Goal: classify bike lanes/roads based on gynocrope data to come up with a sense on how safe they care.

Labels!!

Classes:

1. Bump
2. Pothole
3. …

First case: Geo-data +3D:

Of a certain places is there any issue? 🡺 indicate map in a city 🡺 potential out put A to B 🡺 safety grade

Classification routes

Second case: 3D

Is there any issues 🡺 classification 🡺 x% of accuracy that a sample in a specific kind of issue.

Steps to follow:

1. Append data set and get the mean 🡺 df.app(type).mean()
2. Our research point is safety
3. Add a column to define the data for example CSV for bum give it B, CSV for pothole is P.
4. Distribute tasks between group members
5. Define the key factor for example in our case is the height 🡺 Z ax 🡺 up and down factor
6. + value is up to the level of the street
7. – value is down the level of the street

* **What is Fourier Transforms With scipy.fft: Python Signal Processing**

The Fourier transform is a powerful tool for analyzing signals and is used in everything from audio processing to image compression. SciPy provides a mature implementation in its scipy.fft module, and in this tutorial, you’ll learn how to use it.

* **Install SciPy and Matplotlib**

Before you can get started, you’ll need to install SciPy and [Matplotlib](https://realpython.com/python-matplotlib-guide/). You can do this one of two ways:

1. **Install with Anaconda:** Download and install the [Anaconda Individual Edition](https://www.anaconda.com/products/individual). It comes with SciPy and Matplotlib, so once you follow the steps in the installer, you’re done!
2. **Install with pip:** If you already have [pip](https://realpython.com/what-is-pip/) installed, then you can install the libraries with the following command:

$ python -m pip install -U scipy matplotlib

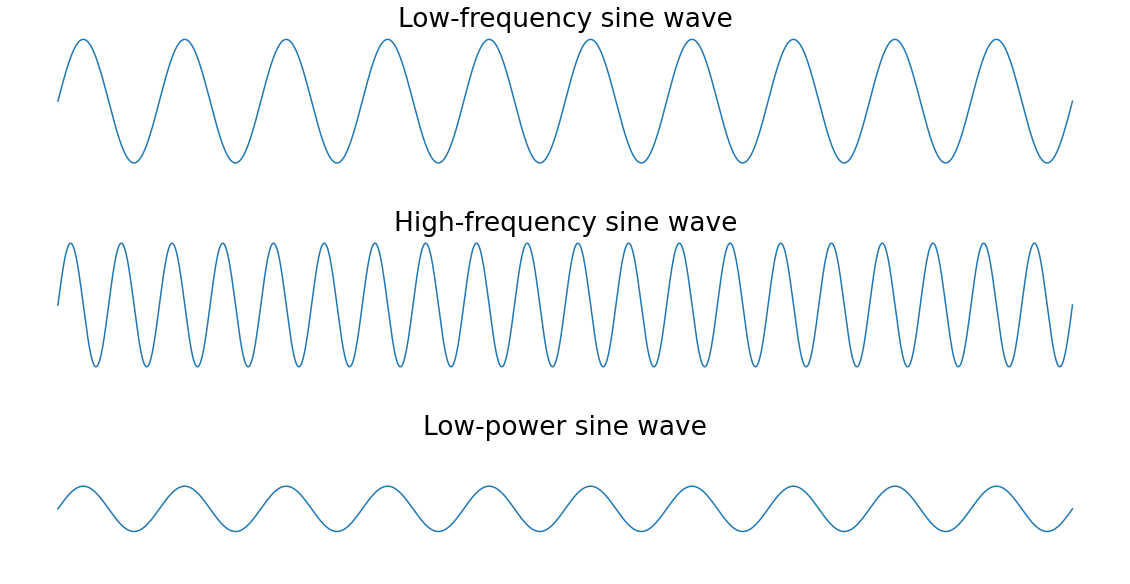
**The Fourier Transform**

[Fourier analysis](https://en.wikipedia.org/wiki/Fourier_analysis) is a field that studies how a **mathematical function** can be decomposed into a series of simpler **trigonometric functions**. The Fourier transform is a tool from this field for decomposing a function into its component frequencies.

Okay, that definition is pretty dense. For the purposes of this tutorial, the Fourier transform is a tool that allows you to take a signal and see the power of each frequency in it. Take a look at the important terms in that sentence:

* A **signal** is information that changes over time. For example, audio, video, and voltage traces are all examples of signals.
* A **frequency** is the speed at which something repeats. For example, clocks tick at a frequency of one hertz (Hz), or one repetition per second.
* **Power**, in this case, just means the strength of each frequency.

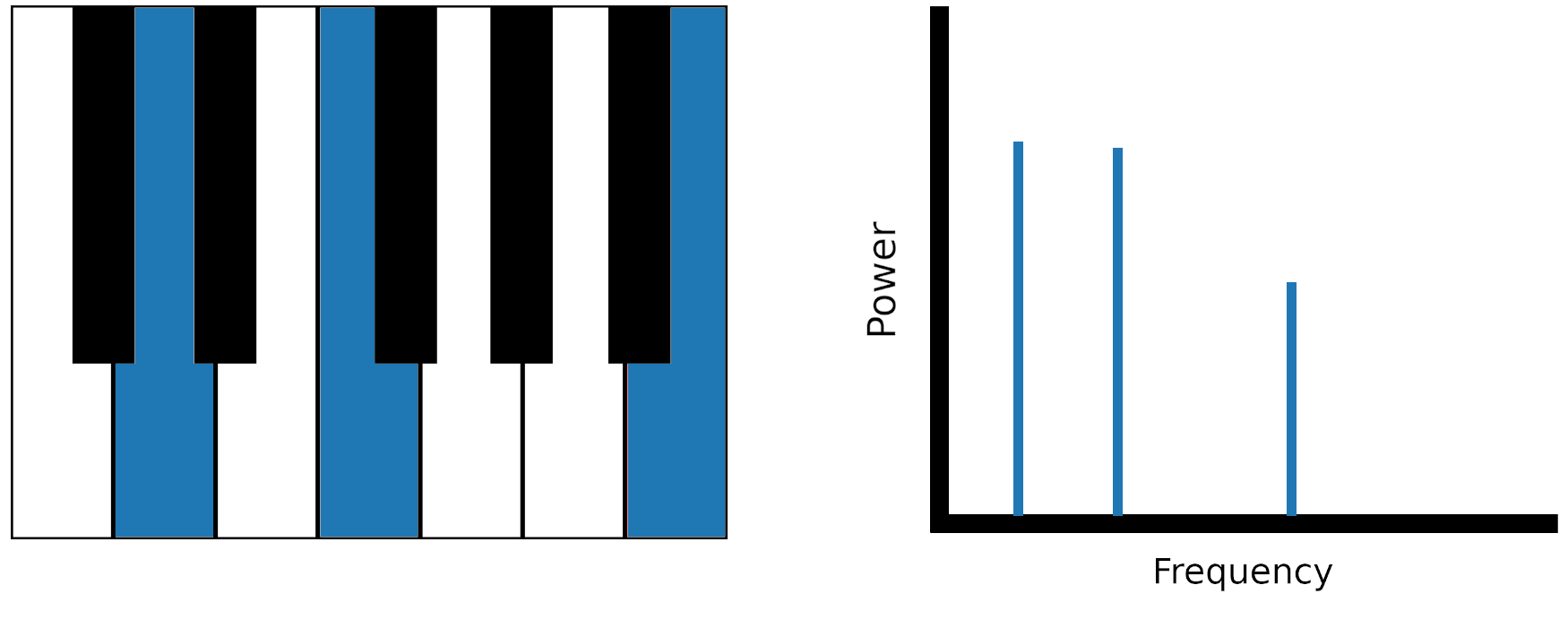
The following image is a visual demonstration of frequency and power on some [sine waves](https://en.wikipedia.org/wiki/Sine_wave):



1. The peaks of the **high-frequency** sine wave are closer together than those of the **low-frequency** sine wave since they repeat more frequently. The **low-power** sine wave has smaller peaks than the other two sine waves.

To make this more concrete, imagine you used the Fourier transform on a recording of someone playing three notes on the piano at the same time. The resulting **frequency spectrum** would show three peaks, one for each of the notes. If the person played one note more softly than the others, then the power of that note’s frequency would be lower than the other two.

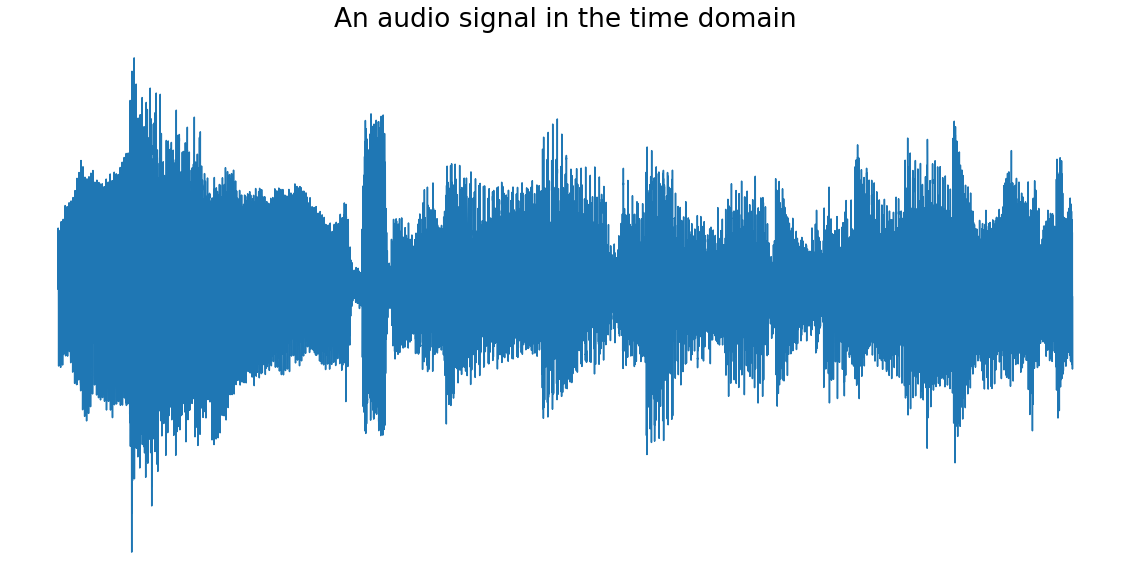
Here’s what that piano example would look like visually:



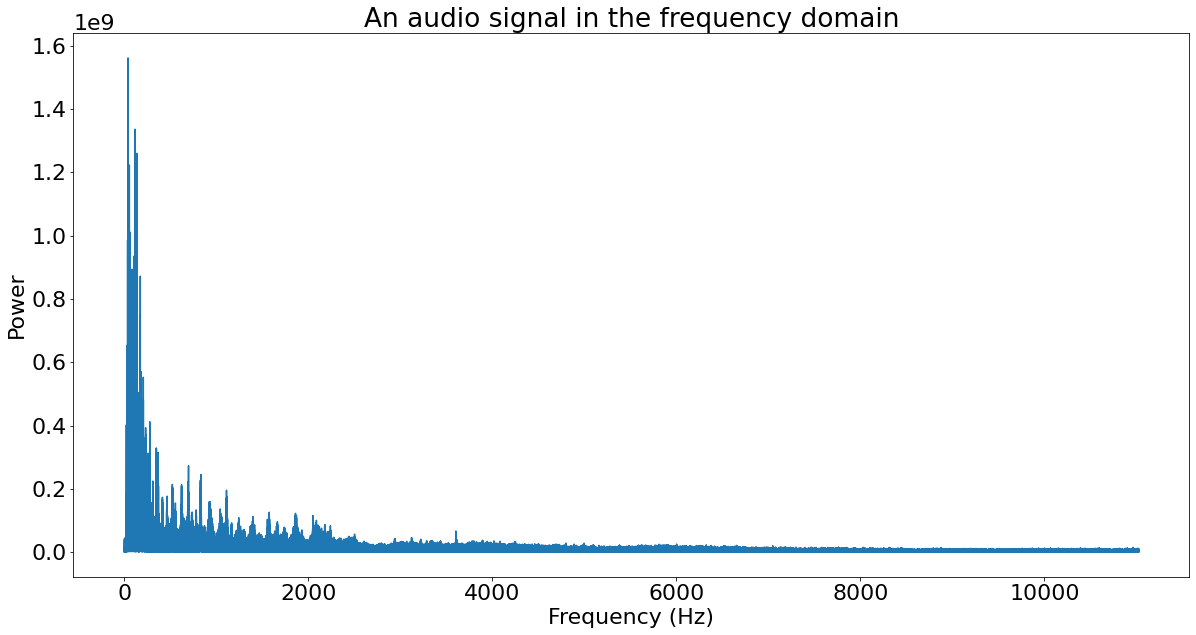
### **Time Domain vs Frequency Domain**

These two terms refer to two different ways of looking at a signal

**Time domain:** information that varies over time, a signal is a wave that varies in amplitude (y-axis) over time (x-axis). The horizontal axis represents time, and the vertical axis represents the amplitude.



**Frequency domain:** its component frequencies. In the frequency domain, a signal is represented as a series of frequencies (x-axis) that each have an associated power (y-axis). The following image is the above audio signal after being Fourier transformed:



## Practical Example: Remove Unwanted Noise From Audio

The steps that you needs to follow to deal with data

1. Create a signal
2. Mixing signals: adding signal together and normalize the results:

Code example:

Adding:

\_, nice\_tone = generate\_sine\_wave(400, SAMPLE\_RATE, DURATION)

\_, noise\_tone = generate\_sine\_wave(4000, SAMPLE\_RATE, DURATION)

noise\_tone = noise\_tone \* 0.3

mixed\_tone = nice\_tone + noise\_tone

Normalize:

normalized\_tone = np.int16((mixed\_tone / mixed\_tone.max()) \* 32767)

plt.plot(normalized\_tone[:1000])

plt.show()

Further normalization:

To listen to the audio, you need to store it in a format that an audio player can read. The easiest way to do that is to use SciPy’s [wavfile.write](https://docs.scipy.org/doc/scipy/reference/io.html) method to store it in a WAV file. 16-bit integers are a standard data type for WAV files, so you’ll normalize your signal to 16-bit integers:

from scipy.io.wavfile import write

# Remember SAMPLE\_RATE = 44100 Hz is our playback rate

write("mysinewave.wav", SAMPLE\_RATE, normalized\_tone)