# Pascual,_Ken_Leonard_Activity_1_NumPy_and_Pandas_7

August 7, 2025

| Technological Institute of the Philippines | Quezon City - Computer Engineering |
|---|---|
| Course Code: | CPE 018 |
| Code Title: | Emerging Technologies in CpE 1 - Fundamentals of Computer Vision |
| 1st Semester | AY 2025-2026 |
| **ACTIVITY NO. 1** | **NumPy and Pandas (Tutorial)** |
| **Name** | Pascual, Ken Leonard |
| **Section** | CPE31S3 |
| **Date Performed**: 6 August 2025 | |
| **Date Submitted**: 6 August 2025 | |
| **Instructor**: | Engr. Verlyn V. Nojor / Engr. Roman M. Richard |

## 0.1 1. Objectives

This activity aims to familiarize students with popular data processing tools, such as NumPy and Pandas, and apply them on sample data.

## 0.2 2. Intended Learning Outcomes

After this activity, the student should be able to:

- Utilize pandas and numpy in fundamental data analysis.
- Demonstrate data analysis operations using numpy and pandas.

## 0.3 3. Procedures and Outputs

### 0.3.1 3.1 Introduction

In this activity, a look at two imporatl tools will be done before jumping into using opencv and more advanced applications. We are looking at NumPy and Pandas!

From Pandas' website:

> *Pandas is a very popular library for working with data (its goal is to be the most powerful and flexible open-source tool, and in our opinion, it has reached that goal). DataFrames are at the center of pandas. A DataFrame is structured like a table or spreadsheet. The rows and the columns both have indexes, and you can perform operations on rows or columns separately.*

*A pandas DataFrame can be easily changed and manipulated. Pandas has helpful functions for handling missing data, performing operations on columns and rows, and transforming data. If that wasn't enough, a lot of SQL functions have counterparts in pandas, such as join, merge, filter by, and group by. With all of these powerful tools, it should come as no surprise that pandas is very popular among data scientists.*

From NumPy's website: > *NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data. There are a few functions that exist in NumPy that we use on pandas DataFrames. For us, the most important part about NumPy is that pandas is built on top of it. So, NumPy is a dependency of Pandas.*

*Disclaimer: Parts of this activity are sourced from https://endaq.com/ and other open source tutorials for data processing tools.*

### 3.1.1 Recap of Python Introduction

1. Python is popular for good reason
2. There are many ways to interact with Python

- Here we are in Google Colab based on Jupyter Notebooks

3. There are many open source libraries to use

- Today we are covering two of the most popular:
  - Numpy
  - Pandas

**3.1.2 SO MANY Other Resources** Python is incredibly popular and so there are a lot of resources you can tap into online. You can either:

- Just start coding and google specific questions when you get stuck (my preference for learning)
- Follow a few online tutorials (like this first)
- Do both!

**Here are a few resources:** * Jake VanderPlas Python Data Science Handbook * Buy the book for $35 * Go through a series of well documented Colab Notebooks * **Highly recommended resource!** * Code Academy: Analyze Data with Python * Udemy Academy: Data Analysis with Pandas & NumPy in Python for Beginner * Datacamp: Introduction to Python

### 0.3.2 3.2 NumPy

NumPy provides an in depth overview of what exactly NumPy is. Quoting them: >*At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.*

Let's get started by importing it, typically shortened to `np` because of how frequently it's called. This will come standard in most Python distributions such as Anaconda. If you need to install it simply: ~~~ !pip install numpy ~~~

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

### 3.2.1 Creating Arrays

**3.2.2 Manually or from Lists** Manually create a list and demonstrate that operations on that list are difficult.

[2]:
```
lst = [0, 1, 2, 69]
lst * 2 #copy all elements in 1st, then include it in the list
```

[2]:
```
[0, 1, 2, 69, 0, 1, 2, 69]
```

That isn't what we expected! For lists, we need to loop through all elements to apply a function to them which can be incredibly time consuming.

[3]:
```
[i * 5 for i in lst] # [0*5, 1*5, 2*5, 69*5]
```

[3]:
```
[0, 5, 10, 345]
```

Now lets make a numpy array and once in an array, let's show how intuitive operations are now that they are performed element by element.

[4]:
```
array = np.array(lst)
print(array)
print(array * 2) # [0*5, 1*5, 2*5, 69*5]
print(array + 2) # [0+5, 1+5, 2+5, 69+5]
```

```
[ 0  1  2 69]
[  0   2   4 138]
[ 2  3  4 71]
```

Let's create a 2D matrix of 32 bit floats.

[5]:
```
np.array([[2, 0, 0],
          [0, 1, 0],
          [0, 0, 1]], dtype=np.float32)
```

[5]:
```
array([[2., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]], dtype=float32)
```

**3.2.3 Using Functions** We'll go through a few here, but for more in depth examples and options see NumPy's Array Creation Routines. These first few examples are for very basic arrays/matrices.

[6]:
```
np.zeros(10   )
```

[6]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[7]:
```
np.zeros([4 ,  4]) # 4x4 matrix with just 0s
```

[7]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[8]:
```
np.eye(5) #identity matrix, 5x5
```

[8]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Now let's start making sequences.

[9]:
```
np.arange(11) # array containing numbers from 0 to 10
```

[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[10]:
```
np.arange(0,  #start
          10) #stop (not included in array)
```

[10]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[11]:
```
np.arange(0,  #start
          10, #stop (not included in array)
          2)  #step size
```

[11]:
```
array([0, 2, 4, 6, 8])
```

[12]:
```
np.linspace(0,  #start
            11, #stop (default to be included, can pass in endpoint=False)
            5)  #number of data points evenly spaced
```

[12]:
```
array([ 0.  ,  2.75,  5.5 ,  8.25, 11.  ])
```

[13]:
```
np.logspace(0, #output starts being raised to this value
            4, #ending raised value
            4) #number of data points
```

[13]:
```
array([1.00000000e+00, 2.15443469e+01, 4.64158883e+02, 1.00000000e+04])
```

Logspace is the equivalent of rasing a base by a linspaced array.

[14]:
```
10 ** np.linspace(0,3,4)
```

[14]:
```
array([   1.,   10.,  100., 1000.])
```

[15]:
```
np.logspace(0, 4, 5, base=2)
```

[15]:
```
array([ 1.,  2.,  4.,  8., 16.])
```

If you don't want to have to do the mental math to know what exponent to raise the values to, you can use geomspace but this only helps for base of 10.

[16]:
```
np.geomspace(1, 1000, 4)
```

[16]:
```
array([   1.,   10.,  100., 1000.])
```

Random numbers!

[17]:
```
np.random.rand(5)
```

[17]:
```
array([0.79915092, 0.274912  , 0.68050321, 0.1356096 , 0.45424754])
```

[18]:
```
np.random.rand(2, 3)
```

[18]:
```
array([[0.87813013, 0.68060803, 0.13814815],
       [0.50049146, 0.65599787, 0.47401145]])
```

**3.2.4 Indexing** Let's first create a simple array.

[19]:
```
array = np.arange(1, 11, 1)
array
```

[19]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Now index the first item.

[20]:
```
array[0]
```

[20]:
```
np.int64(1)
```

Index the last item.

[21]:
```
array[-1]
```

[21]:
```
np.int64(10)
```

Grab every 2nd item.

[22]:
```
array[::2]
```

[22]:
```
array([1, 3, 5, 7, 9])
```

Grab every second item starting at index of 1 (the second value)

[23]:
```
array[1::2]
```

[23]:
```
array([ 2,  4,  6,  8, 10])
```

Start from the second item, going to the 7th but skipping every other. (Our first time using the full `array[start (inclusive): stop (exclusive): step]` array 'slicing' notation)

[24]:
```
array[1:6:2]
```

[24]:
```
array([2, 4, 6])
```

Reverse the order.

[25]:
```
array[::-1]
```

[25]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

Boolean operations to index the array.

[26]:
```
array[array < 5]
```

[26]:
```
array([1, 2, 3, 4])
```

[27]:
```
array[array * 2 < 5]
```

[27]:
```
array([1, 2])
```

Integer list can also index arrays.

[28]:
```
array[[1,3,5]]
```

[28]:
```
array([2, 4, 6])
```

Now let's create a slightly more complicated, 2 dimensional array.

[29]:
```
array_2d = np.arange(10).reshape((2, 5))
array_2d
```

[29]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

Indexing the second dimension.

[30]:
```
array_2d[:, 1]
```

[30]:
```
array([1, 6])
```

Any combination of these indexing methods works as well.

[31]:
```
array_2d[[True, False], 1::2]
```

[31]:
```
array([[1, 3]])
```

[32]:
```
array_2d[1, [0,1,4]]
```

[32]:
```
array([5, 6, 9])
```

### 3.2.5 Operations

[33]:
```
array
```

[33]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[34]:
```
array + 100
```

[34]:
```
array([101, 102, 103, 104, 105, 106, 107, 108, 109, 110])
```

[35]:
```
array * 2
```

[35]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

To raise all the elements in an array to an exponent we have to use the notation ** not ^.

[36]:
```
array ** 2
```

[36]:
```
array([  1,   4,   9,  16,  25,  36,  49,  64,  81, 100])
```

Use that shape to create a new array matching it to do operations with.

[37]:
```
array2 = np.arange(array.shape[0]) * 5
array2
```

[37]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])
```

[38]:
```
array2 + array
```

[38]:
```
array([ 1,  7, 13, 19, 25, 31, 37, 43, 49, 55])
```

### 3.2.6 Stats of an Array

[39]:
```
print(array)
print(array_2d)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

[40]:
```
print(array.shape)
print(array_2d.shape)
```

```
(10,)
(2, 5)
```

[41]:
```
print(len(array))
print(len(array_2d))
```

```
10
2
```

[42]:
```
print(array.max())
print(array_2d.max())
```

```
10
9
```

[43]:
```
print(array.min())
print(array_2d.min())
```

```
1
0
```

[44]:
```
array.std()
```

[44]:
```
np.float64(2.8722813232690143)
```

[45]:
```
array.cumsum()
```

[45]:
```
array([ 1,  3,  6, 10, 15, 21, 28, 36, 45, 55])
```

[46]:
```
array.cumprod()
```

[46]:
```
array([      1,       2,       6,      24,     120,     720,    5040,
         40320,  362880, 3628800])
```

### 3.2.7 Constants

[47]:
```
np.pi
```

[47]:
```
3.141592653589793
```

[48]:
```
np.e
```

[48]:
```
2.718281828459045
```

[49]:
```
np.inf
```

[49]:
```
inf
```

### 3.2.8 Functions

[50]:
```
np.sin(np.pi / 2)
```

[50]:
```
np.float64(1.0)
```

[51]:
```
np.cos(np.pi)
```

[51]:
```
np.float64(-1.0)
```

[52]:
```
np.log(np.e)
```

[52]:
```
np.float64(1.0)
```

[53]:
```
np.log10(100)
```

[53]:
```
np.float64(2.0)
```

[54]:
```
np.log2(64)
```

[54]:
```
np.float64(6.0)
```

To demonstrate rounding, let's first make a new array with decimals.

[55]:
```
array = np.arange(4) / 3
print(array)
np.around(array, 2)
```

```
[0.         0.33333333 0.66666667 1.        ]
```

[55]:
```
array([0.  , 0.33, 0.67, 1.  ])
```

**3.2.9 Looping vs Vectorization** As mentioned in the beginning, NumPy uses machine code with their ndarray objects which is what leads to the performance improvements. Let's demonstrate this by constructing a simple sine wave.

[56]:
```
fs = 500 #sampling rate in Hz
d_t = 1 / fs #time steps in seconds
n_step = 1000 #number of steps (there will be n_step+1 data points)

amp = 1 #amplitude of sine wave
f = 2 #frequency of sine wave

time = np.linspace(0,              #start time
                   n_step * d_t,     #end time
                   num=n_step + 1)  #number of data points, one more than the␣
 ↪steps to include an endpoint
time.shape
```

[56]:
```
(1001,)
```

First we make a function to loop through each element and calculate the amplitude.

[57]:
```
def sine_wave_with_loop(time, amp, f, phase=0):
  length = time.shape[0]
  wave = np.zeros(length)

  for i in range(length-1):
    wave[i] = np.sin(2 * np.pi * f * time[i] + phase * np.pi / 180)*amp
  return wave
```

Now let's time how quickly that executes for our time array of 1,001 data points.

[58]:
```
%timeit sine_wave_with_loop(time, amp, f)
```

```
2.51 ms ± 717 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Now let's do the same using NumPy's sine function and vectorization.

[59]:
```
def sine_wave_with_numpy(time, amp, f, phase=0):
  """Takes in a time array and sine wave parameters, returns an array of the␣
 ↪sine wave amplitude."""
  return np.sin(2 * np.pi * f * time + phase * np.pi / 180) * amp
```

Notice my docstrings!

[60]:
```
help(sine_wave_with_numpy)
```

```
Help on function sine_wave_with_numpy in module __main__:

sine_wave_with_numpy(time, amp, f, phase=0)
    Takes in a time array and sine wave parameters, returns an array of the sine
wave amplitude.
```

[61]:
```
%timeit sine_wave_with_numpy(time, amp, f)
```

```
35.7 µs ± 18.8 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Using vectorization is about **100x faster!** And this increases the longer the loops are.

**3.2.10 Why Vectorization Works So Much Faster** The above example highlights that NumPy is much faster, but why? Because it is using compiled machine code under the hood for it's operations.

Python has the Numba package which can be used to do this compilation which we will do to highlight just why NumPy is faster (and recommended!).

[62]:
```
from numba import njit

numba_sine_wave_with_loop = njit(sine_wave_with_loop)
numba_sine_wave_with_numpy = njit(sine_wave_with_numpy)
```

[63]:
```
%timeit numba_sine_wave_with_loop(time, amp, f)
%timeit numba_sine_wave_with_numpy(time, amp, f)
```

```
38.6 µs ± 20.8 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
33.9 µs ± 15.8 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Let's combine this into a DataFrame we'll discuss next in more detail, but here's a preview.

[64]:
```
import pandas as pd

time_data = pd.DataFrame({'Time (us)':[2.77*100, 27.5, 23.1, 22.6],
                          'Method':['Loop','NumPy','Loop','NumPy'],
                          'Numba?':['w/o','w/o','w','w']}
)
time_data
```

[64]:
```
   Time (us) Method Numba?
0      277.0   Loop    w/o
1       27.5  NumPy    w/o
2       23.1   Loop      w
3       22.6  NumPy      w
```

Now let's plot it and preview Plotly!

[65]:
```
!pip install --upgrade -q plotly
import plotly.express as px

fig = px.bar(time_data,
             x="Method",
             y="Time (us)",
             color="Numba?",
             title="Compute Time of a Sine Wave for 1000 Elements",
             barmode='group')
fig.show()
```

```
                         9.6/9.6 MB
31.1 MB/s eta 0:00:00
```

### 0.3.3 3.3 Pandas

Pandas is built *on top of* NumPy meaning that the data is stored still as NumPy ndarray objects under the hood. But it exposes a much more intuitive labeling/indexing architecture and allows you to link arrays of different types (strings, floats, integers etc.) to one another.

To quote Jake VanderPlas: >*At the very basic level, Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices.*

To start, import pandas as `pd`, again this will come standard in virtually all Python distributions such as Anaconda. But to install is simply: ~~~ !pip install pandas ~~~

[66]:
```
import pandas as pd
```

There are three types of Pandas objects, we'll only focus on the first two: 1. **Series** – 1D labeled homogeneous array, sizeimmutable 2. **Data Frames** – 2D labeled, size-mutable tabular structure with heterogenic columns 3. **Panel** – 3D labeled size mutable array.

#### 3.3.1 Creating a Series

First let's create a few numpy arrays.

[67]:
```
amplitude = sine_wave_with_numpy(time, amp, f, 180)
print(time)
print(amplitude)
```

```
[0.    0.002 0.004 … 1.996 1.998 2.   ]
[ 1.22464680e-16 -2.51300954e-02 -5.02443182e-02 …  5.02443182e-02
  2.51300954e-02  1.10218212e-15]
```

Now let's see what a series looks like made from one of the arrays.

[68]:
```
pd.Series(amplitude)
```

[68]:
```
0       1.224647e-16
1      -2.513010e-02
2      -5.024432e-02
3      -7.532681e-02
4      -1.003617e-01
              …
996     1.003617e-01
997     7.532681e-02
998     5.024432e-02
999     2.513010e-02
1000    1.102182e-15
Length: 1001, dtype: float64
```

This type of series has some value, but you really start to see it when you add in an index.

[69]:
```
series = pd.Series(data=amplitude,
                   index=time,
                   name='Amplitude')
series
```

[69]:
```
0.000    1.224647e-16
0.002   -2.513010e-02
0.004   -5.024432e-02
0.006   -7.532681e-02
0.008   -1.003617e-01
               …
1.992    1.003617e-01
1.994    7.532681e-02
1.996    5.024432e-02
1.998    2.513010e-02
2.000    1.102182e-15
Name: Amplitude, Length: 1001, dtype: float64
```

Here's where Pandas shines - indexing is much more intuitive (and inclusive) to specify based on labels, not those confusing integer locations. We'll come back to this when we have the dataframe next too.

[70]:
```
series[0: 0.01]
```

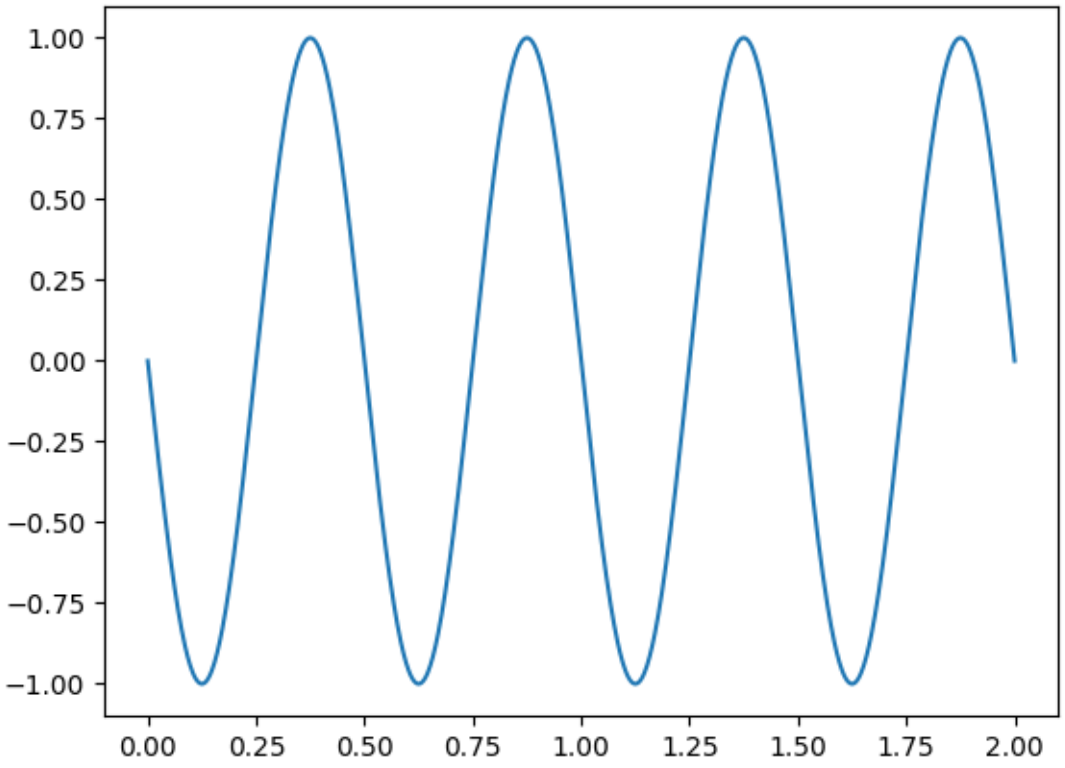
[70]:
```
0.000    1.224647e-16
0.002   -2.513010e-02
0.004   -5.024432e-02
0.006   -7.532681e-02
0.008   -1.003617e-01
0.010   -1.253332e-01
```

```
Name: Amplitude, dtype: float64
```

Being able to plot quickly is also a plus!

[71]:
```
series.plot()
```

[71]:
```
<Axes: >
```




Remember we never left the NumPy array, it is still here and can be accessed with the following.

[72]:
```
series.values
```

[72]:
```
array([ 1.22464680e-16, -2.51300954e-02, -5.02443182e-02, …,
        5.02443182e-02,  2.51300954e-02,  1.10218212e-15])
```

[73]:
```
series.to_numpy()
```

[73]:
```
array([ 1.22464680e-16, -2.51300954e-02, -5.02443182e-02, …,
        5.02443182e-02,  2.51300954e-02,  1.10218212e-15])
```

**3.3.2 Creating a DataFrame** A DataFrame is basically a sequence of aligned series objects, and by aligned I mean they share a common index or label. This let's us mix and match types easily among other benefits.

First we'll start creating dataframes using what is called a "dictionary" with keys and values.

[74]:
```
df = pd.DataFrame({"Phase 0": sine_wave_with_numpy(time, amp, f, 00),
                   "Phase 90": sine_wave_with_numpy(time, amp, f, 90),
                   "Phase 180": sine_wave_with_numpy(time, amp, f, 180),
                   "Phase 270": sine_wave_with_numpy(time, amp, f, 270)},
                  index=time)
df
```

[74]:

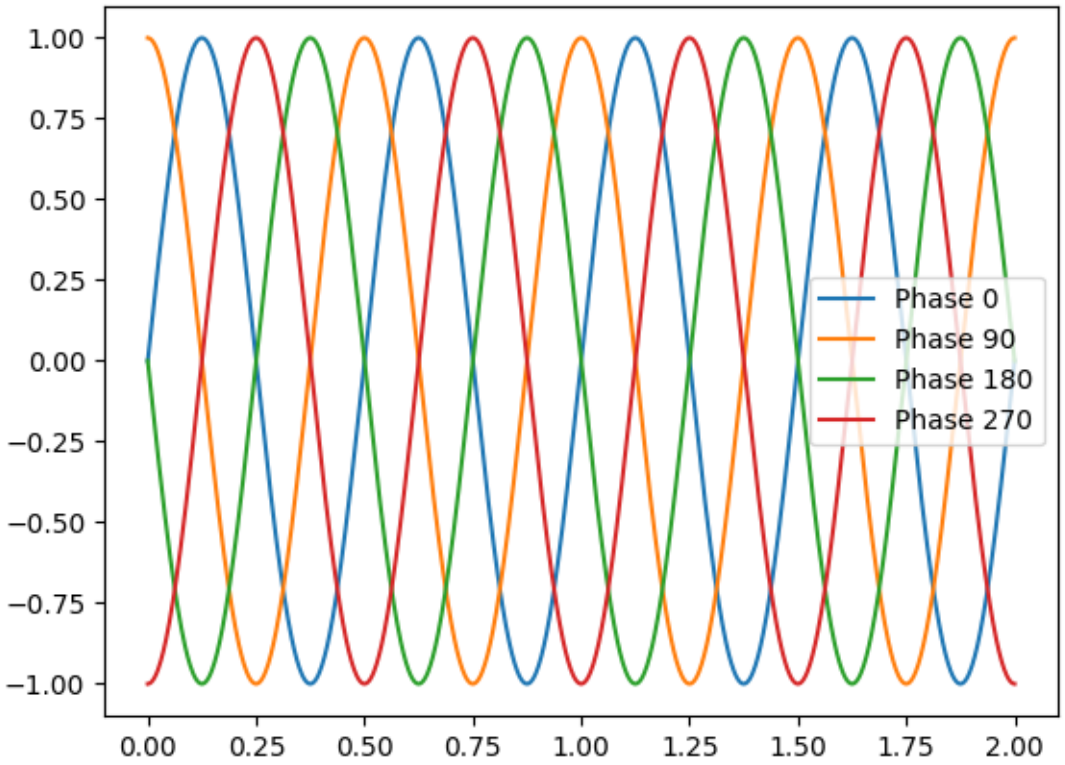
| | Phase 0 | Phase 90 | Phase 180 | Phase 270 |
|---|---|---|---|---|
| 0.000 | 0.000000e+00 | 1.000000 | 1.224647e-16 | -1.000000 |
| 0.002 | 2.513010e-02 | 0.999684 | -2.513010e-02 | -0.999684 |
| 0.004 | 5.024432e-02 | 0.998737 | -5.024432e-02 | -0.998737 |
| 0.006 | 7.532681e-02 | 0.997159 | -7.532681e-02 | -0.997159 |
| 0.008 | 1.003617e-01 | 0.994951 | -1.003617e-01 | -0.994951 |
| … | … | … | … | … |
| 1.992 | -1.003617e-01 | 0.994951 | 1.003617e-01 | -0.994951 |
| 1.994 | -7.532681e-02 | 0.997159 | 7.532681e-02 | -0.997159 |
| 1.996 | -5.024432e-02 | 0.998737 | 5.024432e-02 | -0.998737 |
| 1.998 | -2.513010e-02 | 0.999684 | 2.513010e-02 | -0.999684 |
| 2.000 | -9.797174e-16 | 1.000000 | 1.102182e-15 | -1.000000 |

[1001 rows x 4 columns]

**3.3.3 Plotting (Preview)** Dataframes also wrap around Matplotlib to allow for plotting directly from the dataframe object itself. This can also be done from the Pandas Series object too like we showed earlier.

[75]:
```
df.plot()
```

[75]:
```
<Axes: >
```




[76]:
```
df['Max'] = df.max(axis=1)
df['Min'] = df.min(axis=1)
df
```

[76]:

| | Phase 0 | Phase 90 | Phase 180 | Phase 270 | Max | Min |
|---|---|---|---|---|---|---|
| 0.000 | 0.000000e+00 | 1.000000 | 1.224647e-16 | -1.000000 | 1.000000 | -1.000000 |
| 0.002 | 2.513010e-02 | 0.999684 | -2.513010e-02 | -0.999684 | 0.999684 | -0.999684 |
| 0.004 | 5.024432e-02 | 0.998737 | -5.024432e-02 | -0.998737 | 0.998737 | -0.998737 |
| 0.006 | 7.532681e-02 | 0.997159 | -7.532681e-02 | -0.997159 | 0.997159 | -0.997159 |
| 0.008 | 1.003617e-01 | 0.994951 | -1.003617e-01 | -0.994951 | 0.994951 | -0.994951 |
| … | … | … | … | … | … | … |
| 1.992 | -1.003617e-01 | 0.994951 | 1.003617e-01 | -0.994951 | 0.994951 | -0.994951 |
| 1.994 | -7.532681e-02 | 0.997159 | 7.532681e-02 | -0.997159 | 0.997159 | -0.997159 |
| 1.996 | -5.024432e-02 | 0.998737 | 5.024432e-02 | -0.998737 | 0.998737 | -0.998737 |
| 1.998 | -2.513010e-02 | 0.999684 | 2.513010e-02 | -0.999684 | 0.999684 | -0.999684 |
| 2.000 | -9.797174e-16 | 1.000000 | 1.102182e-15 | -1.000000 | 1.000000 | -1.000000 |

[1001 rows x 6 columns]

This will be the topic of the next webinar, plotting with Plotly!

Note that I need to install an upgraded version of Plotly in Colab because the default Plotly Express version doesn't work in Colab (but their more advanced graph objects does).

[77]:
```
!pip install --upgrade -q plotly
import plotly.express as px
```

[78]:
```
px.line(df).show()
```

**3.3.4 Load from CSV** This dataset was discussed in a blog on vibration metrics and used bearing data as an example.

Note you don't *have* to use a CSV. They have a lot of other file formats natively supported (see full list): * hdf * feather * pickle

But I know everyone likes CSVs!

[79]:
```
df = pd.read_csv('https://info.endaq.com/hubfs/Plots/bearing_data.csv',␣
↪index_col=0)
df
```

[79]:

| Time | Fault_021 | Fault_014 | Fault_007 | Normal |
|---|---|---|---|---|
| 0.000000 | -0.105351 | -0.074395 | 0.053116 | 0.046104 |
| 0.000083 | 0.132888 | 0.056365 | 0.116628 | -0.037134 |
| 0.000167 | -0.056535 | 0.201257 | 0.083654 | -0.089496 |
| 0.000250 | -0.193178 | -0.024528 | -0.026477 | -0.084906 |
| 0.000333 | 0.064879 | -0.072284 | 0.045319 | -0.038594 |
| … | … | … | … | … |
| 9.999667 | 0.095754 | 0.145055 | -0.098923 | 0.064254 |
| 9.999750 | -0.123083 | 0.092263 | -0.067573 | 0.070721 |
| 9.999833 | -0.036508 | -0.168120 | 0.005685 | 0.103265 |
| 9.999917 | 0.097006 | -0.035898 | 0.093400 | 0.124335 |
| 10.000000 | -0.008762 | 0.165846 | 0.130923 | 0.114947 |

[120000 rows x 4 columns]

**3.3.5 Save CSV** Like reading data, there are a host of native formats we can save data from a dataframe. See documentation.

[80]:
```
df.to_csv('bearing-data.csv')
```

**3.3.6 Simple Analysis**

[81]:
```
df.describe()
```

[81]:

| | Fault_021 | Fault_014 | Fault_007 | Normal |
|---|---|---|---|---|
| count | 120000.000000 | 120000.000000 | 120000.000000 | 120000.000000 |
| mean | 0.012251 | 0.002729 | 0.002953 | 0.010755 |
| std | 0.198383 | 0.157761 | 0.121272 | 0.065060 |
| min | -1.037862 | -1.338628 | -0.650390 | -0.269114 |
| 25% | -0.107020 | -0.096649 | -0.072284 | -0.032544 |
| 50% | 0.011682 | 0.001299 | 0.004548 | 0.013351 |

```
75%         0.132054       0.100872       0.080081       0.056535
max         0.917908       1.124376       0.594025       0.251382
```

[82]:
```
df.std()
```

[82]:
```
Fault_021    0.198383
Fault_014    0.157761
Fault_007    0.121272
Normal       0.065060
dtype: float64
```

[83]:
```
df.max()
```

[83]:
```
Fault_021    0.917908
Fault_014    1.124376
Fault_007    0.594025
Normal       0.251382
dtype: float64
```

Note that these built in Pandas functions are using NumPy to process and are the equivalent of doing the following.

[84]:
```
np.max(df)
```

[84]:
```
1.124375968063872
```

[85]:
```
df.quantile(0.25)
```

[85]:
```
Fault_021   -0.107020
Fault_014   -0.096649
Fault_007   -0.072284
Normal      -0.032544
Name: 0.25, dtype: float64
```

[86]:
```
df['abs(max)'] = df.abs().max(axis=1)
df
```

[86]:
```
           Fault_021  Fault_014  Fault_007    Normal  abs(max)
Time
0.000000   -0.105351  -0.074395   0.053116  0.046104  0.105351
0.000083    0.132888   0.056365   0.116628 -0.037134  0.132888
0.000167   -0.056535   0.201257   0.083654 -0.089496  0.201257
0.000250   -0.193178  -0.024528  -0.026477 -0.084906  0.193178
0.000333    0.064879  -0.072284   0.045319 -0.038594  0.072284
…               …          …          …         …         …
9.999667    0.095754   0.145055  -0.098923  0.064254  0.145055
9.999750   -0.123083   0.092263  -0.067573  0.070721  0.123083
9.999833   -0.036508  -0.168120   0.005685  0.103265  0.168120
```

```
9.999917    0.097006  -0.035898   0.093400  0.124335  0.124335
10.000000  -0.008762   0.165846   0.130923  0.114947  0.165846

[120000 rows x 5 columns]
```

**3.3.7 Indexing** Here is where indexing in Python gets a whole lot more intuitive! A dataframe with an index let's use index values (time in this case) to slice the dataframe, not rely on the nth element in the arrays.

[87]:
```
df[0: 0.05]
```

[87]:
```
          Fault_021  Fault_014  Fault_007    Normal  abs(max)
Time
0.000000  -0.105351  -0.074395   0.053116  0.046104  0.105351
0.000083   0.132888   0.056365   0.116628 -0.037134  0.132888
0.000167  -0.056535   0.201257   0.083654 -0.089496  0.201257
0.000250  -0.193178  -0.024528  -0.026477 -0.084906  0.193178
0.000333   0.064879  -0.072284   0.045319 -0.038594  0.072284
…             …          …          …         …         …
0.049584   0.131010   0.129136  -0.014619  0.021487  0.131010
0.049667   0.437675  -0.221399  -0.025340  0.021070  0.437675
0.049750   0.095754  -0.120689   0.033137  0.035256  0.120689
0.049834  -0.137269   0.275977   0.023716  0.044226  0.275977
0.049917   0.150203   0.019167  -0.044670  0.005424  0.150203

[600 rows x 5 columns]
```

We can also use the same convention as before by adding in a step definition, in this case we'll grab every 100th point.

[88]:
```
df[0: 0.05: 100]
```

[88]:
```
          Fault_021  Fault_014  Fault_007    Normal  abs(max)
Time
0.000000  -0.105351  -0.074395   0.053116  0.046104  0.105351
0.008333  -0.481067  -0.137745  -0.010558 -0.012934  0.481067
0.016667  -0.265985  -0.073746  -0.140669  0.027329  0.265985
0.025000  -0.192343   0.203856  -0.168120 -0.029832  0.203856
0.033334   0.018984   0.154476  -0.072933  0.076353  0.154476
0.041667  -0.289975  -0.227409   0.088202  0.016898  0.289975
```

There are ways to use the integer based indexing if you so desire.

[89]:
```
df.iloc[0:10]
```

[89]:
```
          Fault_021  Fault_014  Fault_007    Normal  abs(max)
Time
0.000000  -0.105351  -0.074395   0.053116  0.046104  0.105351
```

```
0.000083   0.132888   0.056365   0.116628 -0.037134  0.132888
0.000167  -0.056535   0.201257   0.083654 -0.089496  0.201257
0.000250  -0.193178  -0.024528  -0.026477 -0.084906  0.193178
0.000333   0.064879  -0.072284   0.045319 -0.038594  0.072284
0.000417   0.214874   0.034761   0.060751  0.025451  0.214874
0.000500  -0.076353   0.094212  -0.174130  0.040680  0.174130
0.000583  -0.065922  -0.070010  -0.229521  0.042558  0.229521
0.000667   0.206529  -0.079431   0.045482  0.038177  0.206529
0.000750   0.021487   0.092426   0.027452  0.044018  0.092426
```

**3.3.8 Rolling** I love the rolling method which allows for easy rolling window calculations, something you'll do frequently with time series data.

[89]:
```
```

[90]:
```
df.rolling(8).max()[::8]
```

[90]:
```
          Fault_021  Fault_014  Fault_007    Normal  abs(max)
Time
0.000000        NaN        NaN        NaN       NaN       NaN
0.000667   0.214874   0.201257   0.116628  0.042558  0.229521
0.001333   0.293313   0.116628   0.152689  0.044018  0.293313
0.002000   0.318764   0.179491   0.077969  0.031292  0.318764
0.002667   0.160634   0.109481   0.175917  0.065714  0.175917
…               …          …          …         …         …
9.996750   0.219255  -0.001787   0.005848  0.016272  0.219255
9.997417   0.306873   0.273378   0.227084  0.070929  0.306873
9.998083   0.292270   0.102497   0.097461  0.097423  0.292270
9.998750   0.116616   0.075370   0.181278  0.078857  0.235369
9.999417   0.174820   0.225460   0.016081  0.146239  0.225460

[15000 rows x 5 columns]
```

[91]:
```
px.line(df.rolling(8).max()[::8]).show()
```

**3.3.9 Datetime Data (Yay Finance!)** Let's use Yahoo Finance and stock data as a relatable example of data with datetimes.

[92]:
```
!pip install -q yfinance
import yfinance as yf
```

[93]:
```
df =  yf.download(["SPY", "AAPL", "MSFT", "AMZN", "GOOGL"],
                  start='2019-01-01',
                  end='2021-09-24')
df
```

```
/tmp/ipython-input-1766271411.py:1: FutureWarning:
```

```
YF.download() has changed argument auto_adjust default to True

[**********************100%**********************]  5 of 5 completed
```

[93]:

| Price | Close | | | | | \ |
|---|---|---|---|---|---|---|
| Ticker | AAPL | AMZN | GOOGL | MSFT | SPY | |
| Date | | | | | | |
| 2019-01-02 | 37.617863 | 76.956497 | 52.419624 | 94.945503 | 226.285782 | |
| 2019-01-03 | 33.870842 | 75.014000 | 50.967831 | 91.452667 | 220.885971 | |
| 2019-01-04 | 35.316753 | 78.769501 | 53.582146 | 95.706032 | 228.284729 | |
| 2019-01-07 | 35.238148 | 81.475502 | 53.475292 | 95.828125 | 230.084671 | |
| 2019-01-08 | 35.909904 | 82.829002 | 53.944973 | 96.522919 | 232.246384 | |
| … | … | … | … | … | … | |
| 2021-09-17 | 143.151245 | 173.126007 | 139.960617 | 290.638245 | 418.453369 | |
| 2021-09-20 | 140.093384 | 167.786499 | 137.892502 | 285.239777 | 411.476044 | |
| 2021-09-21 | 140.573593 | 167.181503 | 138.204147 | 285.724365 | 411.087402 | |
| 2021-09-22 | 142.945435 | 169.002502 | 139.447189 | 289.388000 | 415.097473 | |
| 2021-09-23 | 143.905914 | 170.800003 | 140.374146 | 290.337799 | 420.140930 | |

| Price | High | | | | | … | \ |
|---|---|---|---|---|---|---|---|
| Ticker | AAPL | AMZN | GOOGL | MSFT | SPY | … | |
| Date | | | | | | … | |
| 2019-01-02 | 37.839398 | 77.667999 | 52.723303 | 95.537032 | 227.217421 | … | |
| 2019-01-03 | 34.711717 | 76.900002 | 52.995173 | 94.072308 | 224.829556 | … | |
| 2019-01-04 | 35.385836 | 79.699997 | 53.678071 | 96.250618 | 228.935964 | … | |
| 2019-01-07 | 35.452537 | 81.727997 | 53.812268 | 96.964240 | 231.504716 | … | |
| 2019-01-08 | 36.164789 | 83.830498 | 54.341593 | 97.621475 | 232.734817 | … | |
| … | … | … | … | … | … | … | |
| 2021-09-17 | 145.856290 | 174.870499 | 142.594813 | 295.125711 | 422.216986 | … | |
| 2021-09-20 | 141.955540 | 170.949997 | 138.171332 | 289.523717 | 413.865029 | … | |
| 2021-09-21 | 141.720306 | 168.985001 | 139.176814 | 288.380033 | 415.144902 | … | |
| 2021-09-22 | 143.513871 | 169.449997 | 140.047092 | 290.977526 | 417.154677 | … | |
| 2021-09-23 | 144.150936 | 171.447998 | 140.851780 | 291.636543 | 421.762055 | … | |

| Price | Open | | | | | \ |
|---|---|---|---|---|---|---|
| Ticker | AAPL | AMZN | GOOGL | MSFT | SPY | |
| Date | | | | | | |
| 2019-01-02 | 36.896092 | 73.260002 | 51.053815 | 93.471369 | 222.486919 | |
| 2019-01-03 | 34.297233 | 76.000504 | 52.220321 | 93.987800 | 224.522019 | |
| 2019-01-04 | 34.428238 | 76.500000 | 51.817229 | 93.630978 | 223.943165 | |
| 2019-01-07 | 35.421569 | 80.115501 | 53.726286 | 95.433772 | 228.556077 | |
| 2019-01-08 | 35.626436 | 83.234497 | 53.976283 | 96.748262 | 232.291625 | |
| … | … | … | … | … | … | |
| 2021-09-17 | 145.856290 | 174.420502 | 142.177819 | 294.805883 | 421.790397 | |
| 2021-09-20 | 140.936258 | 169.800003 | 137.337830 | 287.207281 | 412.272372 | |
| 2021-09-21 | 141.063636 | 168.750000 | 138.916368 | 286.586980 | 413.836638 | |
| 2021-09-22 | 141.573306 | 167.550003 | 138.473536 | 287.594978 | 413.381570 | |

```
2021-09-23  143.729491  169.002502  140.147007  289.649665  416.984056

Price          Volume
Ticker           AAPL       AMZN     GOOGL      MSFT        SPY
Date
2019-01-02  148158800  159662000  31868000  35329300  126925200
2019-01-03  365248800  139512000  41960000  42579100  144140700
2019-01-04  234428400  183652000  46022000  44060600  142628800
2019-01-07  219111200  159864000  47446000  35656100  103139100
2019-01-08  164101200  177628000  35414000  31514400  102512600
…                 …          …         …         …          …
2021-09-17  129868800   92332000  53384000  41372500  118425000
2021-09-20  123478900   93382000  46518000  38278700  166445500
2021-09-21   75834000   55618000  25332000  22364100   92526100
2021-09-22   76404300   48228000  25056000  26626300  102350100
2021-09-23   64838200   47588000  20952000  18604600   76396000

[688 rows x 25 columns]
```

[94]:
```
df.columns.tolist()
```

[94]:
```
[('Close', 'AAPL'),
 ('Close', 'AMZN'),
 ('Close', 'GOOGL'),
 ('Close', 'MSFT'),
 ('Close', 'SPY'),
 ('High', 'AAPL'),
 ('High', 'AMZN'),
 ('High', 'GOOGL'),
 ('High', 'MSFT'),
 ('High', 'SPY'),
 ('Low', 'AAPL'),
 ('Low', 'AMZN'),
 ('Low', 'GOOGL'),
 ('Low', 'MSFT'),
 ('Low', 'SPY'),
 ('Open', 'AAPL'),
 ('Open', 'AMZN'),
 ('Open', 'GOOGL'),
 ('Open', 'MSFT'),
 ('Open', 'SPY'),
 ('Volume', 'AAPL'),
 ('Volume', 'AMZN'),
 ('Volume', 'GOOGL'),
 ('Volume', 'MSFT'),
 ('Volume', 'SPY')]
```

Let's compare not the price, but the relative performance.

[95]:
```
#There's no Adj Close column, so let's create one
# Normalize each stock's Close price to start at 1
df_modified = df.xs('Close', axis=1, level=0)  # select only 'Close' level
df_modified = df_modified / df_modified.iloc[0]

# Reset index
df_modified = df_modified.reset_index()

# Melt df to long format for plotting
df_melted = df_modified.melt(id_vars='Date', var_name='Stock',␣
 ↪value_name='Relative Performance')
```

[96]:
```
# Plot
px.line(df_melted, x='Date', y='Relative Performance', color='Stock').show()
```

The rolling function will play very nicely with datetime data as shown here when I get the moving average over a 40 day period. And this can handle unevenly sampled date easily.

[97]:
```
# Convert Date to datetime and set as index
df_melted['Date'] = pd.to_datetime(df_melted['Date'])

# Apply time-based rolling per stock
df_smoothed = (
    df_melted.set_index('Date')
    .groupby('Stock')['Relative Performance']
    .rolling('40d')
    .mean()
    .reset_index()
)

# Plot
px.line(df_smoothed, x='Date', y='Relative Performance', color='Stock').show()
```

Indexing with datetime data though will require a slightly extra step, but then it is easy.

[98]:
```
from datetime import date
start = date(2021, 4, 1)
end = date(2021, 4, 30)
```

[99]:
```
df[start:end]
```

[99]:

| Price | Close | | | | \ |
|---|---|---|---|---|---|
| Ticker | AAPL | AMZN | GOOGL | MSFT | SPY |
| Date | | | | | |
| 2021-04-01 | 120.166016 | 158.050003 | 105.854149 | 233.900192 | 377.336639 |
| 2021-04-05 | 122.999199 | 161.336502 | 110.286568 | 240.385910 | 382.752625 |
| 2021-04-06 | 123.302071 | 161.190994 | 109.804459 | 239.218048 | 382.526520 |
| 2021-04-07 | 124.953110 | 163.969498 | 111.284088 | 241.186920 | 382.969238 |

```
2021-04-08  127.356453  164.964996  111.850685  244.420105  384.787140
2021-04-09  129.935623  168.610001  112.856667  246.929489  387.584503
2021-04-12  128.216156  168.969498  111.561928  246.987381  387.725830
2021-04-13  131.332687  170.000000  112.049500  249.477402  388.875000
2021-04-14  128.987961  166.649994  111.427231  246.678528  387.546936
2021-04-15  131.401062  168.954498  113.581306  250.452240  391.710144
2021-04-16  131.068909  169.972000  113.457054  251.648972  393.019379
2021-04-19  131.733231  168.600494  113.805466  249.718689  391.088409
2021-04-20  130.043091  166.734497  113.271172  249.255417  388.225098
2021-04-21  130.424057  168.100998  113.238373  251.494568  391.898499
2021-04-22  128.900055  165.451996  111.954575  248.203461  388.319275
2021-04-23  131.225204  167.044006  114.310944  252.044693  392.529541
2021-04-26  131.615997  170.449997  114.807961  252.430710  393.348999
2021-04-27  131.293564  170.871506  113.866112  252.836121  393.264221
2021-04-28  130.502258  172.925003  117.248817  245.684479  393.151276
2021-04-29  130.404556  173.565506  118.924767  243.705917  395.656677
2021-04-30  128.431091  173.371002  116.973465  243.387421  393.057007

Price             High                                                …  \
Ticker            AAPL        AMZN       GOOGL        MSFT         SPY  …
Date                                                                  …
2021-04-01  121.318828  158.121994  106.150869  234.373098  377.393180  …
2021-04-05  123.253210  161.798004  110.785081  241.244879  383.298946  …
2021-04-06  124.200872  162.365494  110.735877  240.704348  383.581449  …
2021-04-07  124.972646  165.180496  111.555956  242.181007  383.317739  …
2021-04-08  127.385761  166.225006  112.912319  245.279073  384.843652  …
2021-04-09  129.974695  168.610001  113.020684  247.064607  387.754068  …
2021-04-12  129.789061  169.751999  112.171765  248.686026  387.998962  …
2021-04-13  131.557399  171.600006  112.498808  250.153007  389.506089  …
2021-04-14  131.889532  170.206497  112.695625  249.805552  389.911099  …
2021-04-15  131.889542  169.850006  114.115608  250.867240  391.983305  …
2021-04-16  131.567153  170.339996  114.028130  251.899916  393.631611  …
2021-04-19  132.348720  171.796494  114.517694  252.363175  392.529523  …
2021-04-20  132.407331  169.149506  114.264709  251.127778  390.975445  …
2021-04-21  130.668297  168.143005  113.320378  251.591087  392.105719  …
2021-04-22  131.059127  168.643494  113.761737  252.652713  392.567277  …
2021-04-23  132.006760  168.750000  114.618600  252.392156  393.951827  …
2021-04-26  131.948160  171.422501  115.533612  253.289693  393.923576  …
2021-04-27  132.290067  173.000000  115.231421  254.013585  393.848227  …
2021-04-28  131.909082  174.494003  120.844251  247.595454  394.667758  …
2021-04-29  133.911852  175.722504  119.491368  247.170759  396.278338  …
2021-04-30  130.482697  177.699997  118.382510  244.256050  394.224989  …

Price             Open                                                \
Ticker            AAPL        AMZN       GOOGL        MSFT         SPY
Date
2021-04-01  120.810812  155.897003  103.988841  230.155468  375.255037
```

```
2021-04-05  121.015972  158.649994  106.717473  234.295903  380.021105
2021-04-06  123.585390  161.187500  109.886964  238.976765  382.187448
2021-04-07  122.930804  161.690002  109.995316  239.169794  382.357006
2021-04-08  125.978936  165.544998  112.525137  243.956845  384.231419
2021-04-09  126.809356  165.235001  111.602690  244.053380  384.664620
2021-04-12  129.466663  167.760498  112.034095  245.829223  386.981717
2021-04-13  129.388547  170.042496  111.878528  248.290306  387.622279
2021-04-14  131.830917  170.201996  112.695625  248.502645  388.846741
2021-04-15  130.736737  168.550003  112.423258  248.936973  389.703881
2021-04-16  131.205682  169.000000  113.779621  250.423262  393.009950
2021-04-19  130.433873  169.516495  112.832306  251.118145  392.077427
2021-04-20  131.909087  168.679993  114.189659  248.830756  389.864004
2021-04-21  129.310324  165.800003  112.925248  249.911764  387.603413
2021-04-22  129.974702  168.584000  113.091752  251.137446  391.728996
2021-04-23  129.114968  165.955002  112.674260  248.888716  388.884373
2021-04-26  131.723463  167.399994  114.539073  252.536890  393.188891
2021-04-27  131.899274  172.173492  115.190665  252.459704  393.650406
2021-04-28  131.215435  171.740005  118.911840  247.151472  393.537460
2021-04-29  133.325671  175.255005  118.749320  246.553074  395.901582
2021-04-30  128.743711  176.255997  117.695135  241.032507  393.367852

Price          Volume
Ticker           AAPL       AMZN     GOOGL      MSFT       SPY
Date
2021-04-01   75089100   58806000  39880000  30338000  99682900
2021-04-05   88651200   66698000  48510000  36910600  91684800
2021-04-06   80171300   50756000  35240000  22931900  62021000
2021-04-07   83466700   66924000  24134000  22719800  55836300
2021-04-08   88844600   56242000  28664000  23625200  57863100
2021-04-09  106686700   86830000  26146000  24326800  61104600
2021-04-12   91420000   65636000  25024000  27148700  56704900
2021-04-13   91266500   66316000  25682000  23837500  56551000
2021-04-14   87222800   62904000  21002000  23070900  61659900
2021-04-15   89347100   64672000  29174000  25627500  60229800
2021-04-16   84922400   63720000  26282000  24878600  82037300
2021-04-19   94264200   54508000  30290000  23209300  78498500
2021-04-20   94812300   52460000  22288000  19722900  81851800
2021-04-21   68847100   44224000  23204000  24030400  66793000
2021-04-22   84566500   51612000  24146000  25606200  97582800
2021-04-23   78657500   63856000  29066000  21462600  73209200
2021-04-26   66905100   97614000  32038000  19763300  52182400
2021-04-27   66015800   76542000  44386000  31014200  51303100
2021-04-28  107760100   92638000  81106000  46903100  51238900
2021-04-29  151101000  153648000  41234000  40589000  78544300
2021-04-30  109839500  140186000  44856000  30945100  85527000

[21 rows x 25 columns]
```

[100]:
```
pd.date_range(start='1/1/2019', end='08/31/2021', freq='M')
```

```
/tmp/ipython-input-3907196692.py:1: FutureWarning:

'M' is deprecated and will be removed in a future version, please use 'ME'
instead.
```

[100]:
```
DatetimeIndex(['2019-01-31', '2019-02-28', '2019-03-31', '2019-04-30',
               '2019-05-31', '2019-06-30', '2019-07-31', '2019-08-31',
               '2019-09-30', '2019-10-31', '2019-11-30', '2019-12-31',
               '2020-01-31', '2020-02-29', '2020-03-31', '2020-04-30',
               '2020-05-31', '2020-06-30', '2020-07-31', '2020-08-31',
               '2020-09-30', '2020-10-31', '2020-11-30', '2020-12-31',
               '2021-01-31', '2021-02-28', '2021-03-31', '2021-04-30',
               '2021-05-31', '2021-06-30', '2021-07-31', '2021-08-31'],
              dtype='datetime64[ns]', freq='ME')
```

[101]:
```
df.resample(rule='Q').max()
```

```
/tmp/ipython-input-2442371637.py:1: FutureWarning:

'Q' is deprecated and will be removed in a future version, please use 'QE'
instead.
```

[101]:

| Price | Close | | | | | \ |
|---|---|---|---|---|---|---|
| Ticker | AAPL | AMZN | GOOGL | MSFT | SPY | |
| Date | | | | | | |
| 2019-03-31 | 46.671360 | 90.962997 | 61.438038 | 113.361336 | 258.670380 | |
| 2019-06-30 | 50.656933 | 98.123001 | 64.423630 | 130.400406 | 268.781738 | |
| 2019-09-30 | 53.991020 | 101.049500 | 61.925613 | 133.958817 | 275.703308 | |
| 2019-12-31 | 71.000847 | 93.489998 | 67.717377 | 151.460648 | 297.627655 | |
| 2020-03-31 | 79.300591 | 108.511002 | 75.788963 | 179.797546 | 311.820618 | |
| 2020-06-30 | 89.073112 | 138.220505 | 72.798401 | 194.979980 | 299.618927 | |
| 2020-09-30 | 130.667374 | 176.572495 | 85.357574 | 222.477020 | 333.060974 | |
| 2020-12-31 | 133.341354 | 172.181503 | 90.704506 | 216.617599 | 351.009857 | |
| 2021-03-31 | 139.652832 | 169.000000 | 105.299484 | 235.904816 | 373.305206 | |
| 2021-06-30 | 134.031677 | 175.272003 | 121.805496 | 262.542084 | 404.511047 | |
| 2021-09-30 | 153.569534 | 186.570496 | 144.349777 | 295.823578 | 428.258545 | |

| Price | High | | | | | … | \ |
|---|---|---|---|---|---|---|---|
| Ticker | AAPL | AMZN | GOOGL | MSFT | SPY | … | |
| Date | | | | | | … | |
| 2019-03-31 | 47.293355 | 91.187500 | 61.453445 | 113.927104 | 259.079177 | … | |
| 2019-06-30 | 51.508596 | 98.220001 | 64.461906 | 130.987194 | 269.778520 | … | |
| 2019-09-30 | 54.581627 | 101.790001 | 63.041422 | 135.193273 | 276.269251 | … | |
| 2019-12-31 | 71.078210 | 95.070000 | 67.945008 | 152.022809 | 298.420310 | … | |

```
2020-03-31   79.305435  109.297501   76.080716  181.703187  312.502607  …
2020-06-30   90.494765  139.800003   73.349590  195.832676  299.813597  …
2020-09-30  134.367908  177.612503   85.790479  223.639113  334.038637  …
2020-12-31  135.389914  174.811996   91.641890  219.086815  355.309669  …
2021-03-31  141.535585  171.699997  106.617588  237.002537  374.878200  …
2021-06-30  134.472053  177.699997  122.361650  262.783924  405.191439  …
2021-09-30  154.128175  188.654007  145.382084  296.424468  429.071220  …

Price             Open                                                  \
Ticker            AAPL        AMZN       GOOGL        MSFT         SPY
Date
2019-03-31   46.731163   90.508499   61.076204  112.682376  257.562042
2019-06-30   50.451195   97.449997   63.643806  130.088103  268.945285
2019-09-30   54.191105  101.280998   61.771034  133.921161  275.584600
2019-12-31   70.389115   94.146004   67.793414  151.927521  298.365015
2020-03-31   78.704363  108.653503   75.904770  181.655543  311.313738
2020-06-30   88.701299  139.000000   72.727823  193.619531  297.968757
2020-09-30  133.988118  177.350006   84.469412  220.191259  331.357039
2020-12-31  134.668053  173.399506   90.484335  217.917533  350.944141
2021-03-31  140.082055  171.250504  104.761211  235.943324  372.781368
2021-06-30  133.325671  176.255997  121.968496  261.855273  404.340928
2021-09-30  153.853781  187.199997  144.350301  295.629697  428.381398

Price          Volume
Ticker           AAPL       AMZN      GOOGL      MSFT        SPY
Date
2019-03-31  365248800  230124000   82296000  55636400  144140700
2019-06-30  259309200  181974000  133178000  38033900  144729900
2019-09-30  277125600  121164000  121216000  48992400  178745400
2019-12-31  275978000  192528000   65428000  53477500  147142100
2020-03-31  426510000  311346000   96520000  97012700  392220700
2020-06-30  264476000  240764000  108358000  67111700  209243600
2020-09-30  374336800  177852000   91468000  78983000  148011100
2020-12-31  262330500  167728000   99878000  63354900  172304200
2021-03-31  185549500  141972000   97882000  69870600  183433000
2021-06-30  151101000  153648000   81106000  46903100  134811000
2021-09-30  140893200  199312000   95130000  41372500  166445500

[11 rows x 25 columns]
```

**3.3.10 Sorting & Filtering on Tabular Data** To highlight filtering in DataFrames, we'll use a dataset with a bunch of different columns/series of different types. This data was pulled directly from the enDAQ cloud API off some example recording files.

[102]:
```
df = pd.read_csv('https://info.endaq.com/hubfs/data/endaq-cloud-table.csv')
df
```

[102]:

| | Unnamed: 0 | tags | \ |
|---|---|---|---|
| 0 | 0 | ['Vibration Severity: High', 'Shock Severity: … | |
| 1 | 1 | ['Vibration Severity: Low', 'Acceleration Seve… | |
| 2 | 2 | ['Shock Severity: Very Low', 'Acceleration Sev… | |
| 3 | 3 | ['Shock Severity: Very Low', 'Acceleration Sev… | |
| 4 | 4 | ['Shock Severity: Very Low', 'Acceleration Sev… | |
| 5 | 5 | ['Drive-Test', 'Vibration Severity: Very Low',… | |
| 6 | 6 | ['Acceleration Severity: High', 'Shock Severit… | |
| 7 | 7 | ['Vibration Severity: High', 'Acceleration Sev… | |
| 8 | 8 | ['Shock Severity: Very Low', 'Acceleration Sev… | |
| 9 | 0 | ['Shock Severity: Medium', 'Acceleration Sever… | |
| 10 | 1 | ['Vibration Severity: High', 'Shock Severity: … | |
| 11 | 2 | ['Acceleration Severity: High', 'Shock Severit… | |
| 12 | 3 | ['Shock Severity: Very Low', 'Vibration Severi… | |
| 13 | 4 | ['Big-Mining'] | |
| 14 | 5 | ['Shock Severity: Medium', 'Acceleration Sever… | |
| 15 | 6 | ['Ford'] | |
| 16 | 7 | ['Shock Severity: High', 'Vibration Severity: … | |
| 17 | 8 | ['Shock Severity: Very Low', 'Surgical', 'Vibr… | |
| 18 | 9 | ['Acceleration Severity: High', 'Shock Severit… | |
| 19 | 10 | ['Vibration Severity: High', 'Shock Severity: … | |
| 20 | 11 | ['Vibration Severity: High', 'Mining-Hammer', … | |
| 21 | 12 | ['Mining-Hammer', 'Vibration Severity: High', … | |
| 22 | 13 | ['Shock Severity: Very Low', 'Vibration Severi… | |

| | id | serial_number_id | \ |
|---|---|---|---|
| 0 | 6cd65b8f-1187-3bf9-a431-35304a7f84b7 | 9695 | |
| 1 | 342aacc3-cd91-3124-8cf4-0c7fece6dcab | 9316 | |
| 2 | 5c5592b4-2dbc-3124-be8f-c7179eecda49 | 10118 | |
| 3 | ef832e45-50fa-38b4-8f2c-91769f7cef6e | 10309 | |
| 4 | 0396df6a-56b0-3e43-8e46-e1d0d7485ff5 | 9295 | |
| 5 | 5e293d65-c9e0-3aa1-9098-c9762e8fbc86 | 11046 | |
| 6 | cd81c850-9e22-3b20-b570-7411e7a144cc | 0 | |
| 7 | 0931a23d-3515-3d11-bf97-bb9b2c7863be | 11162 | |
| 8 | c1571d10-2329-3aea-aa5d-223733f6336b | 10916 | |
| 9 | 8ce137ff-904e-314b-81ff-ff2cea279db2 | 9874 | |
| 10 | 9d7383fb-40d6-34c1-9d0c-37f11c29bff5 | 11456 | |
| 11 | 5f35ed7f-ff55-36a3-896d-276f06a0e340 | 11456 | |
| 12 | f7240576-47c6-34b8-bdce-3fe11bb5f1c6 | 11046 | |
| 13 | c2c234cc-8055-3fba-ad8b-446c4b6f85dc | 5120 | |
| 14 | 69fd99f8-3d85-38ec-9f5d-67e9d7b7e868 | 10030 | |
| 15 | f38361de-30a7-3dda-aec4-e87a67d1ecbb | 9695 | |
| 16 | 40718e01-f422-3926-a22d-acf6daf1182b | 7530 | |
| 17 | ee6dc613-d422-347c-8119-f122a55ac81a | 11071 | |
| 18 | f642d81c-7fe7-37fd-ab75-d493fc0556db | 9680 | |
| 19 | 0c0e92ae-214a-32bd-a5bb-5e1801da06b2 | 9680 | |
| 20 | 99ac47a8-63c2-38a4-8d92-508698eb3752 | 9680 | |

```
21  ebf31fb2-9d36-37de-999c-3edb01f17fb2              9680
22  e39fe506-f39a-3301-866c-9da6a30f9577              9695

                                          file_name  file_size  \
0                       train-passing-1632515146.ide   10492602
1                         Seat-Top_09-1632515145.ide   10491986
2                              Bolted-1632515144.ide    6149229
3                             RMI-2000-1632515143.ide    5909632
4                         Seat-Base_21-1632515142.ide    5248836
5                        Drive-Home_01-1632515142.ide    3632799
6                         HiTest-Shock-1632515141.ide    2655894
7                   Calibration-Shake-1632515140.IDE    2218130
8                   FUSE_HSTAB_000005-1632515139.ide     537562
9                           Coffee_002-1631722736.IDE   60959516
10                100_Joules_900_lbs-1629315313.ide    1596714
11                 50_Joules_900_lbs-1629315312.ide    1597750
12                     Drive-Home_07-1626805222.ide   36225758
13                Mining-SSX28803_06-1626457584.IDE  402920686
14  200922_Moto_Max_Run5_Control_Larry-1626297441.ide    4780893
15                          ford_f150-1626296561.ide   96097059
16               Motorcycle-Car-Crash-1626277852.ide   10489262
17                surgical-instrument-1625829182.ide     541994
18          LOC__3__DAQ41551_11_01_02-1625170795.IDE    2343292
19             LOC__4__DAQ41551_15_05-1625170794.IDE    6927958
20             LOC__6__DAQ41551_25_01-1625170793.IDE    8664238
21          LOC__2__DAQ38060_06_03_05-1625170793.IDE    1519172
22                       Tilt_000000-1625156721.IDE     719403

    recording_length  recording_ts  created_ts  modified_ts  …  \
0          73.612335    1588184436  1632515146   1632515146  …
1         172.704559    1575800071  1632515146   1632515146  …
2          29.396118    1619041447  1632515144   1632515144  …
3          60.250855            24  1632515143   1632515143  …
4          83.092255    1575800210  1632515143   1632515143  …
5          61.755371    1616178955  1632515142   1632515142  …
6          20.331848    1543936974  1632515141   1632515141  …
7          27.882690    1621278970  1632515140   1632515140  …
8          18.491791    1619108004  1632515140   1632515140  …
9         769.299896     952113744  1631722736   1631722736  …
10         20.200623    1627327315  1629315313   1629315313  …
11         20.201752    1627329399  1629315312   1629315312  …
12        634.732056    1616182557  1626805222   1626805222  …
13       3238.119202    1536953304  1626457585   1626457585  …
14         99.325134    1600818455  1626297442   1626297442  …
15       1207.678344    1584142508  1626296561   1626296561  …
16        151.069336    1562173372  1626277852   1626277852  …
17          6.951172    1619110390  1625829183   1625829183  …
```

```
18         28.456818    1616645179  1625170795   1625170795  …
19         64.486054    1616646130  1625170794   1625170794  …
20         63.878937    1616648007  1625170793   1625170793  …
21         27.057647    1616640862  1625170793   1625170793  …
22         23.355163    1625156461  1625156722   1625156722  …

                                    psdResultantOctave  \
0   [0.    0.    0.001 0.024 0.032 0.008 0.008 0.0…
1   [0.    0.    0.    0.    0.    0.    0.    0. …
2   [0.001 0.003 0.004 0.064 0.106 0.102 0.103 0.1…
3   [0.    0.    0.    0.    0.001 0.002 0.002 0.0…
4   [0.002 0.007 0.008 0.005 0.002 0.005 0.002 0.0…
5   [0.    0.    0.    0.    0.001 0.    0.    0. …
6   [0.304 0.486 0.399 0.454 0.347 0.178 0.185 0.1…
7   [2.100e-02 7.400e-02 7.500e-02 4.900e-02 5.500…
8   [ 0.  0.  0.  0.  0.  0.  0. nan nan nan nan n…
9                                                  []
10  [0.071 0.201 0.293 0.686 1.565 1.028 0.856 0.9…
11  [0.074 0.196 0.327 0.874 1.521 0.917 0.478 0.7…
12                                                 []
13                                                 []
14          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
15                                                 []
16          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
17          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
18          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
19          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
20          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
21          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
22          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

                                      samplePeakWindow  temperatureMeanFull  \
0           [1.241 1.682 1.944 … 1.852 1.372 1.473]               23.432
1           [0.035 0.024 0.007 … 0.067 0.031 0.022]               20.133
2           [5.283 5.944 5.19  … 0.356 1.079 1.099]               23.172
3           [0.13  0.118 0.1   … 0.105 0.1   0.066]               21.806
4           [0.084 0.137 0.178 … 0.347 0.324 0.286]               17.820
5           [0.001 0.003 0.002 … 0.011 0.008 0.006]               29.061
6     [ 3.088  3.019  2.893 … 73.794 40.005 24.303]                9.538
7           [7.486 7.496 7.137 … 7.997 8.294 7.806]               24.545
8           [0.001 0.001 0.001 … 0.002 0.006 0.006]               18.874
9                                                  []               24.540
10          [0.304 0.274 0.296 … 1.513 1.313 0.652]               24.180
11          [0.304 0.21  0.098 … 0.693 1.418 0.932]               24.175
12                                                 []               28.832
13                                                 []                  NaN
14    [ 3.34  13.586 10.13  …  7.737  8.978  8.672]                   NaN
```

```
15                                                  []                  NaN
16          [3.371 3.641 3.649 … 6.168 7.463 7.04 ]                26.989
17          [0.667 0.549 1.05  … 1.109 2.86  3.99 ]                21.889
18  [ 32.679  24.925  30.95  … 106.511  27.715  …                  33.452
19    [14.223 14.903 24.949 … 11.482 21.319 32.766]                32.202
20  [ 66.83   54.723  63.536 … 109.259  85.341  …                  26.031
21  [ 33.294  29.392  19.99  … 175.123 162.043  …                  25.616
22          [0.033 0.029 0.03  …   nan   nan   nan]                26.410

    accelerometerSampleRateFull  \
0                        19999.0
1                         3996.0
2                        20000.0
3                         4012.0
4                         4046.0
5                         4013.0
6                        19997.0
7                         5000.0
8                          504.0
9                         5000.0
10                        5000.0
11                        5000.0
12                        4012.0
13                           NaN
14                        4014.0
15                           NaN
16                       10001.0
17                        5000.0
18                       20010.0
19                       19992.0
20                       19992.0
21                       19998.0
22                        5000.0

                                      psdPeakOctaves  microphonoeRMSFull  \
0   [0.    0.    0.001 0.131 0.229 0.045 0.05  0.0…                 NaN
1   [0.    0.    0.001 0.    0.001 0.    0.001 0.0…                 NaN
2   [0.001 0.004 0.004 0.128 0.125 0.119 0.14  0.1…              25.507
3   [0.    0.    0.    0.    0.008 0.01  0.019 0.0…               1.754
4   [0.002 0.014 0.016 0.013 0.003 0.031 0.003 0.0…                 NaN
5   [0.    0.    0.    0.    0.007 0.    0.    0. …              16.243
6   [ 0.304  0.537  0.479  0.811  0.624  0.554  0…                 NaN
7   [2.10000e-02 9.00000e-02 8.60000e-02 6.30000e-…                 NaN
8   [0.    0.    0.001 0.001 0.    0.    0.      n…                 NaN
9                                                 []                NaN
10  [0.071 0.236 0.311 1.148 2.036 1.791 1.498 2.7…                 NaN
11  [0.074 0.232 0.392 1.258 2.045 2.049 1.143 2.3…                 NaN
```

```
12                                                    []              16.277
13                                                    []                 NaN
14  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
15                                                    []                 NaN
16  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
17  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
18  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
19  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
20  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
21  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN
22  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]\n [0…                   NaN

    gyroscopeRMSFull                                pvssResultantOctave  \
0              0.625  [ 10.333  25.952  30.28   52.579 172.174 275.8…
1              0.945  [53.648 76.9   43.779 44.217 39.351 11.396  9…
2                NaN  [ 26.338  27.636  64.097 102.733 107.863 124.6…
3              1.557  [ 0.854  1.159  1.662  1.815  3.022  6.139 10…
4              2.666  [ 74.73   79.453 101.006 151.429  73.92   53.4…
5              0.363  [ 2.336  7.82  19.078 10.384 16.975 38.326 18…
6                NaN  [1378.444 2470.941 4368.78  5033.327 5814.49  …
7              0.166  [ 90.703 198.651 288.078 183.492 126.417 108.4…
8              0.749  [ 2.617  6.761 17.326 34.067 28.721  9.469 15…
9              0.082                                                []
10            24.471  [ 194.741  361.14   563.488  855.34  1712.229 …
11            15.575  [ 242.141  388.517  736.981 1070.284 1843.722 …
12             4.185                                                []
13               NaN                                                []
14               NaN  [165.983 278.352 717.331 950.513 697.421 358.6…
15               NaN                                                []
16            19.810  [ 2723.153  5977.212 11406.291 12031.397  7337…
17             4.647  [ 88.953 171.73  104.303  71.184  58.883  72.4…
18           286.217  [2333.993 5629.021 6417.48  5802.895 3692.838 …
19           277.654  [ 378.188  748.336 1054.232 1115.369  756.905 …
20           245.929  [ 780.577 1168.663 1851.417 1094.945  793.122 …
21           266.815  [1073.684  940.873 1063.151  957.568  975.826 …
22            11.933  [142.046 273.02  216.774 129.288  54.011  24.0…

    accelerationRMSFull                                psdResultant1Hz
0                 0.372                          [0. 0. 0. … 0. 0. 0.]
1                 0.082                       [ 0.  0.  0. … nan nan nan]
2                 2.398          [0.    0.001 0.002 … 0.002 0.001 0.001]
3                 0.079                       [ 0.  0.  0. … nan nan nan]
4                 0.130          [0.001 0.002 0.002 …   nan   nan   nan]
5                 0.021                       [ 0.  0.  0. … nan nan nan]
6                11.645          [0.157 0.304 0.42  … 0.001 0.01  0.013]
7                 2.712          [0.007 0.021 0.055 …   nan   nan   nan]
8                 0.011                       [ 0.  0.  0. … nan nan nan]
```

```
9                 0.059                                                    []
10                2.877          [0.02  0.071 0.138 …   nan   nan   nan]
11                2.423          [0.021 0.074 0.137 …   nan   nan   nan]
12                0.097                                                    []
13                  NaN                                                    []
14                3.528          [0.006 0.016 0.04  …   nan   nan   nan]
15                  NaN                                                    []
16                1.732  [2.250e-01 8.390e-01 2.025e+00 … 1.000e-03 1…
17                1.568          [0.001 0.001 0.003 …   nan   nan   nan]
18               94.197    [ 2.09   8.631 18.196 … 69.565 59.225 60.801]
19               46.528    [ 0.072  0.302  0.816 … 52.226 45.804 52.81 ]
20               54.408    [ 0.162  0.587  1.495 … 45.639 43.86  40.177]
21              131.087    [ 0.371  0.409  0.787 … 27.577 24.778 37.822]
22                0.044          [0.008 0.02  0.03  …   nan   nan   nan]

[23 rows x 29 columns]
```

[103]:
```
df.columns
```

[103]:
```
Index(['Unnamed: 0', 'tags', 'id', 'serial_number_id', 'file_name',
       'file_size', 'recording_length', 'recording_ts', 'created_ts',
       'modified_ts', 'device', 'gpsLocationFull', 'velocityRMSFull',
       'gpsSpeedFull', 'pressureMeanFull', 'samplePeakStartTime',
       'displacementRMSFull', 'psuedoVelocityPeakFull', 'accelerationPeakFull',
       'psdResultantOctave', 'samplePeakWindow', 'temperatureMeanFull',
       'accelerometerSampleRateFull', 'psdPeakOctaves', 'microphonoeRMSFull',
       'gyroscopeRMSFull', 'pvssResultantOctave', 'accelerationRMSFull',
       'psdResultant1Hz'],
      dtype='object')
```

There's a lot of data here! So we'll focus on just a handful of columns and convert the time in seconds to a datetime object.

[104]:
```
df = df[['serial_number_id', 'file_name', 'file_size', 'recording_length',␣
 ↪'recording_ts',
        'accelerationPeakFull', 'psuedoVelocityPeakFull',␣
 ↪'accelerationRMSFull',
        'velocityRMSFull', 'displacementRMSFull', 'pressureMeanFull',␣
 ↪'temperatureMeanFull']].copy()

df['recording_ts'] = pd.to_datetime(df['recording_ts'], unit='s')
df = df.sort_values(by=['recording_ts'], ascending=False)
df
```

[104]:
```
    serial_number_id                                  file_name  \
11             11456          50_Joules_900_lbs-1629315312.ide
10             11456         100_Joules_900_lbs-1629315313.ide
22              9695                Tilt_000000-1625156721.IDE
```

```
7              11162                    Calibration-Shake-1632515140.IDE
17             11071                  surgical-instrument-1625829182.ide
8              10916                    FUSE_HSTAB_000005-1632515139.ide
2              10118                                Bolted-1632515144.ide
20              9680              LOC__6__DAQ41551_25_01-1625170793.IDE
19              9680              LOC__4__DAQ41551_15_05-1625170794.IDE
18              9680           LOC__3__DAQ41551_11_01_02-1625170795.IDE
21              9680           LOC__2__DAQ38060_06_03_05-1625170793.IDE
12             11046                       Drive-Home_07-1626805222.ide
5              11046                       Drive-Home_01-1632515142.ide
14             10030  200922_Moto_Max_Run5_Control_Larry-1626297441.ide
0               9695                       train-passing-1632515146.ide
15              9695                           ford_f150-1626296561.ide
4               9295                        Seat-Base_21-1632515142.ide
1               9316                         Seat-Top_09-1632515145.ide
16              7530                 Motorcycle-Car-Crash-1626277852.ide
6                  0                       HiTest-Shock-1632515141.ide
13              5120                  Mining-SSX28803_06-1626457584.IDE
9               9874                           Coffee_002-1631722736.IDE
3              10309                             RMI-2000-1632515143.ide

    file_size  recording_length        recording_ts  accelerationPeakFull  \
11    1597750         20.201752 2021-07-26 19:56:39                231.212
10    1596714         20.200623 2021-07-26 19:21:55                218.634
22     719403         23.355163 2021-07-01 16:21:01                  0.378
7     2218130         27.882690 2021-05-17 19:16:10                  8.783
17     541994          6.951172 2021-04-22 16:53:10                  5.739
8      537562         18.491791 2021-04-22 16:13:24                  0.202
2     6149229         29.396118 2021-04-21 21:44:07                 15.343
20    8664238         63.878937 2021-03-25 04:53:27                564.966
19    6927958         64.486054 2021-03-25 04:22:10                585.863
18    2343292         28.456818 2021-03-25 04:06:19                622.040
21    1519172         27.057647 2021-03-25 02:54:22                995.670
12   36225758        634.732056 2021-03-19 19:35:57                 23.805
5     3632799         61.755371 2021-03-19 18:35:55                  0.479
14    4780893         99.325134 2020-09-22 23:47:35                 29.864
0    10492602         73.612335 2020-04-29 18:20:36                  7.513
15   96097059       1207.678344 2020-03-13 23:35:08                    NaN
4     5248836         83.092255 2019-12-08 10:16:50                  1.085
1    10491986        172.704559 2019-12-08 10:14:31                  1.105
16   10489262        151.069336 2019-07-03 17:02:52                480.737
6     2655894         20.331848 2018-12-04 15:22:54                619.178
13  402920686       3238.119202 2018-09-14 19:28:24                    NaN
9    60959516        769.299896 2000-03-03 20:02:24                  2.698
3     5909632         60.250855 1970-01-01 00:00:24                  0.332

    psuedoVelocityPeakFull  accelerationRMSFull  velocityRMSFull  \
```

| | | | |
|---|---|---|---|
| 11 | 2907.650 | 2.423 | 54.507 |
| 10 | 2961.256 | 2.877 | 53.875 |
| 22 | 330.946 | 0.044 | 11.042 |
| 7 | 1142.282 | 2.712 | 46.346 |
| 17 | 387.312 | 1.568 | 24.418 |
| 8 | 53.375 | 0.011 | 1.504 |
| 2 | 148.276 | 2.398 | 14.101 |
| 20 | 2357.599 | 54.408 | 145.223 |
| 19 | 2153.020 | 46.528 | 148.591 |
| 18 | 8907.949 | 94.197 | 372.049 |
| 21 | 5845.241 | 131.087 | 323.287 |
| 12 | 356.128 | 0.097 | 6.117 |
| 5 | 40.197 | 0.021 | 1.081 |
| 14 | 1280.349 | 3.528 | 55.569 |
| 0 | 419.944 | 0.372 | 6.969 |
| 15 | NaN | NaN | NaN |
| 4 | 251.009 | 0.130 | 7.318 |
| 1 | 86.595 | 0.082 | 1.535 |
| 16 | 12831.590 | 1.732 | 143.437 |
| 6 | 6058.093 | 11.645 | 167.835 |
| 13 | NaN | NaN | NaN |
| 9 | 1338.396 | 0.059 | 5.606 |
| 3 | 17.287 | 0.079 | 1.247 |

| | displacementRMSFull | pressureMeanFull | temperatureMeanFull |
|---|---|---|---|
| 11 | 1.066 | 98.745 | 24.175 |
| 10 | 1.053 | 98.751 | 24.180 |
| 22 | 0.345 | 99.510 | 26.410 |
| 7 | 0.617 | 102.251 | 24.545 |
| 17 | 0.242 | 99.879 | 21.889 |
| 8 | 0.036 | 90.706 | 18.874 |
| 2 | 0.154 | 99.652 | 23.172 |
| 20 | 3.088 | 102.875 | 26.031 |
| 19 | 2.615 | 105.750 | 32.202 |
| 18 | 9.580 | 105.682 | 33.452 |
| 21 | 3.144 | 104.473 | 25.616 |
| 12 | 0.135 | 101.988 | 28.832 |
| 5 | 0.023 | 100.284 | 29.061 |
| 14 | 1.060 | NaN | NaN |
| 0 | 0.061 | 104.620 | 23.432 |
| 15 | NaN | NaN | NaN |
| 4 | 0.190 | 98.930 | 17.820 |
| 1 | 0.040 | 98.733 | 20.133 |
| 16 | 3.988 | 100.363 | 26.989 |
| 6 | 4.055 | 101.126 | 9.538 |
| 13 | NaN | NaN | NaN |
| 9 | 0.104 | 100.339 | 24.540 |

```
3                 0.005           100.467               21.806
```

Filtering is made simple with boolean expressions that can be combined. There is also a method to sort_values by columns/series.

[105]:
```
mask = df.recording_ts > pd.to_datetime('2021-01-01')
df[mask].sort_values(by=['serial_number_id'], ascending=False)
```

[105]:

| | serial_number_id | file_name | file_size | \ |
|---|---|---|---|---|
| 11 | 11456 | 50_Joules_900_lbs-1629315312.ide | 1597750 | |
| 10 | 11456 | 100_Joules_900_lbs-1629315313.ide | 1596714 | |
| 7 | 11162 | Calibration-Shake-1632515140.IDE | 2218130 | |
| 17 | 11071 | surgical-instrument-1625829182.ide | 541994 | |
| 12 | 11046 | Drive-Home_07-1626805222.ide | 36225758 | |
| 5 | 11046 | Drive-Home_01-1632515142.ide | 3632799 | |
| 8 | 10916 | FUSE_HSTAB_000005-1632515139.ide | 537562 | |
| 2 | 10118 | Bolted-1632515144.ide | 6149229 | |
| 22 | 9695 | Tilt_000000-1625156721.IDE | 719403 | |
| 19 | 9680 | LOC__4__DAQ41551_15_05-1625170794.IDE | 6927958 | |
| 20 | 9680 | LOC__6__DAQ41551_25_01-1625170793.IDE | 8664238 | |
| 21 | 9680 | LOC__2__DAQ38060_06_03_05-1625170793.IDE | 1519172 | |
| 18 | 9680 | LOC__3__DAQ41551_11_01_02-1625170795.IDE | 2343292 | |

| | recording_length | recording_ts | accelerationPeakFull | \ |
|---|---|---|---|---|
| 11 | 20.201752 | 2021-07-26 19:56:39 | 231.212 | |
| 10 | 20.200623 | 2021-07-26 19:21:55 | 218.634 | |
| 7 | 27.882690 | 2021-05-17 19:16:10 | 8.783 | |
| 17 | 6.951172 | 2021-04-22 16:53:10 | 5.739 | |
| 12 | 634.732056 | 2021-03-19 19:35:57 | 23.805 | |
| 5 | 61.755371 | 2021-03-19 18:35:55 | 0.479 | |
| 8 | 18.491791 | 2021-04-22 16:13:24 | 0.202 | |
| 2 | 29.396118 | 2021-04-21 21:44:07 | 15.343 | |
| 22 | 23.355163 | 2021-07-01 16:21:01 | 0.378 | |
| 19 | 64.486054 | 2021-03-25 04:22:10 | 585.863 | |
| 20 | 63.878937 | 2021-03-25 04:53:27 | 564.966 | |
| 21 | 27.057647 | 2021-03-25 02:54:22 | 995.670 | |
| 18 | 28.456818 | 2021-03-25 04:06:19 | 622.040 | |

| | psuedoVelocityPeakFull | accelerationRMSFull | velocityRMSFull | \ |
|---|---|---|---|---|
| 11 | 2907.650 | 2.423 | 54.507 | |
| 10 | 2961.256 | 2.877 | 53.875 | |
| 7 | 1142.282 | 2.712 | 46.346 | |
| 17 | 387.312 | 1.568 | 24.418 | |
| 12 | 356.128 | 0.097 | 6.117 | |
| 5 | 40.197 | 0.021 | 1.081 | |
| 8 | 53.375 | 0.011 | 1.504 | |
| 2 | 148.276 | 2.398 | 14.101 | |
| 22 | 330.946 | 0.044 | 11.042 | |

```
19                2153.020              46.528          148.591
20                2357.599              54.408          145.223
21                5845.241             131.087          323.287
18                8907.949              94.197          372.049

    displacementRMSFull  pressureMeanFull  temperatureMeanFull
11                1.066            98.745               24.175
10                1.053            98.751               24.180
7                 0.617           102.251               24.545
17                0.242            99.879               21.889
12                0.135           101.988               28.832
5                 0.023           100.284               29.061
8                 0.036            90.706               18.874
2                 0.154            99.652               23.172
22                0.345            99.510               26.410
19                2.615           105.750               32.202
20                3.088           102.875               26.031
21                3.144           104.473               25.616
18                9.580           105.682               33.452
```

[106]:
```
mask = (df.recording_ts > pd.to_datetime('2021-01-01')) & (df.
 ↪accelerationPeakFull > 100)
df[mask].sort_values(by=['accelerationPeakFull'], ascending=False)
```

[106]:
```
    serial_number_id                                file_name  file_size  \
21              9680  LOC__2__DAQ38060_06_03_05-1625170793.IDE    1519172
18              9680  LOC__3__DAQ41551_11_01_02-1625170795.IDE    2343292
19              9680     LOC__4__DAQ41551_15_05-1625170794.IDE    6927958
20              9680     LOC__6__DAQ41551_25_01-1625170793.IDE    8664238
11             11456          50_Joules_900_lbs-1629315312.ide    1597750
10             11456         100_Joules_900_lbs-1629315313.ide    1596714

    recording_length        recording_ts  accelerationPeakFull  \
21         27.057647 2021-03-25 02:54:22               995.670
18         28.456818 2021-03-25 04:06:19               622.040
19         64.486054 2021-03-25 04:22:10               585.863
20         63.878937 2021-03-25 04:53:27               564.966
11         20.201752 2021-07-26 19:56:39               231.212
10         20.200623 2021-07-26 19:21:55               218.634

    psuedoVelocityPeakFull  accelerationRMSFull  velocityRMSFull  \
21                5845.241             131.087          323.287
18                8907.949              94.197          372.049
19                2153.020              46.528          148.591
20                2357.599              54.408          145.223
11                2907.650               2.423           54.507
10                2961.256               2.877           53.875
```

| | displacementRMSFull | pressureMeanFull | temperatureMeanFull |
|---|---|---|---|
| 21 | 3.144 | 104.473 | 25.616 |
| 18 | 9.580 | 105.682 | 33.452 |
| 19 | 2.615 | 105.750 | 32.202 |
| 20 | 3.088 | 102.875 | 26.031 |
| 11 | 1.066 | 98.745 | 24.175 |
| 10 | 1.053 | 98.751 | 24.180 |

Another preview to plotly, but visualizing dataframes is made easy, even with mixed types.

[107]:
```
px.scatter(df,
           x="recording_ts",
           y="accelerationRMSFull",
           size="recording_length",
           color="serial_number_id",
           hover_name="file_name",
           log_y=True,
           size_max=60).show()
```

Plotly automatically made my colors a colorbar because I specified it based on a numeric value. If instead I change the type to string and replot, we'll see discrete series for each device.

[108]:
```
df['device'] = df["serial_number_id"].astype(str)

px.scatter(df,
           x="recording_ts",
           y="accelerationRMSFull",
           size="recording_length",
           color="device",
           hover_name="file_name",
           log_y=True,
           size_max=60).show()
```

## 0.4 4. Supplementary Activity

[109]:
```
import pandas as pd
import numpy as np
import math
#load the dataset (diabetes.csv)

file_id = "1pu-jKb567z96eLTBt4oqohvsICmkK7U1" #from https://drive.google.com/
 ↪file/d/1pu-jKb567z96eLTBt4oqohvsICmkK7U1/view?usp=sharing
#construct downloadable URL
url = f"https://drive.google.com/uc?id={file_id}"

diabetes_df = pd.read_csv(url)
```

[110]:
```
"""This is also possible"""
import pandas as pd
import numpy as np
import math
import re

original_url = "https://drive.google.com/file/d/
 ↪1pu-jKb567z96eLTBt4oqohvsICmkK7U1/view?usp=sharing"
# Extract the file ID
file_id = re.search(r'/file/d/([^/]+)', original_url).group(1)

# Construct a downloadable URL
url = f"https://drive.google.com/uc?id={file_id}"

# Load dataset
diabetes = pd.read_csv(url)
```

Download the dataset attached on the canvas course and answer the following:

1. Identify the column names

[111]:
```
diabetes.columns.tolist()
```

[111]:
```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age',
 'Outcome']
```

2. Identify the data types of the data

[112]:
```
diabetes.dtypes
```

[112]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

3. Display the total number of records

[113]:
```
"""
diabetes.shape reveals the dataframe's dimensions in the form (rows, columns)
just get the [rows] to find the number of records
"""

print("Number of records:", diabetes.shape[0])
```

```
Number of records: 768
```

4. Display the first 20 records

[114]:
```
diabetes.head(20) #adding a number between "()" specifies the number of records␣
↪to be displayed
```

[114]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | \ |
|---|---|---|---|---|---|---|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | |
| 5 | 5 | 116 | 74 | 0 | 0 | 25.6 | |
| 6 | 3 | 78 | 50 | 32 | 88 | 31.0 | |
| 7 | 10 | 115 | 0 | 0 | 0 | 35.3 | |
| 8 | 2 | 197 | 70 | 45 | 543 | 30.5 | |
| 9 | 8 | 125 | 96 | 0 | 0 | 0.0 | |
| 10 | 4 | 110 | 92 | 0 | 0 | 37.6 | |
| 11 | 10 | 168 | 74 | 0 | 0 | 38.0 | |
| 12 | 10 | 139 | 80 | 0 | 0 | 27.1 | |
| 13 | 1 | 189 | 60 | 23 | 846 | 30.1 | |
| 14 | 5 | 166 | 72 | 19 | 175 | 25.8 | |
| 15 | 7 | 100 | 0 | 0 | 0 | 30.0 | |
| 16 | 0 | 118 | 84 | 47 | 230 | 45.8 | |
| 17 | 7 | 107 | 74 | 0 | 0 | 29.6 | |
| 18 | 1 | 103 | 30 | 38 | 83 | 43.3 | |
| 19 | 1 | 115 | 70 | 30 | 96 | 34.6 | |

| | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|
| 0 | 0.627 | 50 | 1 |
| 1 | 0.351 | 31 | 0 |
| 2 | 0.672 | 32 | 1 |
| 3 | 0.167 | 21 | 0 |
| 4 | 2.288 | 33 | 1 |
| 5 | 0.201 | 30 | 0 |
| 6 | 0.248 | 26 | 1 |
| 7 | 0.134 | 29 | 0 |
| 8 | 0.158 | 53 | 1 |
| 9 | 0.232 | 54 | 1 |
| 10 | 0.191 | 30 | 0 |

```
11                        0.537   34        1
12                        1.441   57        0
13                        0.398   59        1
14                        0.587   51        1
15                        0.484   32        1
16                        0.551   31        1
17                        0.254   31        1
18                        0.183   33        0
19                        0.529   32        1
```

5. Display the last 20 records

[115]:
```
diabetes.tail(20)
```

[115]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
748            3      187             70             22      200  36.4
749            6      162             62              0        0  24.3
750            4      136             70              0        0  31.2
751            1      121             78             39       74  39.0
752            3      108             62             24        0  26.0
753            0      181             88             44      510  43.3
754            8      154             78             32        0  32.4
755            1      128             88             39      110  36.5
756            7      137             90             41        0  32.0
757            0      123             72              0        0  36.3
758            1      106             76              0        0  37.5
759            6      190             92              0        0  35.5
760            2       88             58             26       16  28.4
761            9      170             74             31        0  44.0
762            9       89             62              0        0  22.5
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
748                     0.408   36        1
749                     0.178   50        1
750                     1.182   22        1
751                     0.261   28        0
752                     0.223   25        0
753                     0.222   26        1
754                     0.443   45        1
755                     1.057   37        1
756                     0.391   39        0
757                     0.258   52        1
758                     0.197   26        0
```

```
759                       0.278   66        1
760                       0.766   22        0
761                       0.403   43        1
762                       0.142   33        0
763                       0.171   63        0
764                       0.340   27        0
765                       0.245   30        0
766                       0.349   47        1
767                       0.315   23        0
```

6. Change the Outcome column to Diagnosis

[116]:
```
# make sure to reflect changes into another dataframe to keep the original␣
 ↪dataframe's state
diabetes_1 = diabetes.rename(columns={"Outcome":"Diagnosis"})
diabetes_1.head()
```

[116]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Diagnosis
0                     0.627   50          1
1                     0.351   31          0
2                     0.672   32          1
3                     0.167   21          0
4                     2.288   33          1
```

7. Create a new column Classification that display “Diabetes” if the value of outcome is 1 , otherwise “No Diabetes”

[117]:
```
diabetes_1['Classification'] = np.where(diabetes_1['Diagnosis'] == 1,␣
 ↪'Diabetes', 'No Diabetes')
diabetes_1.head()
```

[117]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Diagnosis Classification
0                     0.627   50          1       Diabetes
1                     0.351   31          0    No Diabetes
```

```
2                     0.672   32           1       Diabetes
3                     0.167   21           0    No Diabetes
4                     2.288   33           1       Diabetes
```

8. Create a new dataframe “withDiabetes” that gathers data with diabetes

[118]:
```
withDiabetes = diabetes_1.query("Classification == 'Diabetes'").
 ↪reset_index(drop=True)
#reset_index organizes indexing in the new dataframe. when drop=True, it␣
 ↪prevents the old index from being added as a column
withDiabetes.head()
```

[118]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            8      183             64              0        0  23.3
2            0      137             40             35      168  43.1
3            3       78             50             32       88  31.0
4            2      197             70             45      543  30.5

   DiabetesPedigreeFunction  Age  Diagnosis Classification
0                     0.627   50          1       Diabetes
1                     0.672   32          1       Diabetes
2                     2.288   33          1       Diabetes
3                     0.248   26          1       Diabetes
4                     0.158   53          1       Diabetes
```

9. Create a new dataframe “noDiabetes” thats gathers data with no diabetes

[119]:
```
noDiabetes = diabetes_1.query("Classification == 'No Diabetes'").
 ↪reset_index(drop=True)
noDiabetes.head()
```

[119]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            1       85             66             29        0  26.6
1            1       89             66             23       94  28.1
2            5      116             74              0        0  25.6
3           10      115              0              0        0  35.3
4            4      110             92              0        0  37.6

   DiabetesPedigreeFunction  Age  Diagnosis Classification
0                     0.351   31          0    No Diabetes
1                     0.167   21          0    No Diabetes
2                     0.201   30          0    No Diabetes
3                     0.134   29          0    No Diabetes
4                     0.191   30          0    No Diabetes
```

10. Create a new dataframe “Pedia” that gathers data with age 0 to 19

[120]:
```
Pedia = diabetes_1[(diabetes_1['Age'] >= 0) & (diabetes_1['Age'] <= 19)].
 ↪reset_index(drop=True)
Pedia.head()
```

[120]:
```
Empty DataFrame
Columns: [Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI,
DiabetesPedigreeFunction, Age, Diagnosis, Classification]
Index: []
```

11. Create a new dataframe "Adult" that gathers data with age greater than 19

[121]:
```
Adult = diabetes_1[diabetes_1['Age'] > 19].sort_values(by='Age').
 ↪reset_index(drop=True)
Adult.head()
```

[121]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | \ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 126 | 86 | 27 | 120 | 27.4 | |
| 1 | 2 | 99 | 60 | 17 | 160 | 36.6 | |
| 2 | 1 | 114 | 66 | 36 | 200 | 38.1 | |
| 3 | 2 | 84 | 0 | 0 | 0 | 0.0 | |
| 4 | 0 | 78 | 88 | 29 | 40 | 36.9 | |

| | DiabetesPedigreeFunction | Age | Diagnosis | Classification |
|---|---|---|---|---|
| 0 | 0.515 | 21 | 0 | No Diabetes |
| 1 | 0.453 | 21 | 0 | No Diabetes |
| 2 | 0.289 | 21 | 0 | No Diabetes |
| 3 | 0.304 | 21 | 0 | No Diabetes |
| 4 | 0.434 | 21 | 0 | No Diabetes |

12. Use numpy to get the average age and glucose value.

[122]:
```
average_age = np.mean(diabetes_1['Age']).round(5)
average_glucose = np.mean(diabetes_1['Glucose']).round(5)

print("Average Age:", average_age)
print("Average Glucose:", average_glucose)
```

```
Average Age: 33.24089
Average Glucose: 120.89453
```

13. Use numpy to get the median age and glucose value.

[123]:
```
median_age = np.median(diabetes_1['Age']).round(5)
median_glucose = np.median(diabetes_1['Glucose']).round(5)

print("Median Age:", median_age)
print("Median Glucose:", median_glucose)
```

```
Median Age: 29.0
Median Glucose: 117.0
```

14. Use numpy to get the middle values of glucose and age.

[124]:
```
median_age = np.median(diabetes_1['Age']).round(5)
median_glucose = np.median(diabetes_1['Glucose']).round(5)

print("Median Age:", median_age)
print("Median Glucose:", median_glucose)
```

```
Median Age: 29.0
Median Glucose: 117.0
```

15. Use numpy to get the standard deviation of the skinthickness.

[125]:
```
std_skt = np.std(diabetes_1['SkinThickness'])
print("Standard Deviation of SkinThickness:", std_skt)
```

```
Standard Deviation of SkinThickness: 15.941828626496978
```

## 0.5 5. Summary, Conclusions and Lessons Learned

## 0.6 Summary

In this activity, I was able to learn about the different functions of numpy and pandas. I feel like numpy syntax in arrays just works in a similar way as MATLAB. Time-series data can be manipulated to be used in graphing via Plotly, though the dataframe must first be melted to a long format to be viable in a time-series graph. Data wrangling with dataframes can also be done via pandas using several APIs and functions.

## 0.7 Conclusions

I was able to accomplish the tasks in the supplementary with less difficulty as I was able to learn these concepts in data wrangling last semester. However, I struggled a bit in melting the dataframe for time-series data, although I managed to debug the code blocks in the Procedure. Overall, this activity was a nice refresher for using pandas and numpy.

## 0.8 Lessons Learned

In this activity, I learned about numpy syntax for handling arrays, and lists. I was also able to learn about numpy functions for computing measures of central tendency such as mean and median.

In addition to that, I was also able to learn about other syntax for pandas such as renaming columns. Most importantly, I was able to learn about usual stuff that I need to do when handling data frames, such as resetting the index for newly-created dataframes when filtering them, and making a copy of the original dataframe when making changes to maintain data integrity. Dataframe rolling was also new to me, which was quite interesting to work with.