Python Libraries for ML @ GCE Raipur

Instructor: Santosh Chapaneri

Mar 2021

NumPy

SciPy

Pandas

Matplotlib

Covid-19 data visualization

NumPy

- Python lists are great. They can store strings, integers, or mixtures.
- NumPy arrays though are **multi-dimensional** and most **engineering** python libraries use them instead.
- They store the **same type of data** in each element and **cannot change size**.

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

x = np.zeros(5)
print(x)
```

```
[0. 0. 0. 0. 0.]
```

In [2]:
```
x = np.zeros( (5,2) )
print(x)
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
```

In [3]:
```
print(np.arange(3, 10))       # Does not include end point
print(np.linspace(0, 1, 25))  # Includes end point
```

```
[3 4 5 6 7 8 9]
[0.         0.04166667 0.08333333 0.125      0.16666667 0.20833333
 0.25       0.29166667 0.33333333 0.375      0.41666667 0.45833333
 0.5        0.54166667 0.58333333 0.625      0.66666667 0.70833333
 0.75       0.79166667 0.83333333 0.875      0.91666667 0.95833333
 1.        ]
```

In [4]:
```
print(np.logspace(0, 1, 25)) # Log spaced numbers
```

```
[ 1.          1.10069417  1.21152766  1.33352143  1.46779927  1.6155981
  1.77827941  1.95734178  2.15443469  2.37137371  2.61015722  2.87298483
  3.16227766  3.48070059  3.83118685  4.21696503  4.64158883  5.10896977
  5.62341325  6.18965819  6.81292069  7.49894209  8.25404185  9.08517576
 10.        ]
```

Numpy Arithmetic Operations

In [5]:
```
# Trivial math
x = np.array( [[50.0, 60.0], [70.0, 80.0]] )
y = np.array( [[10.0, 20.0], [30.0, 40.0]] )

print( np.add(x, y) )
print( np.subtract(x, y) )
print( np.multiply(x, y) )
print( np.divide(x, y) )
```

```
[[ 60.  80.]
 [100. 120.]]
[[40. 40.]
 [40. 40.]]
[[ 500. 1200.]
 [2100. 3200.]]
[[5.         3.        ]
 [2.33333333 2.        ]]
```

In [6]:
```
# Element-wise sqrt of matrix
print( np.sqrt(x) )
```

```
[[7.07106781 7.74596669]
 [8.36660027 8.94427191]]
```

In [7]:
```
# Dot product
v = np.array([9.0, 10.0])
w = np.array([11.0, 12.0])

print( np.dot(v, w) ) # 9 x 11 + 10 x 12
```

```
219.0
```

In [8]:
```
# Dot product of matrices
print(x)
print(y)
print( np.dot(x, y) ) # 50 x 10 + 60 x 30 = 2300, etc.
```

```
[[50. 60.]
 [70. 80.]]
[[10. 20.]
 [30. 40.]]
[[2300. 3400.]
 [3100. 4600.]]
```

In [9]:
```
# Sum along row or column
print(x)
print( 'Sum along columns:', np.sum(x, axis = 0) )
print( 'Sum along rows:', np.sum(x, axis = 1) )
```

```
[[50. 60.]
 [70. 80.]]
Sum along columns: [120. 140.]
Sum along rows: [110. 150.]
```

In [10]:
```
# Matrix reshaping
a = np.arange(40).reshape(5, 8)
print(a)
```

```
[[ 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]]
```

```
In [11]: # 3 x 2 array, all elements having same value
         c = np.full( (3, 2), 8)
         print(c)
```

```
[[8 8]
 [8 8]
 [8 8]]
```

```
In [12]: # Identity matrix
         d = np.eye(3)
         print(d)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
In [13]: # Matrix of random numbers
         e = np.random.random( (4, 3) )
         print(e)
```

```
[[0.35316972 0.53296443 0.68043865]
 [0.41636044 0.40335758 0.42739342]
 [0.60076165 0.60779687 0.54081406]
 [0.00810099 0.16780141 0.47422357]]
```

Code Vectorization

```
In [14]: # Apply cos on each element of the list
         from math import pi
         x = np.linspace(-pi, pi, 4)
         print( np.cos(x) )
```

```
[-1.   0.5  0.5 -1. ]
```

```
In [15]: # Apply sqrt on each element of the list
         z = [i**2 for i in range(1,11)]
         print(z)
         print( np.sqrt(z) )
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

NumPy Speed

- Why people love NumPy?

```
In [16]: %%timeit xvals = range(1000000)
         [xval**2 for xval in xvals]
```

```
1 loop, best of 5: 295 ms per loop
```

```
In [17]: %%timeit a = np.arange(1000000)
         a**2
```

```
100 loops, best of 5: 2.08 ms per loop
```

```
In [18]: import math
```

```
In [19]: %%timeit xvals = range(1000000)
         [math.sin(xval) for xval in xvals]
```

```
1 loop, best of 5: 214 ms per loop
```

```
In [20]: %%timeit a = np.arange(100000)
         np.sin(a)
```

```
100 loops, best of 5: 2.28 ms per loop
```

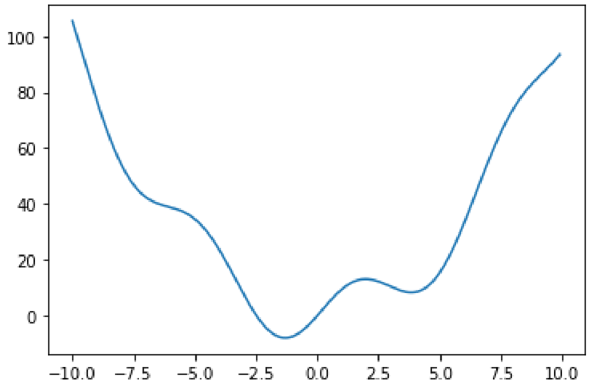
SciPy

The SciPy framework builds on top of the NumPy framework for multidimensional arrays, and provides a large number of higher-level scientific algorithms.

- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)

Optimization

In [21]:
```
# Optimization example
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt

# Function to be minimized wrt x
def f(x):
    return x**2 + 10*np.sin(x)

x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
plt.show()
```



- This function has a global minimum around -1.3 and a local minimum around 3.8.
- Searching for minimum can be done with scipy.optimize.minimize(); given a starting point x0 , it returns the location of the minimum that it has found

In [22]:
```
from scipy.optimize import minimize
result = minimize(f, x0 = 1.2)

print(result)       # Global minimum
print(f(result.x)) # Value at global minimum
```

```
      fun: -7.945823375615282
 hess_inv: array([[0.08577271]])
      jac: array([4.17232513e-07])
  message: 'Optimization terminated successfully.'
     nfev: 27
      nit: 5
     njev: 9
   status: 0
  success: True
        x: array([-1.30643999])
[-7.94582338]
```

Linear Regression using SciPy

In [23]:
```
from scipy import stats
import matplotlib.pyplot as plt

# Linear regression example
# This is a very simple example of
# linear regression using SciPy's stats.linregress

x = np.random.random(10)
y = 1.6*x + np.random.random(10)
```

In [24]:
```
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
print("slope: %f    intercept: %f" % (slope, intercept))
```

```
slope: 1.461813    intercept: 0.589843
```

In [25]:
```
plt.plot(x, y, 'o', label='original data')
plt.plot(x, intercept + slope*x, 'r', label='fitted line')
plt.legend()
plt.show()
```

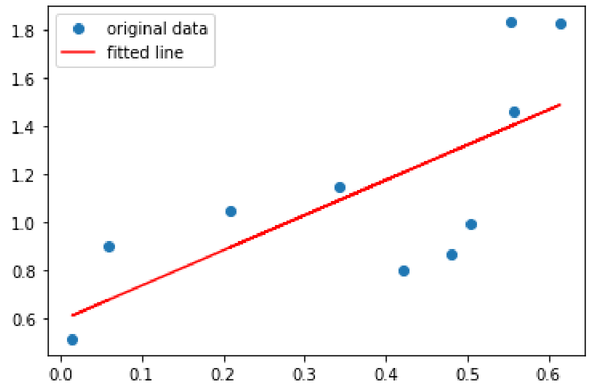


Image Transformations

In [26]:
```
from scipy import misc
from scipy import ndimage
import matplotlib.pyplot as plt

# Load an image
face = misc.face(gray=True)

# Apply a variety of transformations
shifted_face = ndimage.shift(face, (50, 50))
rotated_face = ndimage.rotate(face, 30)
zoomed_face = ndimage.zoom(face, 2)

plt.figure(figsize=(15, 3))
plt.subplot(131) # 1 x 3, 1st plot
plt.imshow(shifted_face, cmap=plt.cm.gray)
plt.axis('off')

plt.subplot(132) # 1 x 3, 2nd plot
plt.imshow(rotated_face, cmap=plt.cm.gray)
plt.axis('off')

plt.subplot(133) # 1 x 3, 3rd plot
plt.imshow(zoomed_face, cmap=plt.cm.gray)
plt.axis('off')

plt.subplots_adjust(wspace=.05, left=.01, bottom=.01, right=.99, top=.99)
plt.show()
```

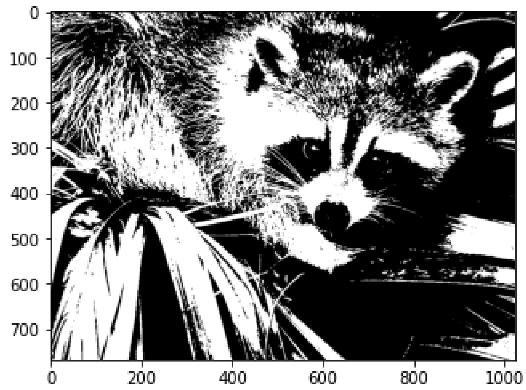
Image Manipulation

In [27]:
```
# Display as grayscale
plt.imshow(face, cmap=plt.cm.gray)
```

Out[27]: <matplotlib.image.AxesImage at 0x7f72ddc4aa50>

In [28]:
```
# Thresholding
plt.imshow(face > 128, cmap=plt.cm.gray)
```

Out[28]: <matplotlib.image.AxesImage at 0x7f72dc3b5dd0>



In [29]:
```
# Contrast change
plt.imshow(255*(face/255)**1.5, cmap=plt.cm.gray)
```

Out[29]: <matplotlib.image.AxesImage at 0x7f72dc32c490>

Interpolation

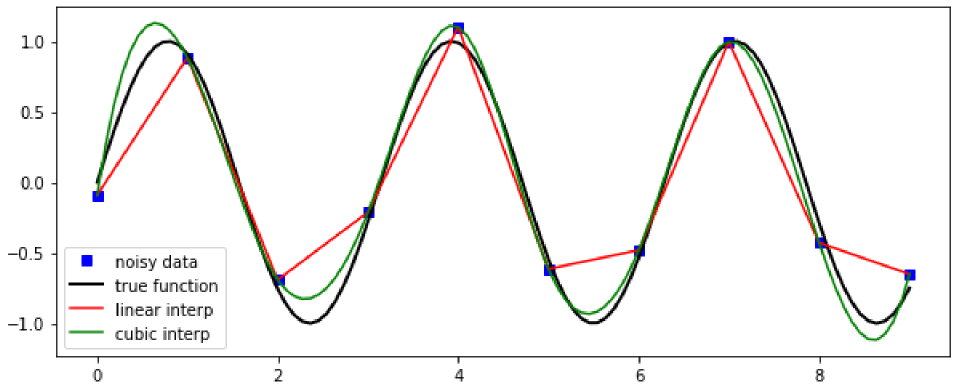
- The `interp1d` function, when given arrays describing X and Y data, returns an object that behaves like a function that can be called for an arbitrary value of x (in the range covered by X), and it returns the corresponding interpolated y value

In [30]:
```
from scipy.interpolate import *

# y = sin(2x)
def f(x):
    return np.sin(2*x)
```

In [31]:
```
n = np.arange(0, 10)
x = np.linspace(0, 9, 100)

# Simulate with noise
y_meas = f(n) + 0.1 * np.random.randn(len(n))

# Ground truth
y_real = f(x)

linear_interpolation = interp1d(n, y_meas)
y_interp1 = linear_interpolation(x)

cubic_interpolation = interp1d(n, y_meas, kind='cubic')
y_interp2 = cubic_interpolation(x)
```

In [32]:
```
fig, ax = plt.subplots(figsize=(10,4))
ax.plot(n, y_meas, 'bs', label='noisy data')
ax.plot(x, y_real, 'k', lw=2, label='true function')
ax.plot(x, y_interp1, 'r', label='linear interp')
ax.plot(x, y_interp2, 'g', label='cubic interp')
ax.legend(loc=3)
```

Out[32]: <matplotlib.legend.Legend at 0x7f72dc2ed9d0>



Integration with SciPy

- Compute $\int_0^1 f(x)dx$
- where $f(x) = \exp(-x^2)$

In [33]:
```
import scipy.integrate as intg
from numpy import exp

f = lambda x : exp(-x**2)
res = intg.quad(f, 0, 1)
print(res)
```

```
(0.7468241328124271, 8.291413475940725e-15)
```

- The quad function returns the two values, in which the first number is the value of integral and the second value is the estimate of the absolute error in the value of integral.

- **Double Integral**

The general form of dblquad is `scipy.integrate.dblquad(func, a, b, gfun, hfun)` .

Here, func is the name of the function to be integrated, ‘a’ and ‘b’ are the lower and upper limits of the x variable, respectively, while gfun and hfun are the names of the functions that define the lower and upper limits of the y variable.

- Compute $\int_0^{1/2} \int_0^{\sqrt{1-4y^2}} 16xy \, dy \, dx$

In [34]:
```
import scipy.integrate as intg
from numpy import exp
from math import sqrt

f = lambda x, y : 16*x*y
g = lambda x : 0
h = lambda y : sqrt(1-4*y**2)

res = intg.dblquad(f, 0, 0.5, g, h)
print(res)
```

```
(0.5, 1.7092350012594845e-14)
```

Pandas

Dataframes

- According to Pandas documentation: *Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels.*
- In human terms, this means that a **dataframe has rows and columns**, can **change size**, and possibly **has mixed data types**.

Peek at the DataFrame contents

- `df.info()` # index & data types
- `df.head(i)` # get first i rows
- `df.tail(i)` # get last i rows
- `df.describe()` # summary stats cols

A very powerful feature in Pandas is **groupby**.

- This function allows us to **group together rows that have the same value in a particular column**.
- Then, we can aggregate this group-by object to compute statistics in each group.

MovieLens 100k movie rating data:

- main page: [http://grouplens.org/datasets/movielens/ (http://grouplens.org/datasets/movielens/)](http://grouplens.org/datasets/movielens/)
- 100,000 ratings from 1000 users on 1700 movies

In [36]:
```
import pandas as pd

users = pd.read_csv('u.user', sep='|', index_col='user_id')
```

In [37]:
```
# print the first 5 rows
users.head()
```

Out[37]:

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

In [38]:
```
# print the first 10 rows
users.head(10)
```

Out[38]:

user_id	age	gender	occupation	zip_code
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
6	42	M	executive	98101
7	57	M	administrator	91344
8	36	M	administrator	05201
9	29	M	student	01002
10	53	M	lawyer	90703

In [39]:
```
# print the last 5 rows
users.tail()
```

Out[39]:

user_id	age	gender	occupation	zip_code
939	26	F	student	33319
940	32	M	administrator	02215
941	20	M	student	97229
942	48	F	librarian	78209
943	22	M	student	77841

In [40]:
```
# column names
users.columns
```

Out[40]:
```
Index(['age', 'gender', 'occupation', 'zip_code'], dtype='object')
```

In [41]:
```
# data types of each column
users.dtypes
```

Out[41]:
```
age            int64
gender        object
occupation    object
zip_code      object
dtype: object
```

In [42]:
```
# number of rows and columns
users.shape
```

Out[42]:
```
(943, 4)
```

In [43]:
```
# select one column using the DataFrame attribute
users.gender
```

Out[43]:
```
user_id
1      M
2      F
3      M
4      M
5      F
      ..
939    F
940    M
941    M
942    F
943    M
Name: gender, Length: 943, dtype: object
```

In [44]:
```
# summarize (describe) the DataFrame
users.describe()                    # describe all numeric columns
```

Out[44]:

	age
count	943.000000
mean	34.051962
std	12.192740
min	7.000000
25%	25.000000
50%	31.000000
75%	43.000000
max	73.000000

In [45]:
```
users.describe(include=['object'])  # describe all object columns
```

Out[45]:

	gender	occupation	zip_code
count	943	943	943
unique	2	21	795
top	M	student	55414
freq	670	196	9

In [46]:
```
users.describe(include='all')       # describe all columns
```

Out[46]:

	age	gender	occupation	zip_code
count	943.000000	943	943	943
unique	NaN	2	21	795
top	NaN	M	student	55414
freq	NaN	670	196	9
mean	34.051962	NaN	NaN	NaN
std	12.192740	NaN	NaN	NaN
min	7.000000	NaN	NaN	NaN
25%	25.000000	NaN	NaN	NaN
50%	31.000000	NaN	NaN	NaN
75%	43.000000	NaN	NaN	NaN
max	73.000000	NaN	NaN	NaN

In [47]:
```
# count the number of occurrences of each value
users.gender.value_counts()     # most useful for categorical variables
```

Out[47]:
```
M    670
F    273
Name: gender, dtype: int64
```

In [48]:
```
users.age.value_counts()        # can also be used with numeric variables
```

Out[48]:
```
30    39
25    38
22    37
28    36
27    35
      ..
11     1
10     1
73     1
66     1
7      1
Name: age, Length: 61, dtype: int64
```

In [49]:
```
# Boolean filtering: only show users with age < 20
users[users.age < 20]
```

Out[49]:

user_id	age	gender	occupation	zip_code
30	7	M	student	55436
36	19	F	student	93117
52	18	F	student	55105
57	16	M	none	84010
67	17	M	student	60402
...	...	...	...	...
872	19	F	student	74078
880	13	M	student	83702
887	14	F	student	27249
904	17	F	student	61073
925	18	F	salesman	49036

77 rows × 4 columns

In [50]:
```
# for each occupation in 'users', count the number of occurrences
users.occupation.value_counts()
```

Out[50]:
```
student          196
other            105
educator          95
administrator     79
engineer          67
programmer        66
librarian         51
writer            45
executive         32
scientist         31
artist            28
technician        27
marketing         26
entertainment     18
healthcare        16
retired           14
lawyer            12
salesman          12
none               9
homemaker          7
doctor             7
Name: occupation, dtype: int64
```

In [51]:
```
# for each occupation, calculate the mean age
users.groupby('occupation').age.mean()
```

Out[51]:
```
occupation
administrator    38.746835
artist           31.392857
doctor           43.571429
educator         42.010526
engineer         36.388060
entertainment    29.222222
executive        38.718750
healthcare       41.562500
homemaker        32.571429
lawyer           36.750000
librarian        40.000000
marketing        37.615385
none             26.555556
other            34.523810
programmer       33.121212
retired          63.071429
salesman         35.666667
scientist        35.548387
student          22.081633
technician       33.148148
writer           36.311111
Name: age, dtype: float64
```

In [52]:
```
# for each occupation, calculate the minimum and maximum ages
users.groupby('occupation').age.agg(['min', 'max'])
```

Out[52]:

	min	max
occupation		
administrator	21	70
artist	19	48
doctor	28	64
educator	23	63
engineer	22	70
entertainment	15	50
executive	22	69
healthcare	22	62
homemaker	20	50
lawyer	21	53
librarian	23	69
marketing	24	55
none	11	55
other	13	64
programmer	20	63
retired	51	73
salesman	18	66
scientist	23	55
student	7	42
technician	21	55
writer	18	60

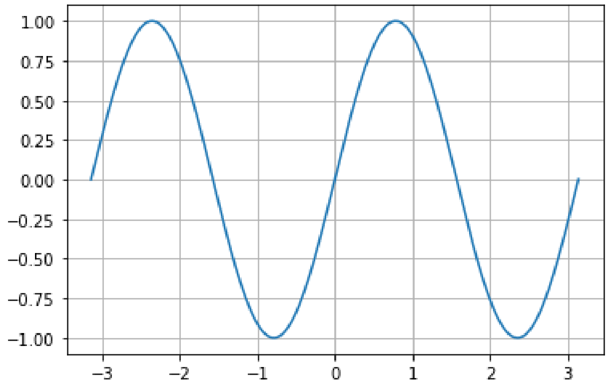
In [53]:
```
# for each combination of occupation and gender, calculate the mean age
users.groupby(['occupation', 'gender']).age.mean()
```

Out[53]:
```
occupation     gender
administrator  F         40.638889
               M         37.162791
artist         F         30.307692
               M         32.333333
doctor         M         43.571429
educator       F         39.115385
               M         43.101449
engineer       F         29.500000
               M         36.600000
entertainment  F         31.000000
               M         29.000000
executive      F         44.000000
               M         38.172414
healthcare     F         39.818182
               M         45.400000
homemaker      F         34.166667
               M         23.000000
lawyer         F         39.500000
               M         36.200000
librarian      F         40.000000
               M         40.000000
marketing      F         37.200000
               M         37.875000
none           F         36.500000
               M         18.600000
other          F         35.472222
               M         34.028986
programmer     F         32.166667
               M         33.216667
retired        F         70.000000
               M         62.538462
salesman       F         27.000000
               M         38.555556
scientist      F         28.333333
               M         36.321429
student        F         20.750000
               M         22.669118
technician     F         38.000000
               M         32.961538
writer         F         37.631579
               M         35.346154
Name: age, dtype: float64
```

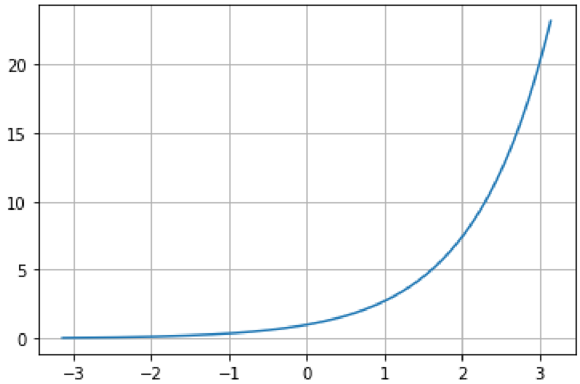
Matplotlib - Plotting

Basic plotting

In [54]:
```
import matplotlib.pyplot as plt
import numpy as np
from math import pi

x = np.linspace(-pi, pi, 200)
y = np.sin(2*x)
plt.plot(x, y)
plt.grid()
plt.show()
```

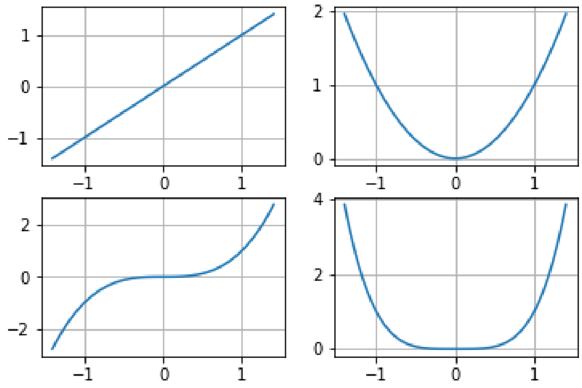


In [55]:
```
y = np.exp(x) # also, try with exp(-x)
plt.plot(x, y)
plt.grid()
plt.show()
```



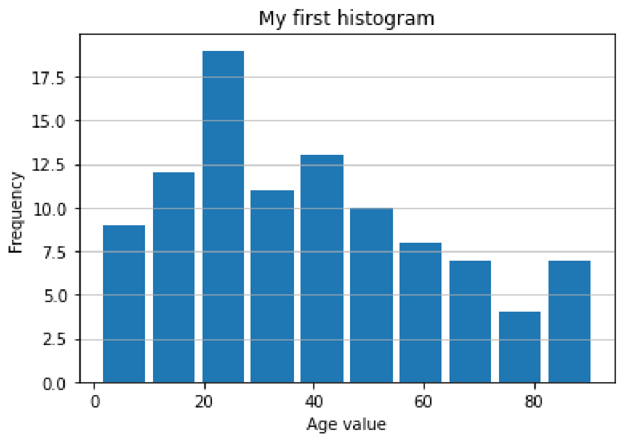
Subplots

In [56]:
```
x = np.linspace(-1.4, 1.4, 50)
plt.subplot(2, 2, 1)  # 2 x 2, top left
plt.plot(x, x)
plt.grid()
plt.subplot(2, 2, 2)  # 2 x 2, top right
plt.plot(x, x**2)
plt.grid()
plt.subplot(2, 2, 3)  # 2 x 2, bottow left
plt.plot(x, x**3)
plt.grid()
plt.subplot(2, 2, 4)  # 2 x 2, bottom right
plt.plot(x, x**4)
plt.grid()
plt.show()
```



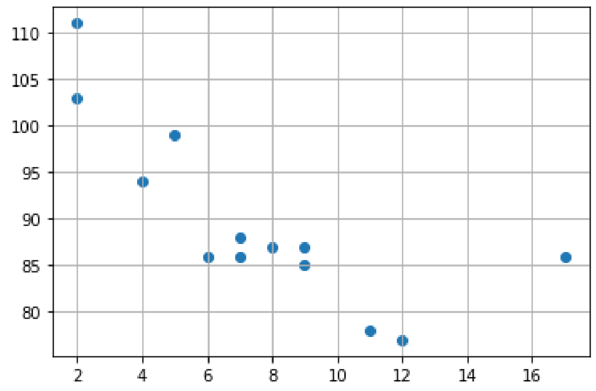
Histograms

In [57]:
```
# Histogram of Age
x = [1,1,2,3,3,5,7,8,9,10,10,11,11,13,13,15,16,17,18,18,18,19,20,21,21,23,24,
     24,25,25,25,25,26,26,26,27,27,27,27,27,29,30,30,31,33,34,34,34,35,36,
     36,37,37,38,38,39,40,41,41,42,43,44,45,45,46,47,48,48,49,50,51,52,53,54,
     55,55,56,57,58,60,61,63,64,65,66,68,70,71,72,74,75,77,81,83,84,87,89,
     90,90,91]

# rwidth: bar's relative width, default 1
plt.hist(x, bins=10, rwidth = 0.85)

# Only the y grid
plt.grid(axis='y', alpha=0.75)

plt.xlabel('Age value')
plt.ylabel('Frequency')
plt.title('My first histogram')
plt.show()
```



Scatter Plot

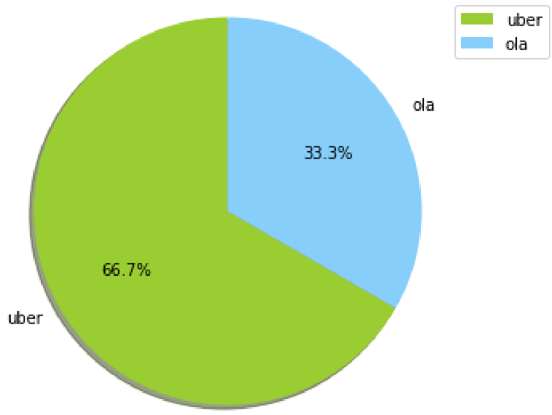
In [58]:
```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
# Note: Both x and y must have same length

plt.scatter(x, y)
plt.grid()
plt.show()
```



Pie Chart

- Pie chart shows the size of items in a data series, proportional to the sum of the items.
- The data points in a pie chart are shown as a percentage of the whole pie.

In [59]:
```
sizes = [10, 5]
mylabels = ['uber','ola']

plt.pie(
    sizes,
    labels = mylabels,
    shadow = True,
    colors = ['yellowgreen', 'lightskyblue'],
    startangle = 90,    # rotate conter-clockwise by 90 degrees
    autopct = '%1.1f%%',# display fraction as percentage
    )
plt.legend(fancybox=True)
plt.axis('equal')     # plot pyplot as circle
plt.tight_layout()
plt.show()
```



COVID-19 Data Visualization

Data taken from [https://github.com/CSSEGISandData/COVID-19 (https://github.com/CSSEGISandData/COVID-19)](https://github.com/CSSEGISandData/COVID-19)

In [60]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
import math
import time
import datetime
%matplotlib inline
```

Pre-processing of COVID-19 data

In [61]:
```
confirmed_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv')

deaths_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_global.csv')

recoveries_df = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_recovered_global.csv')

latest_data = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_daily_reports/03-13-2021.csv')
```

In [62]:
```
# Head of confirmed df
confirmed_df.head()
```

Out[62]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	0	0
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	0	0
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	0	0
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	0	0
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	0	0

5 rows × 421 columns

In [63]:
```
cols = confirmed_df.keys()
confirmed = confirmed_df.loc[:, cols[4]:cols[-1]]
deaths = deaths_df.loc[:, cols[4]:cols[-1]]
recoveries = recoveries_df.loc[:, cols[4]:cols[-1]]
```

In [64]:
```
dates = confirmed.keys()
world_cases = []
total_deaths = []
mortality_rate = []
recovery_rate = []
total_recovered = []
total_active = []

china_cases = []
italy_cases = []
us_cases = []
spain_cases = []
france_cases = []
germany_cases = []
uk_cases = []
india_cases = []

china_deaths = []
italy_deaths = []
us_deaths = []
spain_deaths = []
france_deaths = []
germany_deaths = []
uk_deaths = []
india_deaths = []

china_recoveries = []
italy_recoveries = []
us_recoveries = []
spain_recoveries = []
france_recoveries = []
germany_recoveries = []
uk_recoveries = []
india_recoveries = []

for i in dates:
    confirmed_sum = confirmed[i].sum()
    death_sum = deaths[i].sum()
    recovered_sum = recoveries[i].sum()

    # confirmed, deaths, recovered, and active
    world_cases.append(confirmed_sum)
    total_deaths.append(death_sum)
    total_recovered.append(recovered_sum)
    total_active.append(confirmed_sum - death_sum - recovered_sum)

    # calculate rates
    mortality_rate.append(death_sum/confirmed_sum)
    recovery_rate.append(recovered_sum/confirmed_sum)

    # case studies
    china_cases.append(confirmed_df[confirmed_df['Country/Region']=='China'][i].sum())
    italy_cases.append(confirmed_df[confirmed_df['Country/Region']=='Italy'][i].sum())
    us_cases.append(confirmed_df[confirmed_df['Country/Region']=='US'][i].sum())
    spain_cases.append(confirmed_df[confirmed_df['Country/Region']=='Spain'][i].sum())
    france_cases.append(confirmed_df[confirmed_df['Country/Region']=='France'][i].sum())
    germany_cases.append(confirmed_df[confirmed_df['Country/Region']=='Germany'][i].sum())
    uk_cases.append(confirmed_df[confirmed_df['Country/Region']=='United Kingdom'][i].sum())
    india_cases.append(confirmed_df[confirmed_df['Country/Region']=='India'][i].sum())

    china_deaths.append(deaths_df[deaths_df['Country/Region']=='China'][i].sum())
    italy_deaths.append(deaths_df[deaths_df['Country/Region']=='Italy'][i].sum())
    us_deaths.append(deaths_df[deaths_df['Country/Region']=='US'][i].sum())
    spain_deaths.append(deaths_df[deaths_df['Country/Region']=='Spain'][i].sum())
    france_deaths.append(deaths_df[deaths_df['Country/Region']=='France'][i].sum())
    germany_deaths.append(deaths_df[deaths_df['Country/Region']=='Germany'][i].sum())
    uk_deaths.append(deaths_df[deaths_df['Country/Region']=='United Kingdom'][i].sum())
    india_deaths.append(deaths_df[deaths_df['Country/Region']=='India'][i].sum())

    china_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='China'][i].sum())
    italy_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='Italy'][i].sum())
    us_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='US'][i].sum())
    spain_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='Spain'][i].sum())
    france_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='France'][i].sum())
    germany_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='Germany'][i].sum())
    uk_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='United Kingdom'][i].sum
```

```
())
    india_recoveries.append(recoveries_df[recoveries_df['Country/Region']=='India'][i].sum())
```

In [65]:
```python
def daily_increase(data):
    d = []
    for i in range(len(data)):
        if i == 0:
            d.append(data[0])
        else:
            d.append(data[i] - data[i-1])
    return d

# confirmed cases
world_daily_increase = daily_increase(world_cases)
china_daily_increase = daily_increase(china_cases)
italy_daily_increase = daily_increase(italy_cases)
us_daily_increase = daily_increase(us_cases)
spain_daily_increase = daily_increase(spain_cases)
france_daily_increase = daily_increase(france_cases)
germany_daily_increase = daily_increase(germany_cases)
uk_daily_increase = daily_increase(uk_cases)
india_daily_increase = daily_increase(india_cases)

# deaths
world_daily_death = daily_increase(total_deaths)
china_daily_death = daily_increase(china_deaths)
italy_daily_death = daily_increase(italy_deaths)
us_daily_death = daily_increase(us_deaths)
spain_daily_death = daily_increase(spain_deaths)
france_daily_death = daily_increase(france_deaths)
germany_daily_death = daily_increase(germany_deaths)
uk_daily_death = daily_increase(uk_deaths)
india_daily_death = daily_increase(india_deaths)

# recoveries
world_daily_recovery = daily_increase(total_recovered)
china_daily_recovery = daily_increase(china_recoveries)
italy_daily_recovery = daily_increase(italy_recoveries)
us_daily_recovery = daily_increase(us_recoveries)
spain_daily_recovery = daily_increase(spain_recoveries)
france_daily_recovery = daily_increase(france_recoveries)
germany_daily_recovery = daily_increase(germany_recoveries)
uk_daily_recovery = daily_increase(uk_recoveries)
india_daily_recovery = daily_increase(india_recoveries)
```

22 January 2020

- The WHO mission to Wuhan issued a statement saying that evidence suggested human-to-human transmission in Wuhan but that more investigation was needed to understand the full extent of transmission.

Source: [https://www.who.int/news-room/detail/29-06-2020-covidtimeline (https://www.who.int/news-room/detail/29-06-2020-covidtimeline)](https://www.who.int/news-room/detail/29-06-2020-covidtimeline)

In [66]:
```python
days_since_1_22 = np.array([i for i in range(len(dates))]).reshape(-1, 1)
world_cases = np.array(world_cases).reshape(-1, 1)
total_deaths = np.array(total_deaths).reshape(-1, 1)
total_recovered = np.array(total_recovered).reshape(-1, 1)
adjusted_dates = np.array([i for i in range(len(dates))])
adjusted_dates = adjusted_dates.reshape(1, -1)[0]
```

In [67]:
```python
len(adjusted_dates)
```

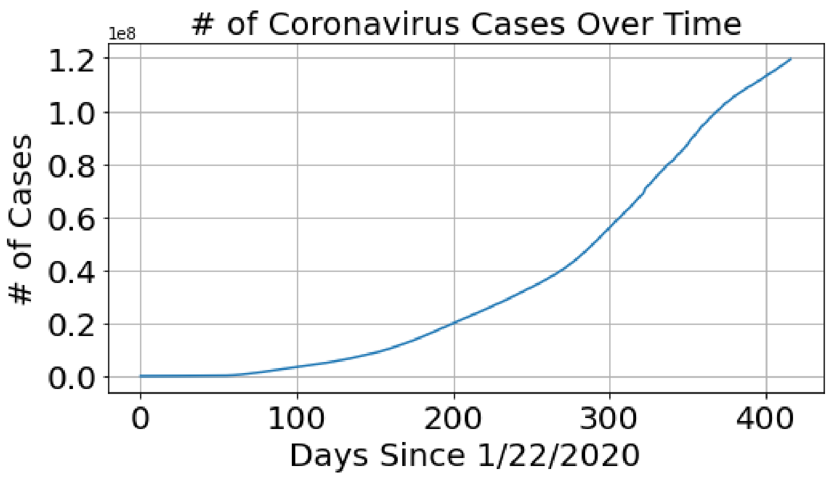
Out[67]: 417

In [68]:
```
adjusted_dates
```

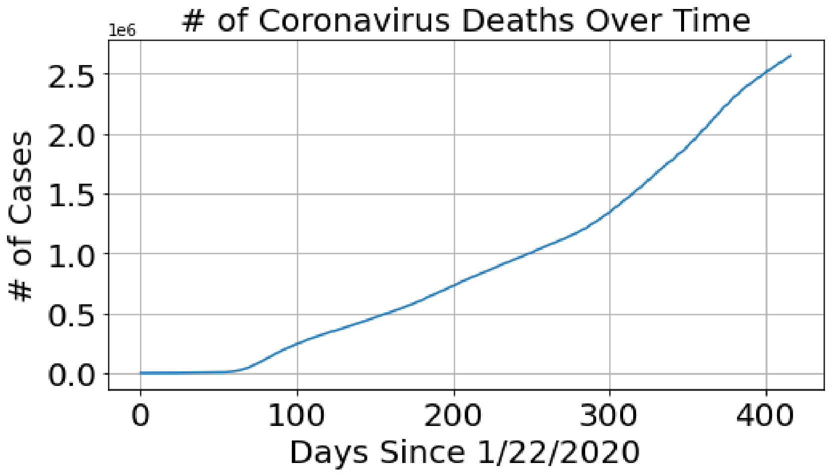
Out[68]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        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, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
       377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
       390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
       403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
       416])
```

Visualization of COVID-19 data

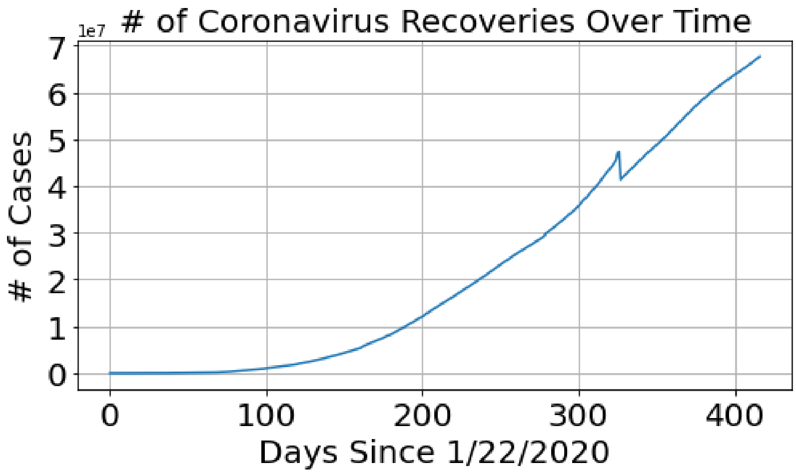
In [69]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, world_cases)
plt.title('# of Coronavirus Cases Over Time', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```



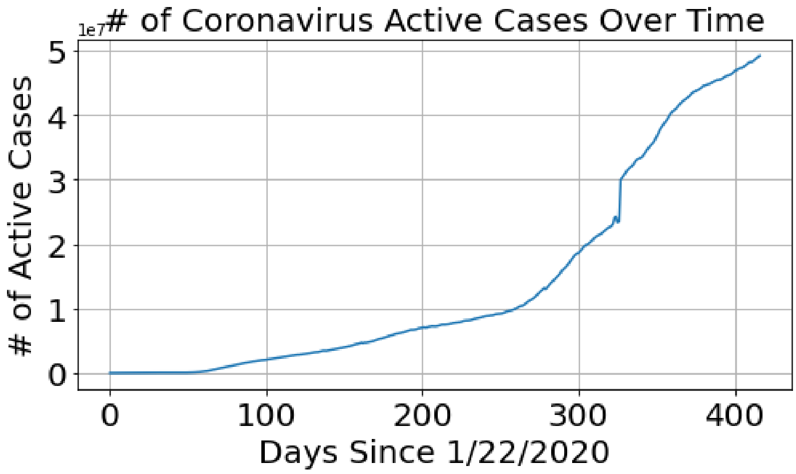
In [70]:
```python
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, total_deaths)
plt.title('# of Coronavirus Deaths Over Time', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```



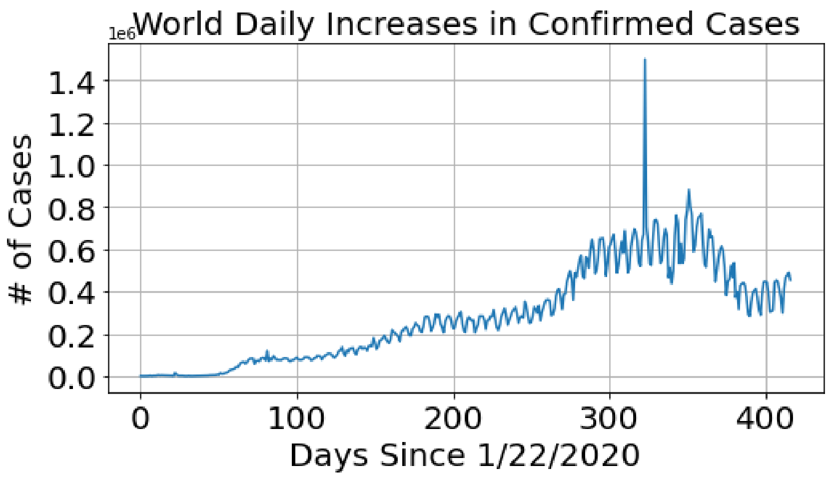
In [71]:
```python
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, total_recovered)
plt.title('# of Coronavirus Recoveries Over Time', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```



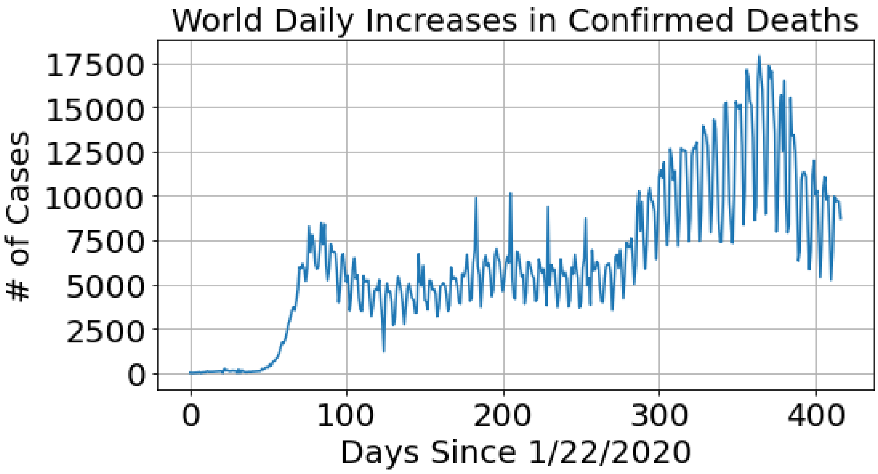
In [72]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, total_active)
plt.title('# of Coronavirus Active Cases Over Time', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Active Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```



In [97]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, world_daily_increase)
#plt.bar(adjusted_dates[270:320], world_daily_increase[270:320])
plt.title('World Daily Increases in Confirmed Cases', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```

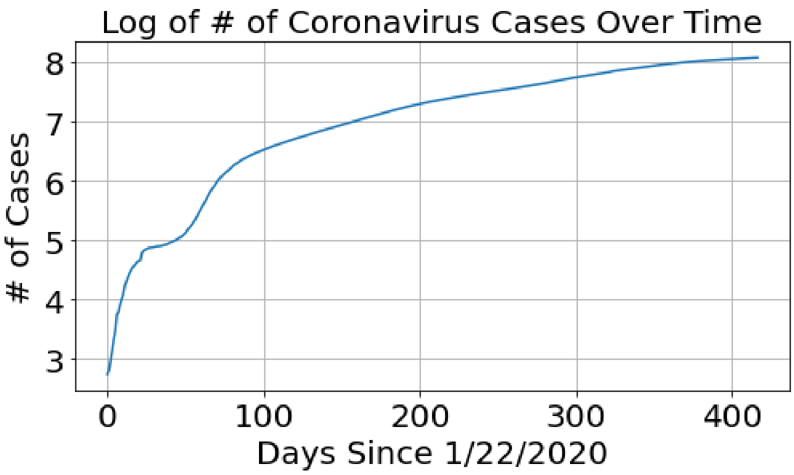


In [98]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, world_daily_death)
plt.title('World Daily Increases in Confirmed Deaths', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```

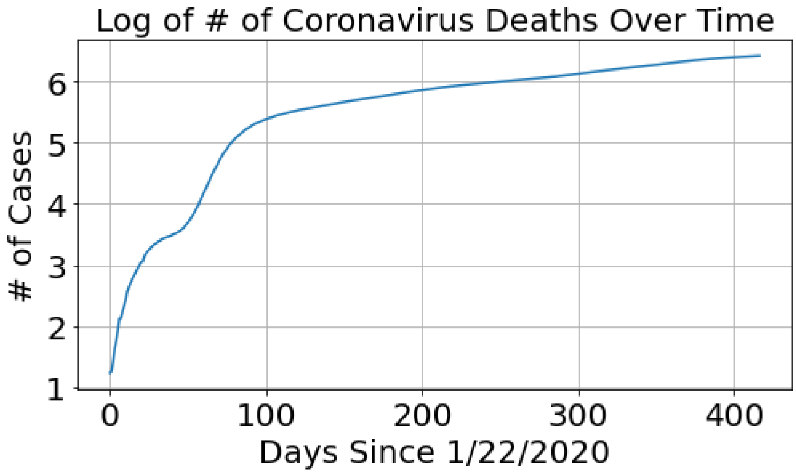


Log graphs

In [76]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, np.log10(world_cases))
plt.title('Log of # of Coronavirus Cases Over Time', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```



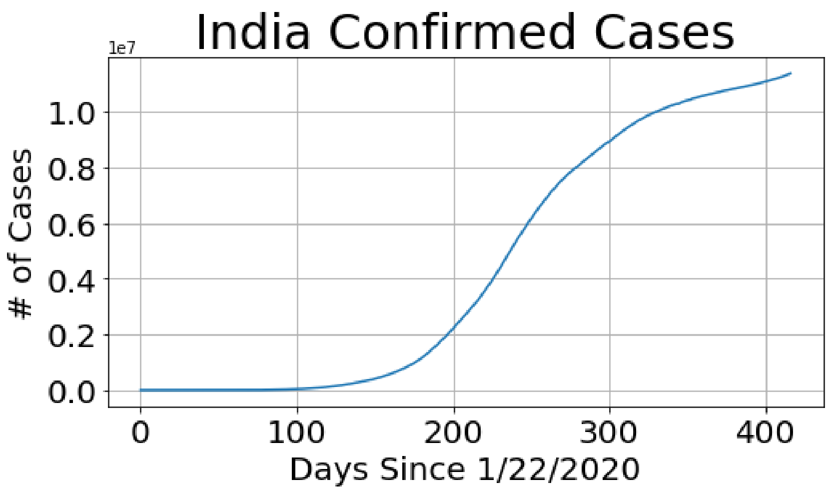
In [77]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, np.log10(total_deaths))
plt.title('Log of # of Coronavirus Deaths Over Time', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```



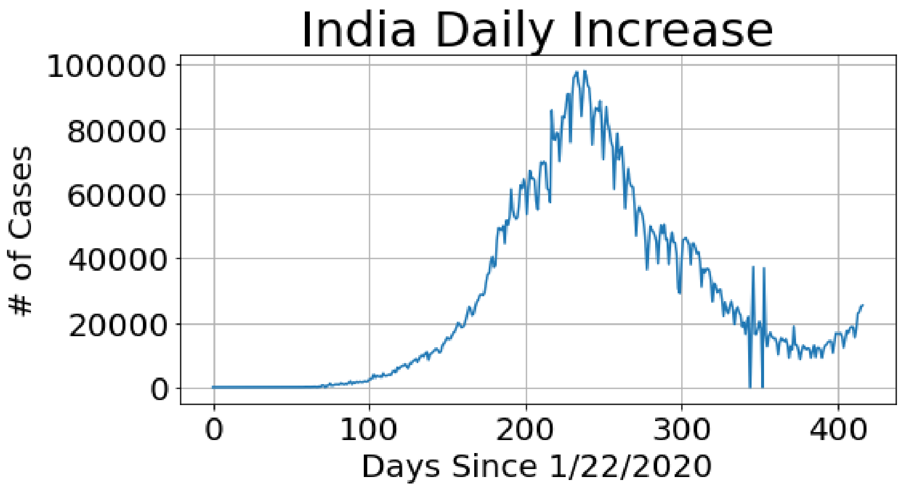
Country-wise Graphs

In [100]:
```
def country_plot(x, y, country, title):
    plt.figure(figsize=(8, 4))
    plt.plot(x, y)
    plt.title('{} {}'.format(country, title), size=30)
    plt.xlabel('Days Since 1/22/2020', size=20)
    plt.ylabel('# of Cases', size=20)
    plt.xticks(size=20)
    plt.yticks(size=20)
    plt.grid()
    plt.show()
```

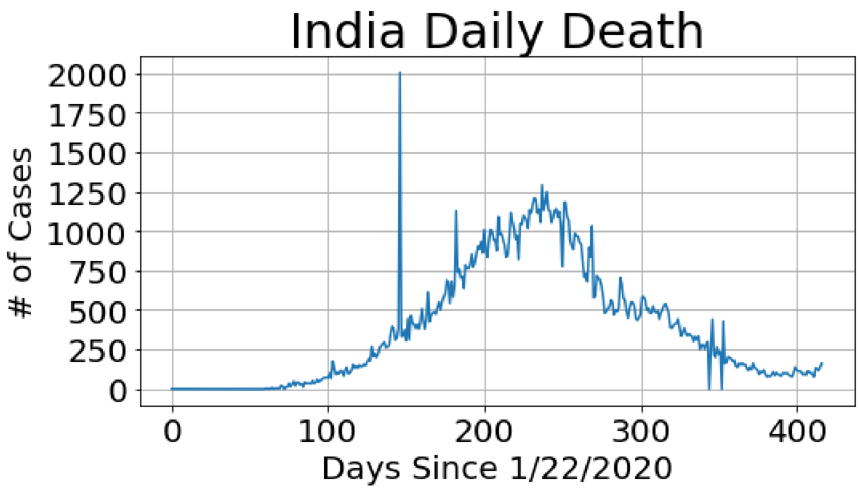
In [101]:
```
country_plot(adjusted_dates, india_cases, 'India', 'Confirmed Cases')
```



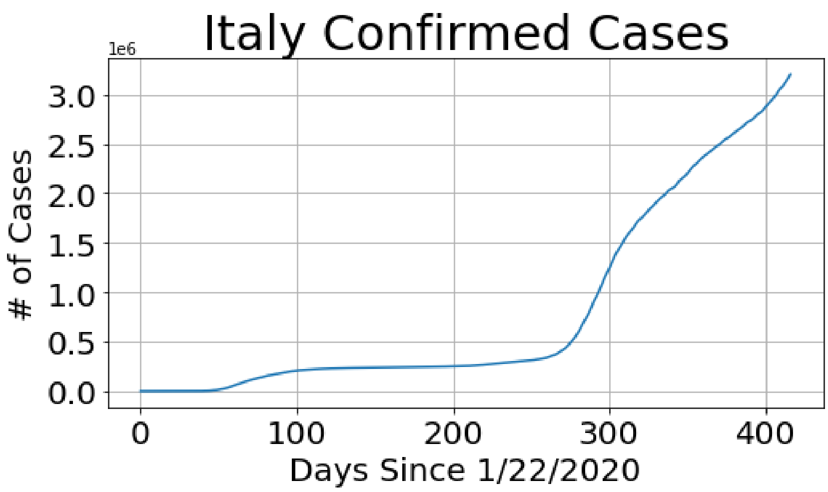
In [102]:
```
country_plot(adjusted_dates, india_daily_increase, 'India', 'Daily Increase')
```



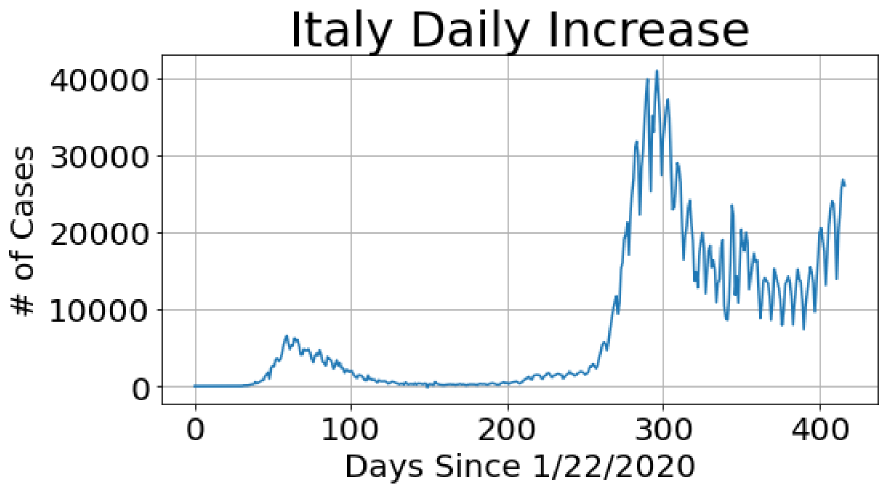
In [103]:
```
country_plot(adjusted_dates, india_daily_death, 'India', 'Daily Death')
```



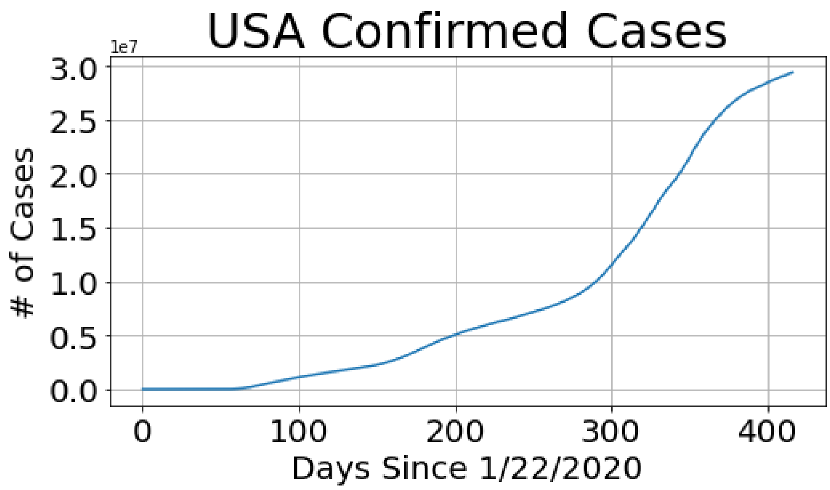
In [104]:
```
country_plot(adjusted_dates, italy_cases, 'Italy', 'Confirmed Cases')
```



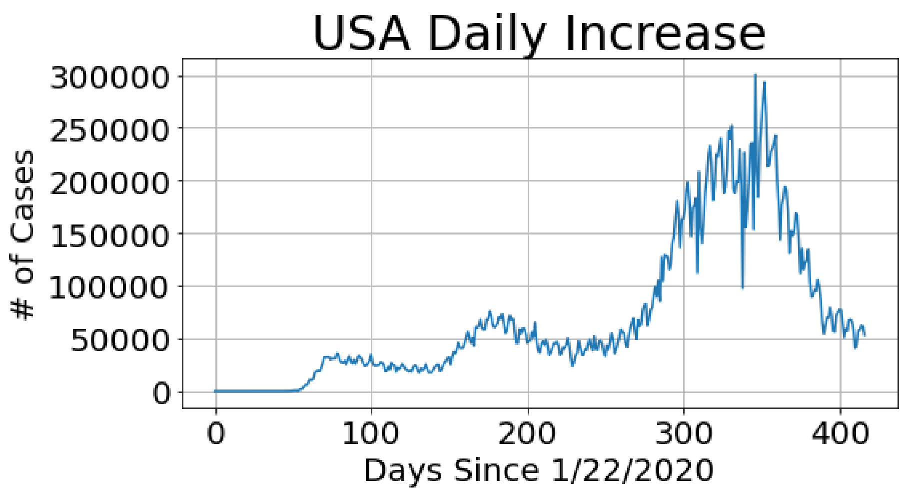
In [105]:
```
country_plot(adjusted_dates, italy_daily_increase, 'Italy', 'Daily Increase')
```



In [106]:
```
country_plot(adjusted_dates, us_cases, 'USA', 'Confirmed Cases')
```



In [107]:
```
country_plot(adjusted_dates, us_daily_increase, 'USA', 'Daily Increase')
```



Multiple plots in one graph

In [108]:
```
plt.figure(figsize=(8, 4))
plt.plot(adjusted_dates, china_cases)
plt.plot(adjusted_dates, italy_cases)
plt.plot(adjusted_dates, us_cases)
plt.plot(adjusted_dates, spain_cases)
plt.plot(adjusted_dates, india_cases)
plt.title('# of Coronavirus Cases', size=20)
plt.xlabel('Days Since 1/22/2020', size=20)
plt.ylabel('# of Cases', size=20)
plt.legend(['China', 'Italy', 'US', 'Spain', 'India'], prop={'size': 20})
plt.xticks(size=20)
plt.yticks(size=20)
plt.grid()
plt.show()
```