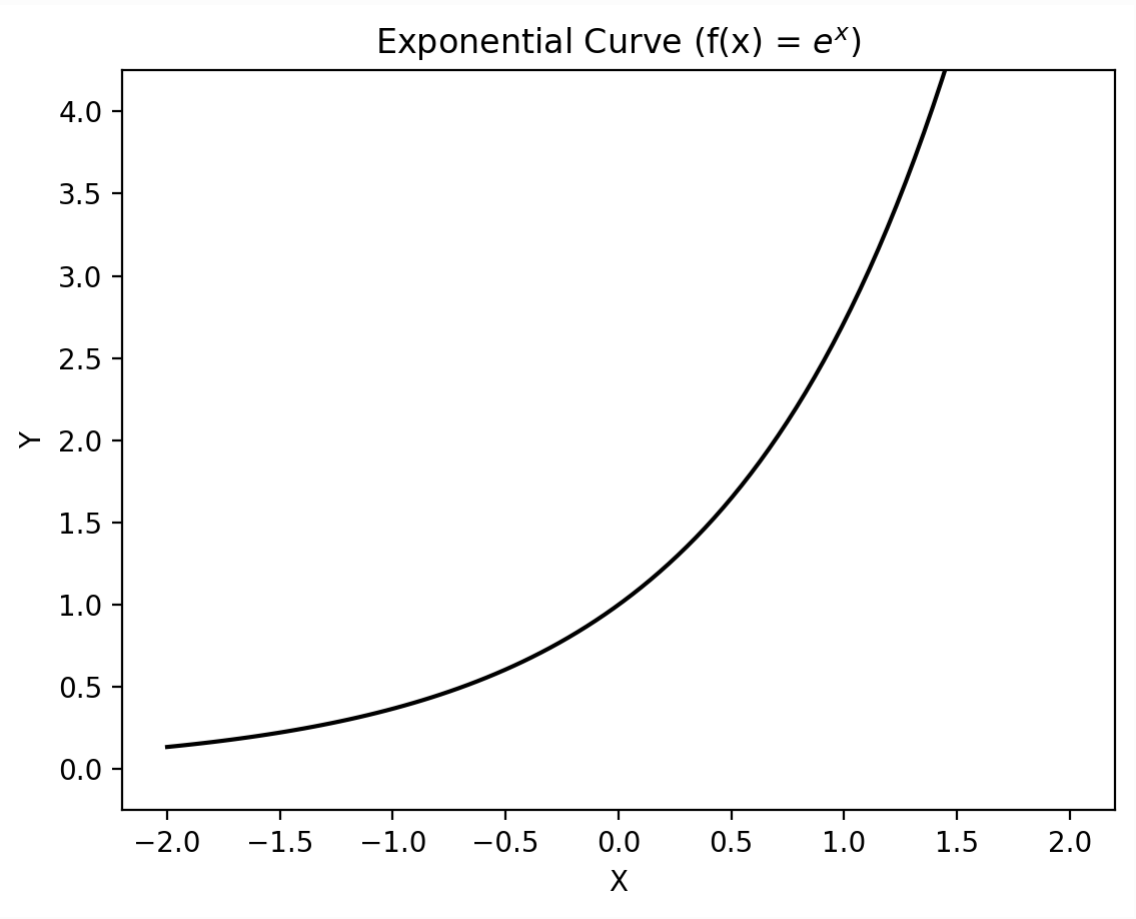
```
Calculated: 1.4142135623730951
Expected: 1.4142135623730951
Match: True
```

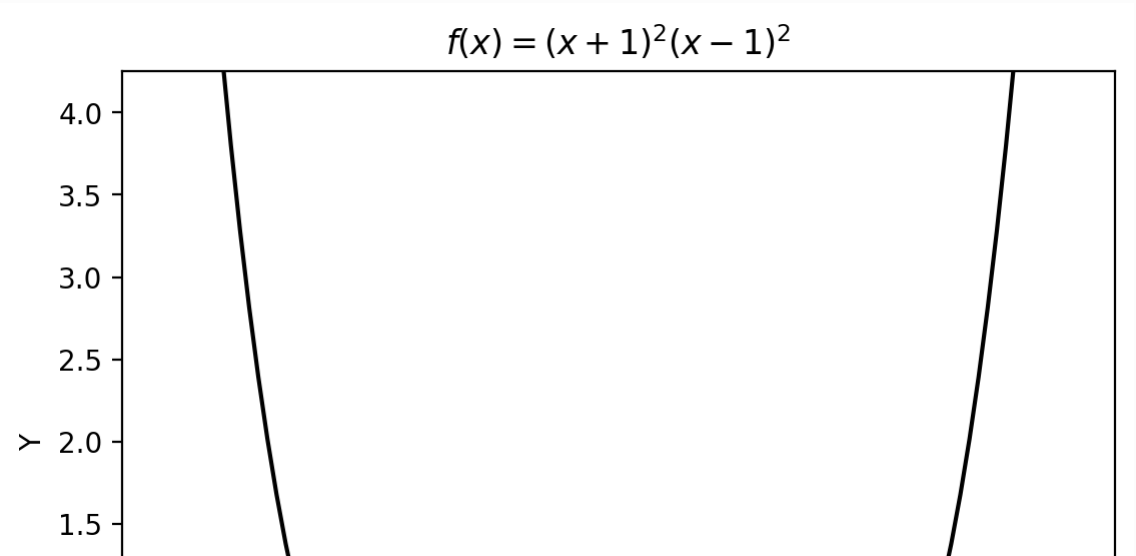
1.2.2 Plotting Functions

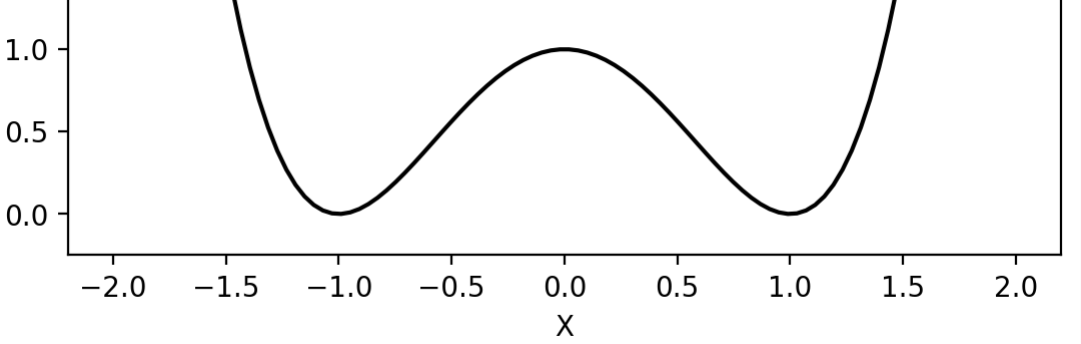
For now, we will just look at how to plot different functions using the matplotlib package.











Sympy

Another package we can use is Sympy for symbolic python. The documentation is found here: https://docs.sympy.org/latest/index.html. The documetation on using matrices with sympy can be found here: https://docs.sympy.org/latest/tutorials/intro-tutorial/matrices.html.

$$\begin{matrix} 1 & 0 & -2 & -1 \\ 2 & 1 & 0 & -1 \\ -1 & 3 & -1 & -2 \end{matrix}$$

```
[ 2 Items
  0: ⎡1  0  0  -1/5⎤
     ⎢0  1  0  -3/5⎥
     ⎣0  0  1   2/5⎦
  1: [ 3 Items
       0: 0
       1: 1
       2: 2
     ]
]
```