AI/ML for Climate Workshop

International Livestock Research Institute (ILRI)

---

hide: - toc

📘 NumPy for Climate and Meteorology

🚀 Interactive Learning

launch binder

Click the Binder button above to launch an interactive Jupyter notebook for NumPy and Pandas climate data analysis!

This notebook teaches NumPy skills mapped to climate workflows with fully synthetic arrays.

Requirements

- Python 3.8+
- numpy (and matplotlib for optional plots)

---

0) Setup

```
import numpy as np
np.__version__
```

Output:

```
'2.3.4'
```

```
# Set working directory

import os
os.chdir("c:\\Users\\yonas\\Documents\\ICPAC\\python-climate")

processed_data_dir = os.path.join("data", "processed")
raw_data_dir = os.path.join("data", "raw")
```

---

1. Creating Arrays

Manual construction of arrays

```
a = np.array([0, 1, 2, 3])
```

```
a
```

Output:

```
array([0, 1, 2, 3])
```

```
# returns the shape of the array
a.shape
```

```
# returns no. of dimension
a.ndim
```

Output:

```
1
```

```
# returns the data type
type(a)
```

Output:

```
numpy.ndarray
```

2-Dimensional Arrays

```
# 2 x 3 array
b = np.array([[0, 1, 2], [3, 4, 5]])
```

```
b
```

Output:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
# returns no. of dimension
b.ndim
```

Output:

```
2
```

```
# returns the shape of the array
b.shape
```

Output:

```
(2, 3)
```

```
# returns the size of the first dimension
len(b)
```

Output:

```
2
```

3-Dimensional Arrays

```
c = np.array([[[1,1,2], [2,3,3]], [[1,1,1], [1,1,1]], [[1,1,1], [1,1,1]]])
```

```
c
```

Output:

```
array([[[1, 1, 2],
        [2, 3, 3]],

       [[1, 1, 1],
        [1, 1, 1]],

       [[1, 1, 1],
        [1, 1, 1]]])
```

```
# returns no. of dimension
c.ndim
```

Output:

```
3
```

```
# returns the shape of the array
c.shape
```

Output:

```
(3, 2, 3)
```

```
# returns the size of the first dimension
len(c)
```

Output:

```
3
```

---

2. Functions for creating arrays

Evenly spaced arrays:

```
a = np.arange(10) # 0 .. n-1  (!)
a
```

Output:

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

```
b = np.arange(1, 9, 2) # start, end (exclusive), step
b
```

Output:

```
array([1, 3, 5, 7])
```

```
c = np.linspace(0, 1, 6)   # start, end, num-points
c
```

Output:

```
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

```
d = np.linspace(0, 1, 6, endpoint=False)
d
```

Output:

```
array([0.   , 0.167, 0.333, 0.5  , 0.667, 0.833])
```

Common Arrays

Ones

```
a = np.ones((4, 3))
a
```

Output:

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

Zeros

```
b = np.zeros((2, 2))
b
```

Output:

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

Diagonal Matrix

```
c = np.eye(3)
c
```

Output:

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

```
d = np.diag(np.array([1, 2, 3, 4]))
d
```

Output:

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

```
e = np.diag(d)
e
```

Output:

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

Random numbers

Uniform

```
a = np.random.rand(4)       # uniform in [0, 1]
a
```

> *Output:*

```
array([0.001, 0.817, 0.992, 0.661])
```

Gaussian

```
b = np.random.randn(4)
b
```

> *Output:*

```
array([ 1.077, -0.3  , -0.459,  0.32 ])
```

---

3. Array Examples

```
lat = np.arange(3, 15, 1.0)
lat
```

> *Output:*

```
array([ 3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13., 14.])
```

```
lat.size
```

> *Output:*

```
12
```

```
lon = np.arange(33, 48, 1.0)
lon
```

Output:

```
array([33., 34., 35., 36., 37., 38., 39., 40., 41., 42., 43., 44., 45.,
       46., 47.])
```

```
lon.size
```

Output:

```
15
```

```
time = np.arange(366)
```

```
time.size
```

Output:

```
366
```

```
doy = time
print(doy)
```

Output:

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365]
```

```
# Random Number Generation
rng = np.random.default_rng(0)
# Draw random samples from a normal (Gaussian) distribution.
noise = rng.normal(0.0,0.8,size=(time.size, lat.size, lon.size))
```

```
noise.shape
```

Output:

```
(366, 12, 15)
```

```
noise.dtype
```

Output:

```
dtype('float64')
```

```
# Create Seasonal Cycle
seasonal = 5.0*np.sin(2*np.pi*doy/365.0)[:,None,None]
```

```
# plot seasonal cycle
import matplotlib.pyplot as plt
plt.plot(doy, seasonal[:,0,0])
plt.xlabel('Day of Year')
plt.ylabel('Seasonal Cycle (°C)')
plt.title('Seasonal Cycle of Temperature')
```

Output:

```
Text(0.5, 1.0, 'Seasonal Cycle of Temperature')
<Figure size 640x480 with 1 Axes>
```

```
seasonal.shape
```

Output:

```
(366, 1, 1)
```

```
# Create Latitudinal Gradient
lat_grad = (24.0 - 0.25*lat)[None,:,None]
lat_grad.shape
```

Output:

```
(1, 12, 1)
```

```
# plot the latitudinal gradient
# plt.plot(lat, lat_grad[0,:,0])
# plt.xlabel('Latitude (°)')
# plt.ylabel('Latitudinal Gradient (°C)')
# plt.title('Latitudinal Gradient of Temperature')
```

Output:

```
Text(0.5, 1.0, 'Latitudinal Gradient of Temperature')
<Figure size 640x480 with 1 Axes>
```

```
# Combine all components to create synthetic 2m temperature data
t2m = 24.0 + seasonal + lat_grad + noise
t2m.dtype
```

Output:

```
dtype('float64')
```

```
t2m.shape
```

Output:

```
(366, 12, 15)
```

```
# plot t2m for a specific lat/lon point
plt.plot(doy, t2m[:,5,7])
plt.xlabel('Day of Year')
plt.ylabel('2m Temperature (°C)')
plt.title('Synthetic 2m Temperature Time Series at Lat 8°, Lon 40°')
```

Output:

```
Text(0.5, 1.0, 'Synthetic 2m Temperature Time Series at Lat 8°, Lon 40°')
<Figure size 640x480 with 1 Axes>
```

```
# plot 2m temperature for all latitudes at a specific day
plt.plot(lat, t2m[200,:,7])
plt.xlabel('Latitude (°)')
plt.ylabel('2m Temperature (°C)')
plt.title('Synthetic 2m Temperature on Day 200 at Lon 40°')
```

Output:

```
Text(0.5, 1.0, 'Synthetic 2m Temperature on Day 200 at Lon 40°')
<Figure size 640x480 with 1 Axes>
```

```
# plot 2m temperature for all longitudes at a specific day
plt.plot(lon, t2m[200,5,:])
plt.xlabel('Longitude (°)')
plt.ylabel('2m Temperature (°C)')
plt.title('Synthetic 2m Temperature on Day 200 at Lat 8°')
```

Output:

```
Text(0.5, 1.0, 'Synthetic 2m Temperature on Day 200 at Lat 8°')
<Figure size 640x480 with 1 Axes>
```

```
# plot 2m temperature for all latitudes and all longitudes at a specific day
plt.pcolormesh(t2m[200,:,:])
plt.colorbar(label='2m Temperature (°C)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Synthetic 2m Temperature on Day 200')
plt.show()
```

Output:

```
<Figure size 640x480 with 2 Axes>
```

```
# plot 2m temperature for all latitudes and all longitudes at a specific day
plt.contourf(t2m[200,:,:],  # imshow, contourf
           extent=(lon.min(), lon.max(), lat.min(), lat.max()),
           origin='lower',
           aspect='auto')
plt.colorbar(label='2m Temperature (°C)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Synthetic 2m Temperature on Day 200')
plt.show()
```

Output:

```
C:\Users\yonas\AppData\Local\Temp\ipykernel_141320\973129106.py:2: UserWarning: The fol
  plt.contourf(t2m[200,:,:],  # imshow, contourf

<Figure size 640x480 with 2 Axes>
```

4. Indexing, Slicing & Masking

```
lat_sel = (lat >= 4) & (lat <= 10)
lat_sel
```

Output:

```
array([False,  True,  True,  True,  True,  True,  True,  True, False,
       False, False, False])
```

```
lon_sel = (lon >= 35) & (lon <= 45)
lon_sel
```

Output:

```
array([False, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False, False])
```

```
t2m_sub = t2m[:, lat_sel, :][:, :, lon_sel]
```

```
t2m_sub.shape
```

Output:

```
(366, 7, 11)
```

```
# Hot mask on day 200
hot_mask = t2m_sub[200] > 35.0
hot_mask
```

Output:

```
array([[ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True]])
```

```
hot_mask.shape
```

Output:

```
(7, 11)
```

```
plt.contourf(hot_mask[:,:],  # imshow, contourf
           extent=(lon.min(), lon.max(), lat.min(), lat.max()),
           origin='lower',
           aspect='auto')
plt.colorbar(label='2m Temperature (°C)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Synthetic 2m Temperature on Day 200')
plt.show()
```

Output:

```
C:\Users\yonas\AppData\Local\Temp\ipykernel_141320\415729875.py:1: UserWarning: The fol
  plt.contourf(hot_mask[:,:],  # imshow, contourf

<Figure size 640x480 with 2 Axes>
```

```
# plot the 2tm that satisfy the hot mask
plt.contourf(t2m_sub[200,:,:]*hot_mask,  # imshow, contourf
           extent=(lon.min(), lon.max(), lat.min(), lat.max()),
           origin='lower')
plt.colorbar(label='2m Temperature (°C)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Synthetic 2m Temperature on Day 200')
plt.show()
```

Output:

```
<Figure size 640x480 with 2 Axes>
```

```
# calculate mean climatology
clim_mean = t2m.mean(axis=0)
clim_mean.shape
```

Output:

```
(12, 15)
```

```
# plot climatology

plt.contourf(clim_mean,  # imshow, contourf
           extent=(lon.min(), lon.max(), lat.min(), lat.max()),
              origin='lower')
plt.colorbar(label='2m Temperature (°C)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Climatology of Synthetic 2m Temperature')
plt.show()
```

Output:

```
<Figure size 640x480 with 2 Axes>
```

```
t2m_anom = t2m - clim_mean[None,:,:] #

t2m_anom.shape
```

Output:

```
(366, 12, 15)
```

```
### plot the anomaly for day 200
plt.contourf(t2m_anom[200,:,:],  # imshow, contourf
           extent=(lon.min(), lon.max(), lat.min(), lat.max()),
              origin='lower')
plt.colorbar(label='2m Temperature Anomaly (°C)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Anomaly of Synthetic 2m Temperature on Day 200')
plt.show()
```

Output:

```
<Figure size 640x480 with 2 Axes>
```

---

5) Broadcasting & Vectorization

```
# create u and v wind components
u = rng.normal(0.0,4.0,size=t2m.shape)
v = rng.normal(0.0,4.0,size=t2m.shape)
```

```
# Compute wind speed
wind_speed = np.sqrt(u*u + v*v)
```

```
# plot the wind speed
plt.contourf(wind_speed[200,:,:],  # imshow, contourf
              extent=(lon.min(), lon.max(), lat.min(), lat.max()),
                origin='lower')
plt.colorbar(label='Wind Speed (m/s)')
plt.xlabel('Longitude (°)')
plt.ylabel('Latitude (°)')
plt.title('Wind Speed on Day 200')
plt.show()
```

Output:

```
<Figure size 640x480 with 2 Axes>
```

6. Reductions & Area Weights

```
# Calculate weights for latitude
weights_lat = np.cos(np.deg2rad(lat))
weights_lat.shape
```

Output:

```
(12,)
```

```
# extend the dimenstion and apply weights
w = weights_lat[None,:,None]
w.shape
```

Output:

```
(1, 12, 1)
```

```
# Calculate weighted mean
weighted_mean = np.sum(t2m*w, axis=(1,2)) / np.sum(w, axis=(1,2))
weighted_mean.shape
```

Output:

```
(366,)
```

7. Linear Algebra (EOF/PCA prep)

```
# Calculate Anomalies
anom = t2m - t2m.mean(axis=0)
anom.shape
```

Output:

```
(366, 12, 15)
```

```
# Reshape the data for SVD
T, NY, NX = anom.shape

print(T, NY, NX)
```

Output:

```
366 12 15
```

```
# Reshape the data for SVD
X = anom.reshape(T, NY*NX)
X.shape
```

Output:

```
(366, 180)
```

```
# Perform SVD
U, s, VT = np.linalg.svd(X, full_matrices=False)
```

```
U.shape, s.shape, VT.shape
```

Output:

```
((366, 180), (180,), (180, 180))
```

```
# Compute the first principal component
pc1 = U[:,0]*s[0]
pc1.shape
```

Output:

```
(366,)
```

```
# Compute the second principal component
pc2 = U[:,1]*s[1]
```

```
# Compute the third principal component
pc3 = U[:,2]*s[2]
```

```
# Compute the first EOF
eof1 = VT[0].reshape(NY,NX)
eof1.shape
```

Output:

```
(12, 15)
```

```
# Compute the second EOF
eof2 = VT[1].reshape(NY,NX)

# Compute the third EOF
eof3 = VT[2].reshape(NY,NX)
```

8. I/O with NumPy

```
# Save t2m_anom to a .npy file
np.save("data/processed/t2m_anom.npy", t2m_anom)
np.save("data/processed/pc1.npy", pc1)
```

```
# Save multiple arrays to a .npz file
np.savez("data/processed/cache_demo.npz", pc1=pc1, eof1=eof1)
```

```
#  list contents of the .npz file
loaded = np.load("data/processed/cache_demo.npz")
```

```
# List contents of the .npz file
list(loaded.keys())
```

Output:

```
['pc1', 'eof1']
```

9. Working with Time

```
dates = np.array("2020-01-01", dtype="datetime64[D]") + np.arange(365)
dates
```

Output:

```
array(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
       '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
       '2020-01-09', '2020-01-10', '2020-01-11', '2020-01-12',
       '2020-01-13', '2020-01-14', '2020-01-15', '2020-01-16',
       '2020-01-17', '2020-01-18', '2020-01-19', '2020-01-20',
       '2020-01-21', '2020-01-22', '2020-01-23', '2020-01-24',
       '2020-01-25', '2020-01-26', '2020-01-27', '2020-01-28',
       '2020-01-29', '2020-01-30', '2020-01-31', '2020-02-01',
       '2020-02-02', '2020-02-03', '2020-02-04', '2020-02-05',
       '2020-02-06', '2020-02-07', '2020-02-08', '2020-02-09',
       '2020-02-10', '2020-02-11', '2020-02-12', '2020-02-13',
       '2020-02-14', '2020-02-15', '2020-02-16', '2020-02-17',
       '2020-02-18', '2020-02-19', '2020-02-20', '2020-02-21',
       '2020-02-22', '2020-02-23', '2020-02-24', '2020-02-25',
       '2020-02-26', '2020-02-27', '2020-02-28', '2020-02-29',
       '2020-03-01', '2020-03-02', '2020-03-03', '2020-03-04',
       '2020-03-05', '2020-03-06', '2020-03-07', '2020-03-08',
       '2020-03-09', '2020-03-10', '2020-03-11', '2020-03-12',
       '2020-03-13', '2020-03-14', '2020-03-15', '2020-03-16',
       '2020-03-17', '2020-03-18', '2020-03-19', '2020-03-20',
       '2020-03-21', '2020-03-22', '2020-03-23', '2020-03-24',
       '2020-03-25', '2020-03-26', '2020-03-27', '2020-03-28',
       '2020-03-29', '2020-03-30', '2020-03-31', '2020-04-01',
       '2020-04-02', '2020-04-03', '2020-04-04', '2020-04-05',
       '2020-04-06', '2020-04-07', '2020-04-08', '2020-04-09',
       '2020-04-10', '2020-04-11', '2020-04-12', '2020-04-13',
       '2020-04-14', '2020-04-15', '2020-04-16', '2020-04-17',
       '2020-04-18', '2020-04-19', '2020-04-20', '2020-04-21',
       '2020-04-22', '2020-04-23', '2020-04-24', '2020-04-25',
       '2020-04-26', '2020-04-27', '2020-04-28', '2020-04-29',
       '2020-04-30', '2020-05-01', '2020-05-02', '2020-05-03',
       '2020-05-04', '2020-05-05', '2020-05-06', '2020-05-07',
       '2020-05-08', '2020-05-09', '2020-05-10', '2020-05-11',
       '2020-05-12', '2020-05-13', '2020-05-14', '2020-05-15',
       '2020-05-16', '2020-05-17', '2020-05-18', '2020-05-19',
       '2020-05-20', '2020-05-21', '2020-05-22', '2020-05-23',
       '2020-05-24', '2020-05-25', '2020-05-26', '2020-05-27',
       '2020-05-28', '2020-05-29', '2020-05-30', '2020-05-31',
       '2020-06-01', '2020-06-02', '2020-06-03', '2020-06-04',
       '2020-06-05', '2020-06-06', '2020-06-07', '2020-06-08',
       '2020-06-09', '2020-06-10', '2020-06-11', '2020-06-12',
       '2020-06-13', '2020-06-14', '2020-06-15', '2020-06-16',
       '2020-06-17', '2020-06-18', '2020-06-19', '2020-06-20',
       '2020-06-21', '2020-06-22', '2020-06-23', '2020-06-24',
       '2020-06-25', '2020-06-26', '2020-06-27', '2020-06-28',
       '2020-06-29', '2020-06-30', '2020-07-01', '2020-07-02',
       '2020-07-03', '2020-07-04', '2020-07-05', '2020-07-06',
       '2020-07-07', '2020-07-08', '2020-07-09', '2020-07-10',
       '2020-07-11', '2020-07-12', '2020-07-13', '2020-07-14',
       '2020-07-15', '2020-07-16', '2020-07-17', '2020-07-18',
       '2020-07-19', '2020-07-20', '2020-07-21', '2020-07-22',
       '2020-07-23', '2020-07-24', '2020-07-25', '2020-07-26',
       '2020-07-27', '2020-07-28', '2020-07-29', '2020-07-30',
       '2020-07-31', '2020-08-01', '2020-08-02', '2020-08-03',
       '2020-08-04', '2020-08-05', '2020-08-06', '2020-08-07',
       '2020-08-08', '2020-08-09', '2020-08-10', '2020-08-11',
       '2020-08-12', '2020-08-13', '2020-08-14', '2020-08-15',
       '2020-08-16', '2020-08-17', '2020-08-18', '2020-08-19',
       '2020-08-20', '2020-08-21', '2020-08-22', '2020-08-23',
       '2020-08-24', '2020-08-25', '2020-08-26', '2020-08-27',
       '2020-08-28', '2020-08-29', '2020-08-30', '2020-08-31',
       '2020-09-01', '2020-09-02', '2020-09-03', '2020-09-04',
       '2020-09-05', '2020-09-06', '2020-09-07', '2020-09-08',
       '2020-09-09', '2020-09-10', '2020-09-11', '2020-09-12',
       '2020-09-13', '2020-09-14', '2020-09-15', '2020-09-16',
       '2020-09-17', '2020-09-18', '2020-09-19', '2020-09-20',
       '2020-09-21', '2020-09-22', '2020-09-23', '2020-09-24',
       '2020-09-25', '2020-09-26', '2020-09-27', '2020-09-28',
       '2020-09-29', '2020-09-30', '2020-10-01', '2020-10-02',
       '2020-10-03', '2020-10-04', '2020-10-05', '2020-10-06',
       '2020-10-07', '2020-10-08', '2020-10-09', '2020-10-10',
       '2020-10-11', '2020-10-12', '2020-10-13', '2020-10-14',
       '2020-10-15', '2020-10-16', '2020-10-17', '2020-10-18',
       '2020-10-19', '2020-10-20', '2020-10-21', '2020-10-22',
       '2020-10-23', '2020-10-24', '2020-10-25', '2020-10-26',
       '2020-10-27', '2020-10-28', '2020-10-29', '2020-10-30',
       '2020-10-31', '2020-11-01', '2020-11-02', '2020-11-03',
       '2020-11-04', '2020-11-05', '2020-11-06', '2020-11-07',
       '2020-11-08', '2020-11-09', '2020-11-10', '2020-11-11',
       '2020-11-12', '2020-11-13', '2020-11-14', '2020-11-15',
       '2020-11-16', '2020-11-17', '2020-11-18', '2020-11-19',
       '2020-11-20', '2020-11-21', '2020-11-22', '2020-11-23',
       '2020-11-24', '2020-11-25', '2020-11-26', '2020-11-27',
       '2020-11-28', '2020-11-29', '2020-11-30', '2020-12-01',
       '2020-12-02', '2020-12-03', '2020-12-04', '2020-12-05',
       '2020-12-06', '2020-12-07', '2020-12-08', '2020-12-09',
       '2020-12-10', '2020-12-11', '2020-12-12', '2020-12-13',
       '2020-12-14', '2020-12-15', '2020-12-16', '2020-12-17',
       '2020-12-18', '2020-12-19', '2020-12-20', '2020-12-21',
       '2020-12-22', '2020-12-23', '2020-12-24', '2020-12-25',
       '2020-12-26', '2020-12-27', '2020-12-28', '2020-12-29',
       '2020-12-30'], dtype='datetime64[D]')
```

```
years = dates.astype("datetime64[Y]").astype(int) + 1970
years
```

Output:

```
array([2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
       2020, 2020])
```

10. Numpy Help

On the web: https://numpy.org/doc/

Asking For Help

```
np.add?
```

```
np.array?
```

```
np.arr*?
np.con*?
```

Exercise 1

1) Create a simple one, two, and three dimensional array.
2) Use the functions len(), numpy.shape() on these arrays.
3) Experiment with arange, linspace, ones, zeros, eye and diag.
4) Create different kinds of arrays with random numbers.
5) Look at the function `np.empty`. What does it do? When might this be useful?