# Experiment 2: Data Visualisation and Wrangling (ggplot)

- **Numpy** is a library for working with arrays of data.
- **Pandas** provides high-performance, easy-to-use data structures and data analysis tools.
- **Scipy** is a library of techniques for numerical and scientific computing.
- **Matplotlib** is a library for making graphs.
- **Seaborn** is a higher-level interface to Matplotlib that can be used to simplify many graphing tasks.
- **Statsmodels** is a library that implements many statistical techniques.

In [1]:

```
import numpy as np    #import numpy
import pandas as pd   #import pandas
```

In [ ]:

```
# calling a fuction from imported library
a = np.array([0,1,2,3,4,5,6,7,8,9,10])
np.mean(a)
```

#### Data Management

- A crucial component to statistical analysis and data science work
- The main data structure that Pandas works with is called a **Data Frame**.
- **2D Data**: Rows=> cases , Columns => variables

### Importing Data

In [ ]:

```
# Store the url string that hosts our .csv file (note that this is a different url than in the video)
url = "Cartwheeldata.csv"

# Read the .csv file and store it as a pandas Data Frame
df = pd.read_csv(url)

# Output object type
type(df)
```

### Viewing Data

In [ ]:

```
# We can view our Data Frame by calling the head() function
df.head()
```

In [ ]:

```
# Output entire Data Frame
df
```

In [ ]:

```
df.columns # columns
```

Splice data frame and select only specific portions of data. There are three different ways of doing so.

1. .loc()
2. .iloc()
3. .ix()

### .loc()

.loc() takes two single/list/range operator separated by ','. The first one indicates the row and the second one indicates columns.

In [ ]:

```
# Return all observations of CWDistance
df.loc[:,"CWDistance"]
```

In [ ]:

```
# Select all rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
df.loc[:,["CWDistance", "Height", "Wingspan"]]
```

In [ ]:

```
# Select few rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
df.loc[:9, ["CWDistance", "Height", "Wingspan"]]
```

In [ ]:

```
# Select range of rows for all columns
df.loc[10:15]
```

In [ ]:

```
#only first 10 observations
df.loc[:9, :]
```

### .iloc()

.iloc() is integer based slicing, whereas .loc() used labels/column names. Here are some examples:

In [ ]:

```
df.iloc[:4]
```

In [ ]:

```
df.iloc[1:5, 2:4]
```

In [ ]:

```
#df.iloc[1:5, ["Gender", "GenderGroup"]]
```

In [ ]:

```
df.dtypes
```

In [ ]:

```
# List unique values in the df['Gender'] column
df.Gender.unique()
```

In [ ]:

```
# Lets explore df["GenderGroup] as well
df.GenderGroup.unique()
```

It seems that these fields may serve the same purpose, which is to specify male vs. female. Lets check this quickly by observing only these two columns:

In [ ]:

```
# Use .loc() to specify a list of mulitple column names
df.loc[:,["Gender", "GenderGroup"]]
```

In [ ]:

```
df.groupby(['Gender','GenderGroup']).size()
```

This output indicates that we have two types of combinations.

- Case 1: Gender = F & Gender Group = 1
- Case 2: Gender = M & GenderGroup = 2.

This validates our initial assumption that these two fields essentially portray the same information.

## NHANES Data basics

In [ ]:

```
url = "nhanes_2015_2016.csv"
da = pd.read_csv(url)
da.head()
```

In [ ]:

```
da.shape # SIZE
```

### Exploring the contents of a data set

In [ ]:

```
da.columns
```

In [ ]:

```
da.dtypes
```

### Slicing a data set

To extract all the values for one variable, the following three approaches are equivalent ("DMDEDUC2" here is an NHANES variable containing a person's educational attainment). In these four lines of code, we are assigning the data from one column of the data frame `da` into new variables `w`, `x`, `y`, and `z`. The first three approaches access the variable by name. The fourth approach accesses the variable by position

In [ ]:

```
w = da["DMDEDUC2"]         # method 1 of slicing
x = da.loc[:, "DMDEDUC2"]  # method 2 of slicing
y = da.DMDEDUC2            # method 3 of slicing
z = da.iloc[:, 9]         # DMDEDUC2 is in column 9
```

**Another Reason to slice a variable** out of a data frame: **pass it into a function**. e.g, we can find the maximum value over all `DMDEDUC2` values

In [ ]:

```
print(da["DMDEDUC2"].max())
print(da.loc[:, "DMDEDUC2"].max())
print(da.DMDEDUC2.max())
print(da.iloc[:, 9].max())
```

variable `da` has type 'DataFrame', while one column of `da` has type 'Series'

In [ ]:

```
print(type(da)) # The type of the variable
print(type(da.DMDEDUC2)) # The type of one column of the data frame
print(type(da.iloc[2,:])) # The type of one row of the data frame
```

ust as a data frame's columns have names, the rows also have names, which are called the "index". However many data sets do not have meaningful row names, so it is more common to extract a row of a data frame using its position.

In [ ]:

```
x = da.iloc[3, :] # extracts row 3 from the data set
```

In [ ]:

```
# contiguos blocks of rows and columns
x = da.iloc[3:5, :]
y = da.iloc[:, 2:5]
```

### Missing values

There is a set of values including **'NA'**, **'NULL'**, and **'NaN'**

- `isnull` : Where are all Missing values
- `notnull` : Where are all non-missing Values

In [ ]:

```
print(pd.isnull(da.DMDEDUC2).sum()) # count of missing
print(pd.notnull(da.DMDEDUC2).sum()) # count of non-missing
print(da.DMDEDUC2.size)
```

# SciPy

Numpy provides a high-performance multidimensional array and basic tools to compute with and manipulate these arrays. SciPy builds on this, and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications.

For this course, we will primariyl be using the **SciPy.Stats** sub-library.

### SciPy.Stats

The SciPy.Stats module contains a large number of probability distributions as well as a growing library of statistical functions such as:

- Continuous and Discrete Distributions (i.e Normal, Uniform, Binomial, etc.)

- Descriptive Statistcs
- Statistical Tests (i.e T-Test)

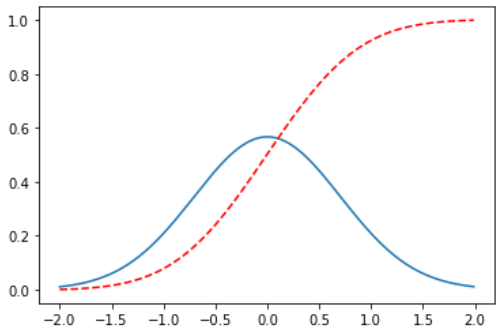
In [4]:

```
from scipy import stats
import numpy as np
### Print Normal Random Variables
print(stats.norm.rvs(size = 10))
#for practice only print(stats.norm.rvs(size =10))
```

```
[-1.40310399  0.73265606  1.75603904  1.50815214 -1.59613411  1.04872966
  0.46294439 -0.18368981  1.49728474 -0.59350861]
```

In [5]:

```
from pylab import *

# Create some test data
dx = .01
X  = np.arange(-2,2,dx)
Y  = exp(-X**2)

# Normalize the data to a proper PDF

Y /= (dx*Y).sum()

# Compute the CDF
CY = np.cumsum(Y*dx)

# Plot both
plot(X,Y)
plot(X,CY,'r--')

show()
```




In [ ]:

```
### Compute the Normal CDF of certain values.
print(stats.norm.cdf(np.array([1,-1., 0, 1, 3, 4, -2, 6])))
```

**Descriptive Statistics**

In [9]:

```
np.random.seed(282629734)
# Generate 1000 Student's T continuous random variables.
x = stats.t.rvs(10, size=1000)
```

In [10]:

```
# Do some descriptive statistics
print(x.min())   # equivalent to np.min(x)

print(x.max())   # equivalent to np.max(x)

print(x.mean())  # equivalent to np.mean(x)

print(x.var())   # equivalent to np.var(x))

stats.describe(x) #decribes the parameters of distribution
```

```
-3.7897557242248197
5.263277329807165
0.014061066398468422
1.288993862079285
```

Out[10]:

```
DescribeResult(nobs=1000, minmax=(-3.7897557242248197, 5.263277329807165), mean=0.014061066398468422, variance=1.2902841462255106, skewness=0.21652778283120955, kurtosis=1.055594041706331)
```

## MatPlotLib

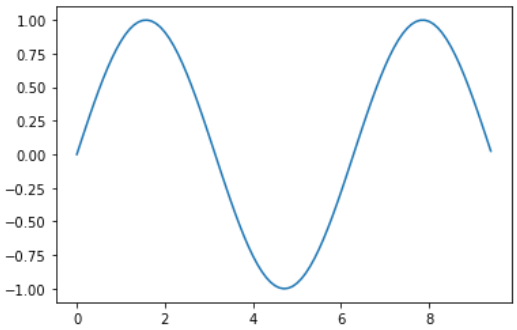
Matplotlib is a plotting library. In this section give a brief introduction to the matplotlib.pyplot module.

In [ ]:

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

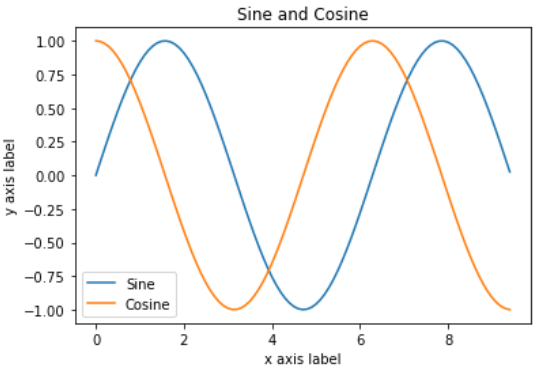
In [6]:

```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = sin(x)
# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```



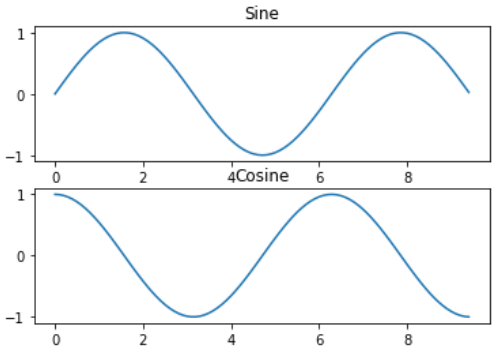

In [7]:

```
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



In [8]:

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```




## Seaborn

Seaborn is complimentary to Matplotlib and it specifically targets statistical data visualization. But it goes even further than that: Seaborn extends Matplotlib and makes generating visualizations convenient.

While Matplotlib is a robust solution for various problems, Seaborn utilizes more concise paramesters for ease-of-use.

#### Scatterplots

In [ ]:

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Store the url string that hosts our .csv file
url = "Cartwheeldata.csv"

# Read the .csv file and store it as a pandas Data Frame
df = pd.read_csv(url)

# Create Scatterplot
sns.lmplot(x='Wingspan', y='CWDistance', data=df)

plt.show()
```

In [ ]:

```
# Scatterplot arguments
sns.lmplot(x='Wingspan', y='CWDistance', data=df,
           fit_reg=False, # No regression line
           hue='Gender')   # Color by evolution stage

plt.show()
```

In [ ]:

```
# Construct Cartwheel distance plot
sns.swarmplot(x='Gender',y='CWDistance',data=df)
plt.show()
```

In [ ]:

```
sns.boxplot(data=df.loc[: , ['Age','Height','Wingspan','CWDistance','Score']])
plt.show()
```

In [ ]:

```
# Male Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'M', ["Age", "Height", "Wingspan", "CWDistance", "Score"]])
plt.show()
```

In [ ]:

```
# Female Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'F', ["Age", "Height", "Wingspan", "CWDistance", "Score"]])
plt.show()
```

In [ ]:

```
# Male Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'M', ["Score"]])

plt.show()
```

In [ ]:

```
# Female Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'F', ["Score"]])
plt.show()
```

#### Histogram

In [ ]:

```
# Distribution Plot (a.k.a. Histogram)
sns.distplot(df.CWDistance)
plt.show()
```

In [ ]:

```
# Count Plot (a.k.a. Bar Plot)
sns.countplot(x='Gender', data=df)

plt.xticks(rotation=-45)

plt.show()
```

## Visualizing Data in Python

#### Tables, Histograms, Boxplots, and Slicing for Statistics

When working with a new dataset, one of the most useful things to do is to begin to visualize the data. By using tables, histograms, box plots, and other visual tools, we can get a better idea of what the data may be trying to tell us, and we can gain insights into the data that we may have not discovered otherwise.

In [12]:

```
# We first need to import the packages that we will be using
import seaborn as sns # For plotting
import matplotlib.pyplot as plt # For showing plots

# Load in the data set
tips_data = sns.load_dataset("tips")
```

#### Visualizing the Data - Tables

In [ ]:

```
# Print out the first few rows of the data
tips_data.head()
```

In [ ]:

```
#### Describing Data
# Print out the summary statistics for the quantitative variables
tips_data.describe()
```

In [ ]:

```
# Plot a histogram of the total bill
sns.distplot(tips_data["total_bill"], kde = False).set_title("Histogram of Total Bill")
plt.show()
```

In [ ]:

```
# Plot a histogram of the Tips only
sns.distplot(tips_data["tip"], kde = False).set_title("Histogram of Total Tip")
plt.show()
```

In [ ]:

```
# Plot a histogram of both the total bill and the tips'
sns.distplot(tips_data["total_bill"], kde = False)
sns.distplot(tips_data["tip"], kde = False).set_title("Histogram of Both Tip Size and Total Bill")
plt.legend(['total bill','tip'])
plt.show()
```

In [ ]:

```
# Create a boxplot of the total bill amounts
sns.boxplot(tips_data["total_bill"]).set_title("Box plot of the Total Bill")
plt.show()
```

In [ ]:

```
# Create a boxplot of the tips amounts
sns.boxplot(tips_data["tip"]).set_title("Box plot of the Tip")
plt.show()
```

In [ ]:

```
# Create a boxplot and histogram of the tips grouped by smoking status
import seaborn as sns
sns.boxplot(x = tips_data["tip"], y = tips_data["smoker"])
plt.show()
```

In [ ]:

```
# Create a boxplot and histogram of the tips grouped by time of day
sns.boxplot(x = tips_data["tip"], y = tips_data["time"])
g = sns.FacetGrid(tips_data, row = "time")
g = g.map(plt.hist, "tip")
plt.show()
```

In [ ]:

```
# Create a boxplot and histogram of the tips grouped by the day
sns.boxplot(x = tips_data["tip"], y = tips_data["day"])

g = sns.FacetGrid(tips_data, row = "day")
g = g.map(plt.hist, "tip")
plt.show()
```