Python Libraries

For this tutorial, we are going to outline the most common uses for each of the following libraries:

- **Numpy** is a library for working with arrays of data.
- **Scipy** is a library of techniques for numerical and scientific computing.
- **Matplotlib** is a library for making visualizations.
- **Seaborn** is a higher-level interface to Matplotlib that can be used to simplify many visualization tasks.

Important*: While this tutorial provides insight into the basics of these libraries, I recommend digging into the documentation that is available online.*

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

We will focus on the numpy array object.

Numpy Array

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

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

In [2]:
```
### Create a 3x1 numpy array
a = np.array([1,2,3])

### Print object type
print(type(a))

### Print shape
print(a.shape)

### Print some values in a
print(a[0], a[1], a[2])

### Create a 2x2 numpy array
b = np.array([[1,2],[3,4]])

### Print shape
print(b.shape)

## Print some values in b
print(b[0,0], b[0,1], b[1,1])

### Create a 3x2 numpy array
c = np.array([[1,2],[3,4],[5,6]])

### Print shape
print(c.shape)
```

```
### Print some values in c
print(c[0,1], c[1,0], c[2,0], c[2,1])
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
(2, 2)
1 2 4
(3, 2)
2 3 5 6
```

In [3]:
```
### 2x3 zero array
d = np.zeros((2,3))

print(d)

### 4x2 array of ones
e = np.ones((4,2))

print(e)

### 2x2 constant array
f = np.full((2,2), 9)

print(f)

### 3x3 random array
g = np.random.random((3,3))

print(g)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
[[9 9]
 [9 9]]
[[0.84380993 0.34998064 0.69815133]
 [0.7631608  0.71279866 0.48071927]
 [0.89632893 0.84547575 0.04323561]]
```

Array Indexing

In [4]:
```
### Create 3x4 array
h = np.array([[1,2,3,4,], [5,6,7,8], [9,10,11,12]])

print(h)

### Slice array to make a 2x2 sub-array
i = h[:2, 1:3]

print(i)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[2 3]
 [6 7]]
```

In [5]:
```
print(h[0,1])

### Modify the slice
i[0,0] = 1738

### Print to show how modifying the slice also changes the base object
print(h[0,1])
```

```
2
1738
```

Datatypes in Arrays

In [6]:
```
### Integer
j = np.array([1, 2])
print(j.dtype)



### Float
k = np.array([1.0, 2.0])
print(k.dtype)

### Force Data Type
l = np.array([1.0, 2.0], dtype=np.int64)
print(l.dtype)
```

```
int64
float64
int64
```

Array Math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

In [7]:
```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
```

```
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

In [8]:
```
x = np.array([[1,2],[3,4]])

### Compute sum of all elements; prints "10"
print(np.sum(x))

### Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=0))

### Compute sum of each row; prints "[3 7]"
print(np.sum(x, axis=1))
```

```
10
[4 6]
[3 7]
```

In [9]:
```
x = np.array([[1,2],[3,4]])

### Compute mean of all elements; prints "2.5"
print(np.mean(x))

### Compute mean of each column; prints "[2 3]"
print(np.mean(x, axis=0))

### Compute mean of each row; prints "[1.5 3.5]"
print(np.mean(x, axis=1))
```

```
2.5
[2. 3.]
[1.5 3.5]
```

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 [1]:
```
from scipy import stats
import numpy as np
```

In [11]:
```
### Print Normal Random Variables
print(stats.norm.rvs(size = 10))
```

```
[ 0.24085899  1.5063869  -1.17484711  0.93922658  1.33109606  1.68452343
  0.39261572  1.97870701 -1.42821024 -1.68559446]
```

In [12]:
```
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()
```

```
<Figure size 640x480 with 1 Axes>
```

In [13]:
```
### Compute the Normal CDF of certain values.
print(stats.norm.cdf(np.array([1,-1., 0, 1, 3, 4, -2, 6])))
```

```
[0.84134475 0.15865525 0.5        0.84134475 0.9986501  0.99996833
 0.02275013 1.        ]
```

Descriptive Statistics

In [14]:
```
np.random.seed(282629734)

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

In [15]:
```python
# 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)
```

```
-3.7897557242248197
5.263277329807165
0.014061066398468422
1.288993862079285
```

Out[15]:
```
DescribeResult(nobs=1000, minmax=(-3.7897557242248197, 5.263277329807165), mean=0.0
14061066398468422, variance=1.2902841462255106, skewness=0.21652778283120955, kurto
sis=1.055594041706331)
```

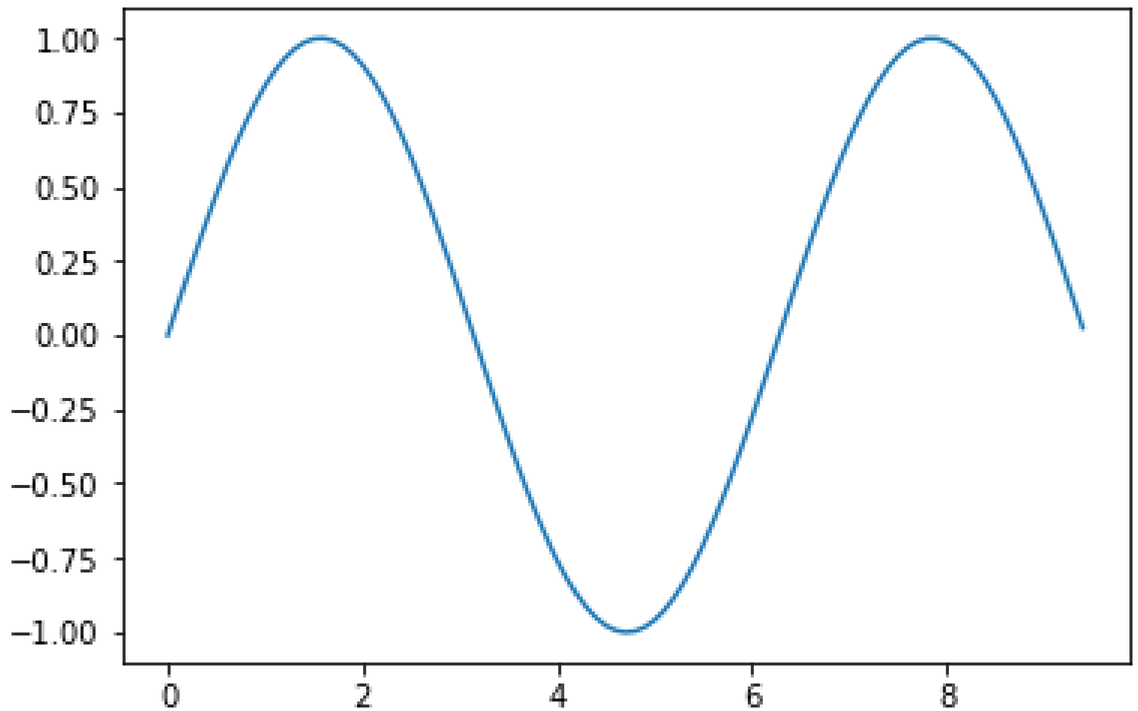
Later in the course, we will discuss distributions and statistical tests such as a T-Test. SciPy has built in functions for these operations.

MatPlotLib

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

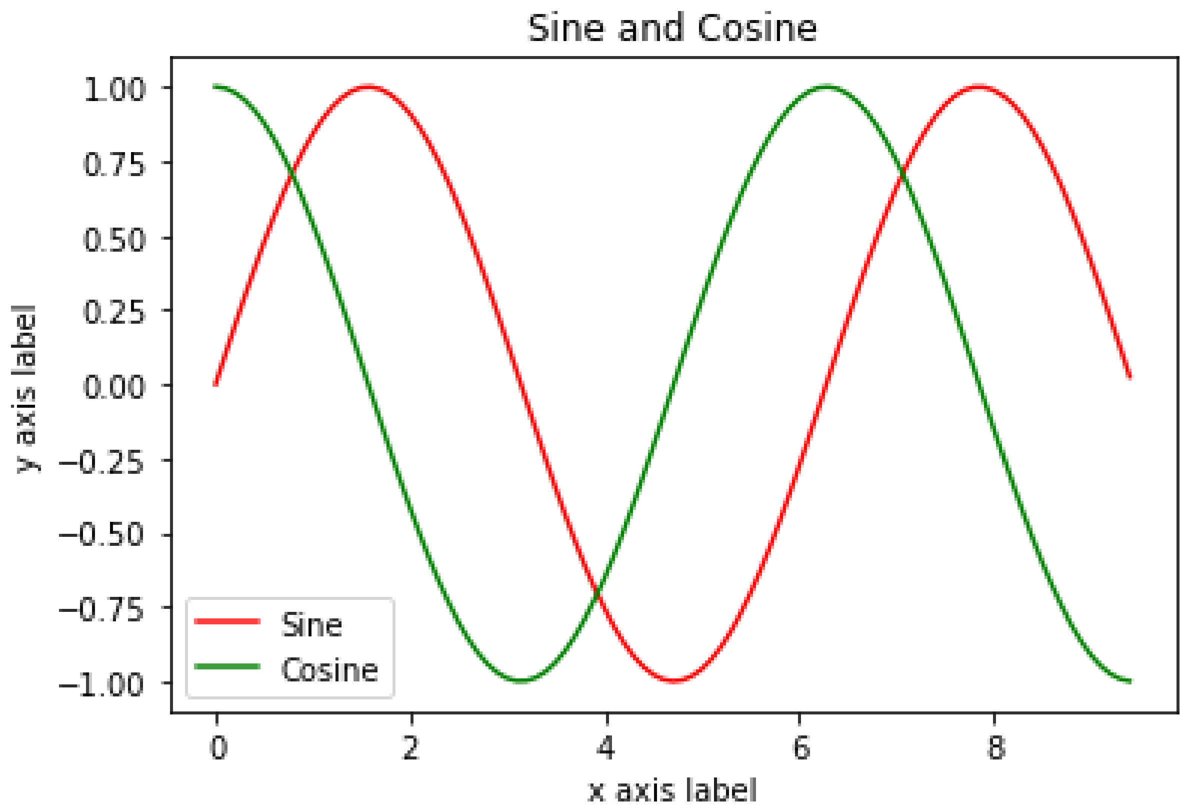
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [17]:
```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```



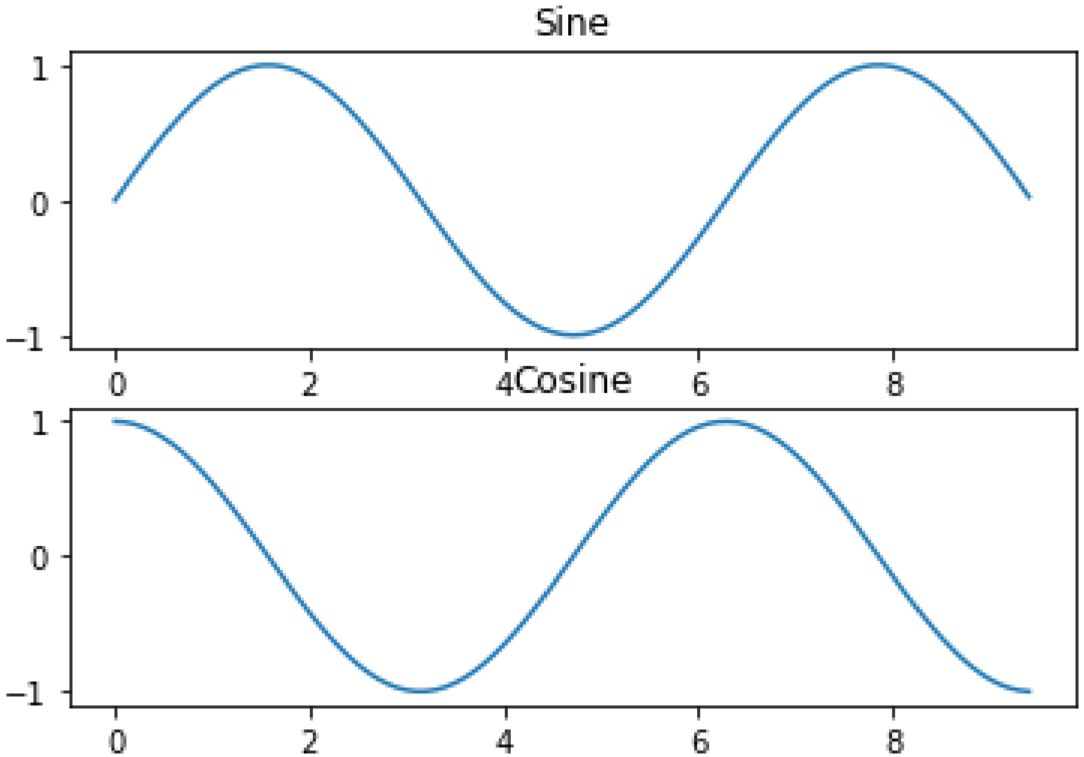
In [20]:

```
# 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, c = 'r')
plt.plot(x, y_cos, c = 'g')
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```


Sine and Cosine
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y axis label
0
2
4
6
8
x axis label
Sine
Cosine

Subplots

In [21]:
```python
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

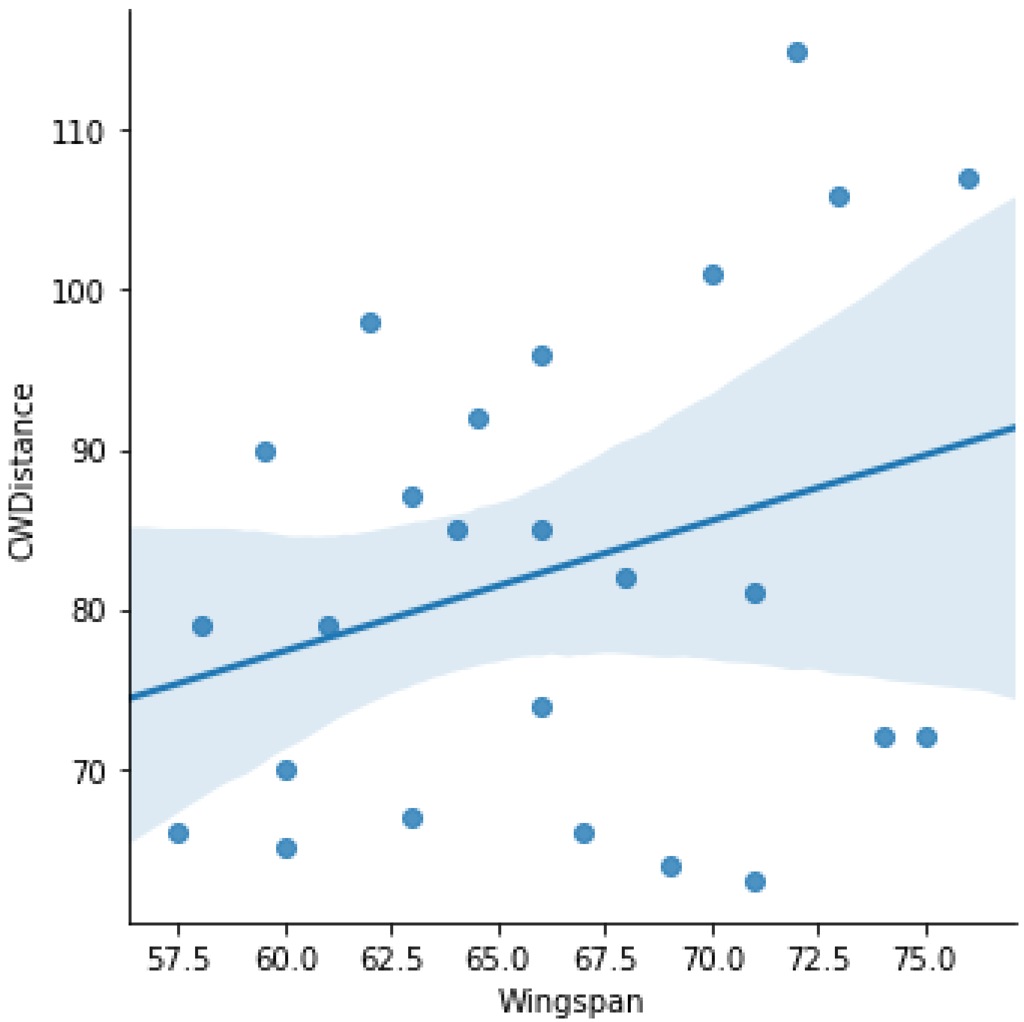
Seaborn is complimentary to Matplotlib and it specifically targets statistical data visualization. But it goes even further than that: Seaborn extends Matplotlib and makes generating visualizations convenient.

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

Scatterplots

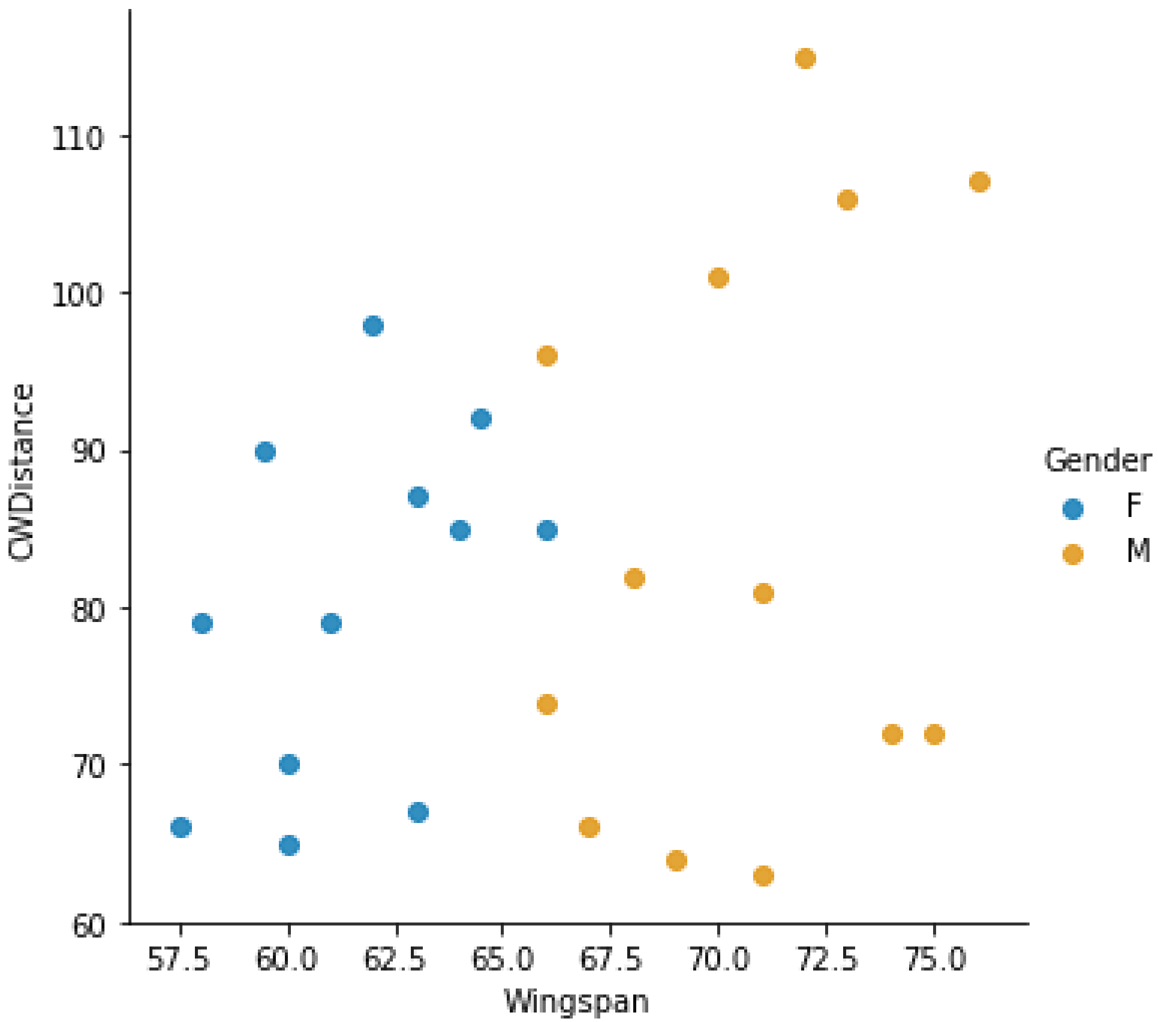
In [3]:

```
# 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()
```



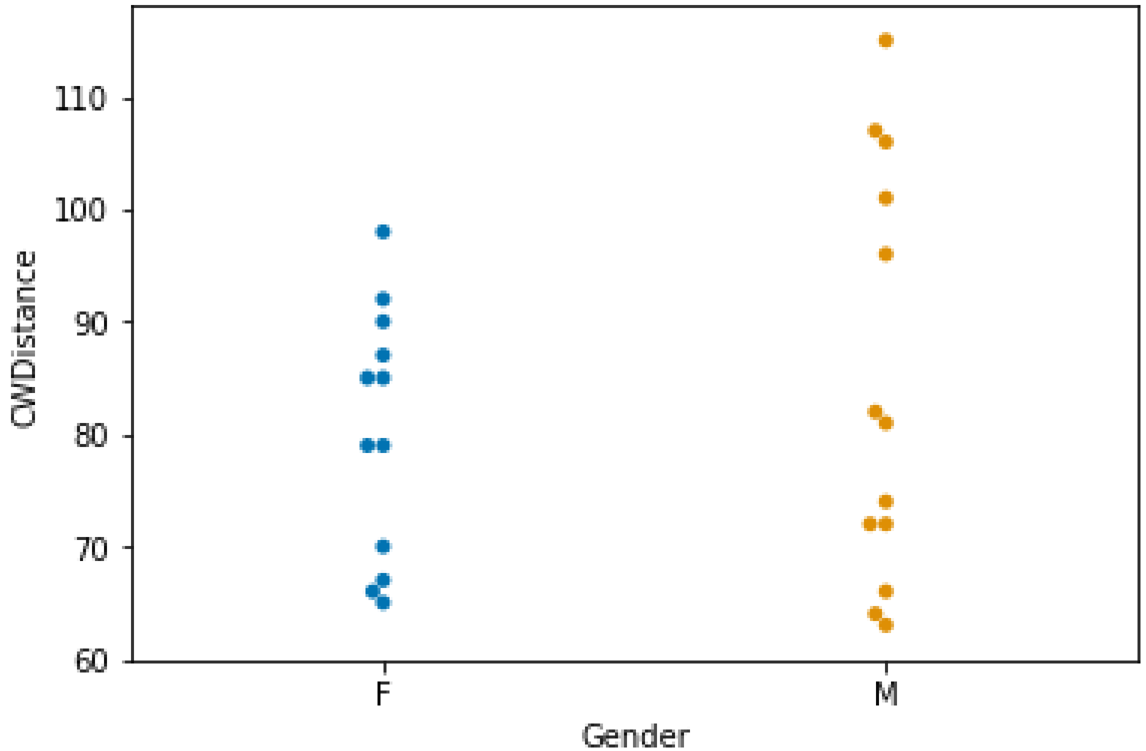
110
100
90
80
70
CWDistance
57.5
60.0
62.5
65.0
67.5
70.0
72.5
75.0
Wingspan

In [27]:
```python
# Scatterplot arguments
sns.set_palette("colorblind")

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

plt.show()
```



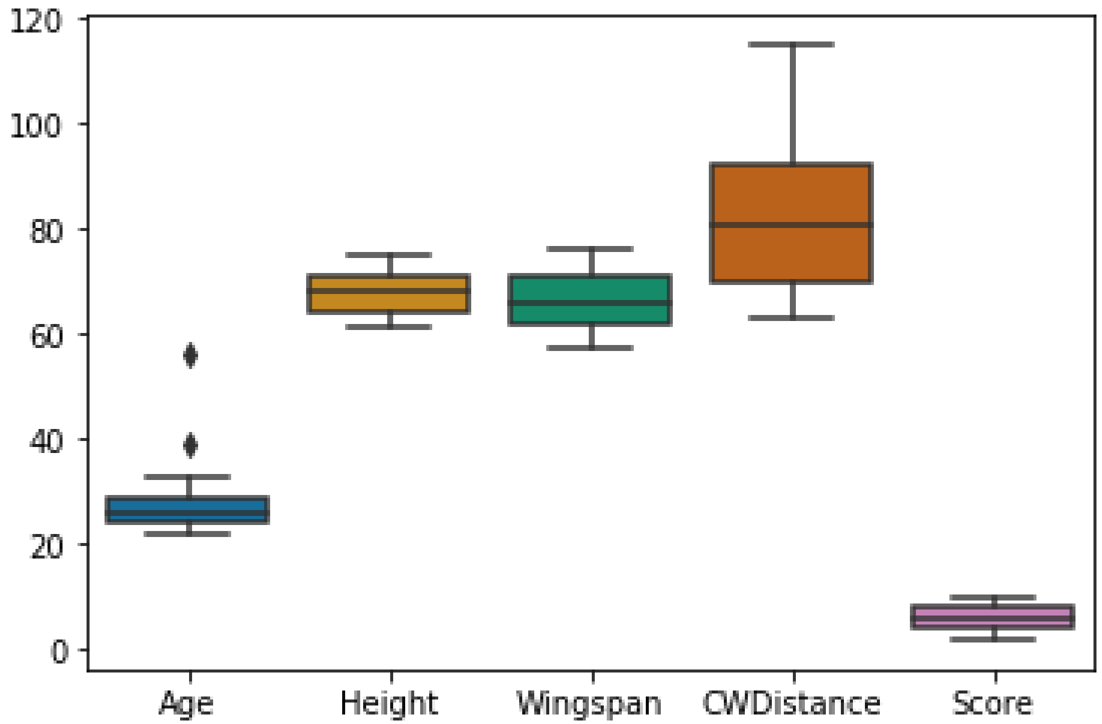
110
100
90
80
70
60
CWDistance
57.5
60.0
62.5
65.0
67.5
70.0
72.5
75.0
Wingspan
Gender
F
M

In [28]:
```
# Construct Cartwheel distance plot
sns.swarmplot(x="Gender", y="CWDistance", data=df)

plt.show()
```

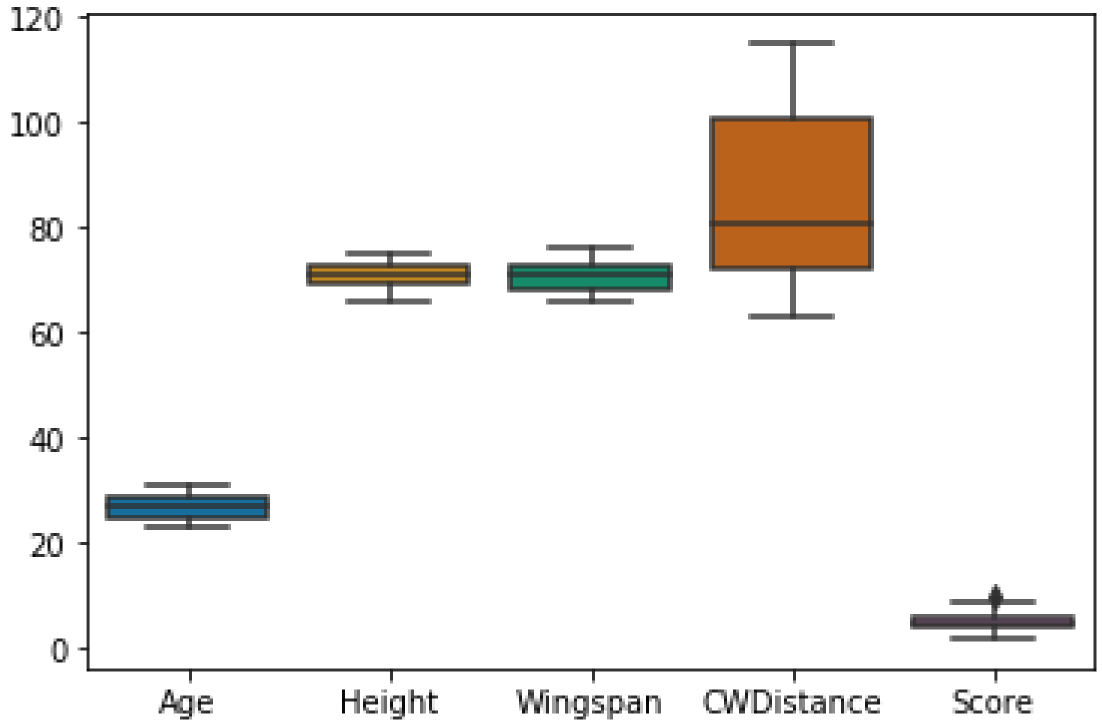


Boxplots

In [29]:
```
sns.boxplot(data=df.loc[:, ["Age", "Height", "Wingspan", "CWDistance", "Score"]])

plt.show()
```

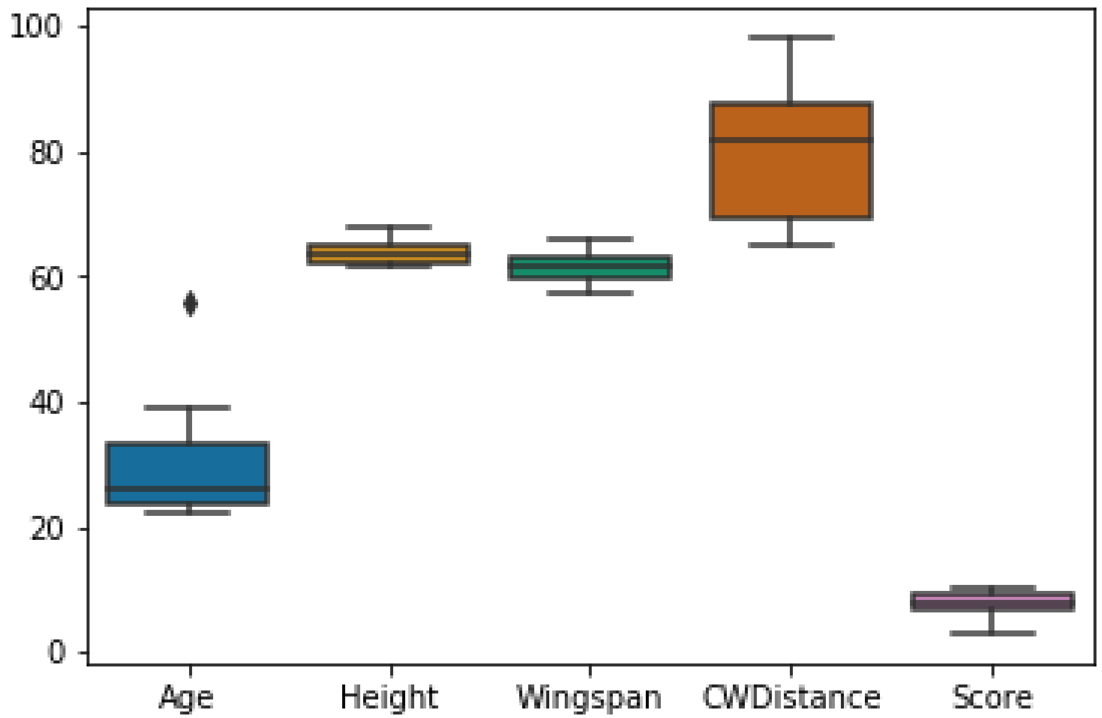


In [30]:
```
# Male Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'M', ["Age", "Height", "Wingspan", "CWDistanc
e", "Score"]])

plt.show()
```

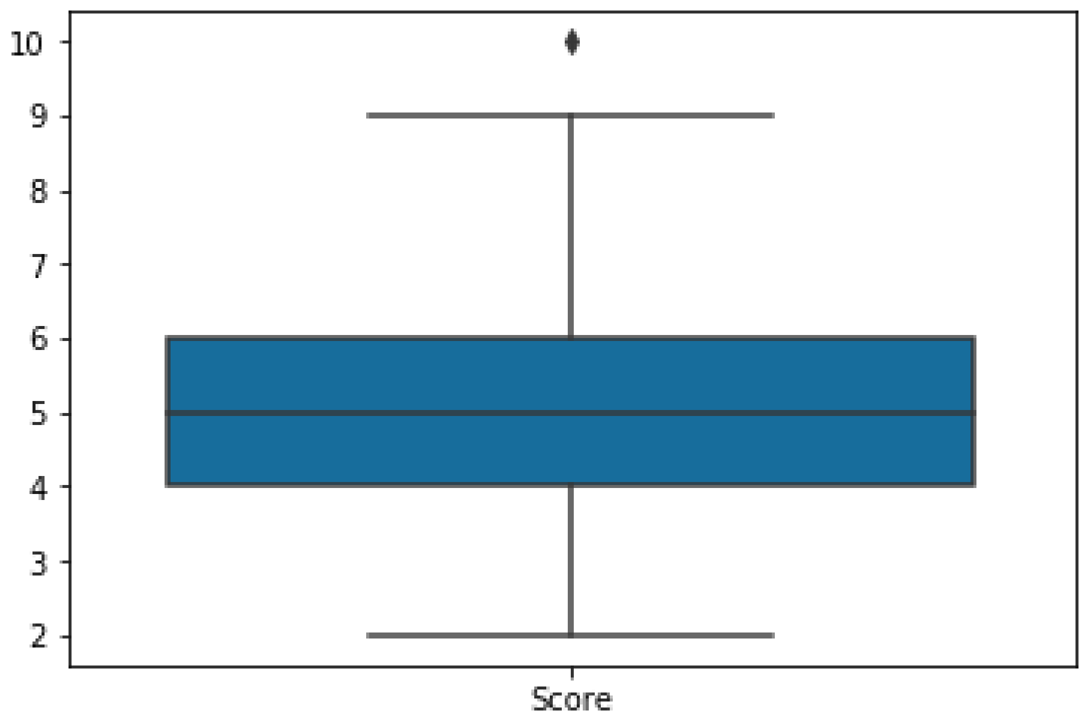


In [31]:

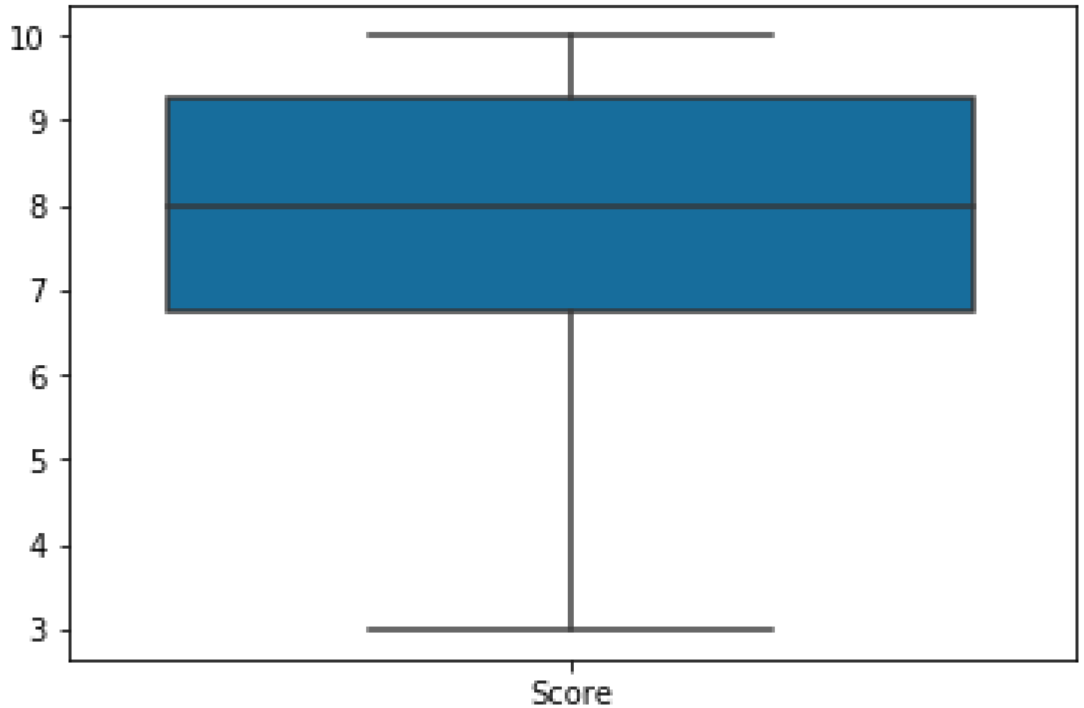
```
# Female Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'F', ["Age", "Height", "Wingspan", "CWDistanc
e", "Score"]])

plt.show()
```



In [32]:
```
# Male Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'M', ["Score"]])

plt.show()
```

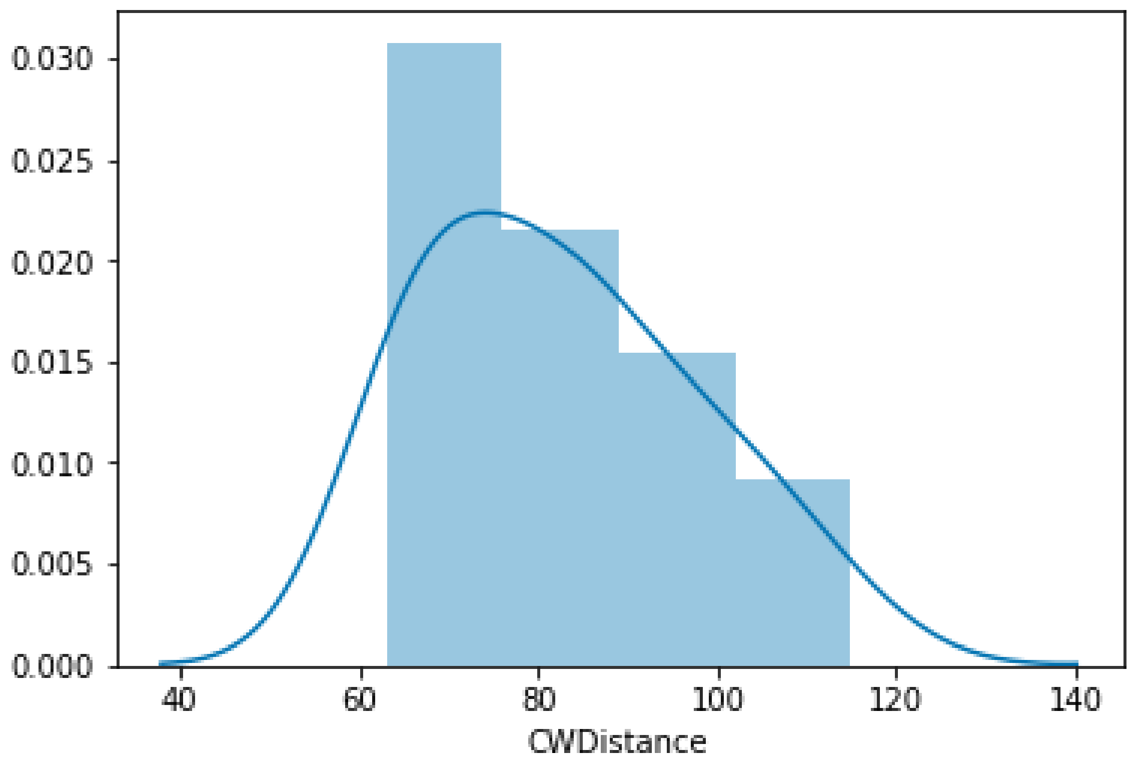


In [33]:
```
# Female Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'F', ["Score"]])

plt.show()
```



Histogram

In [34]:
```
# Distribution Plot (a.k.a. Histogram)
sns.distplot(df.CWDistance)

plt.show()
```



Count Plot

In [35]:

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

plt.xticks(rotation=-45)

plt.show()
```