# Python For Data Mining

# PYTHON PACKAGES FOR DATA MINING

## NUMPY

NumPy is the fundamental package for scientific computing with Python. NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays

## SCIPY

SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.

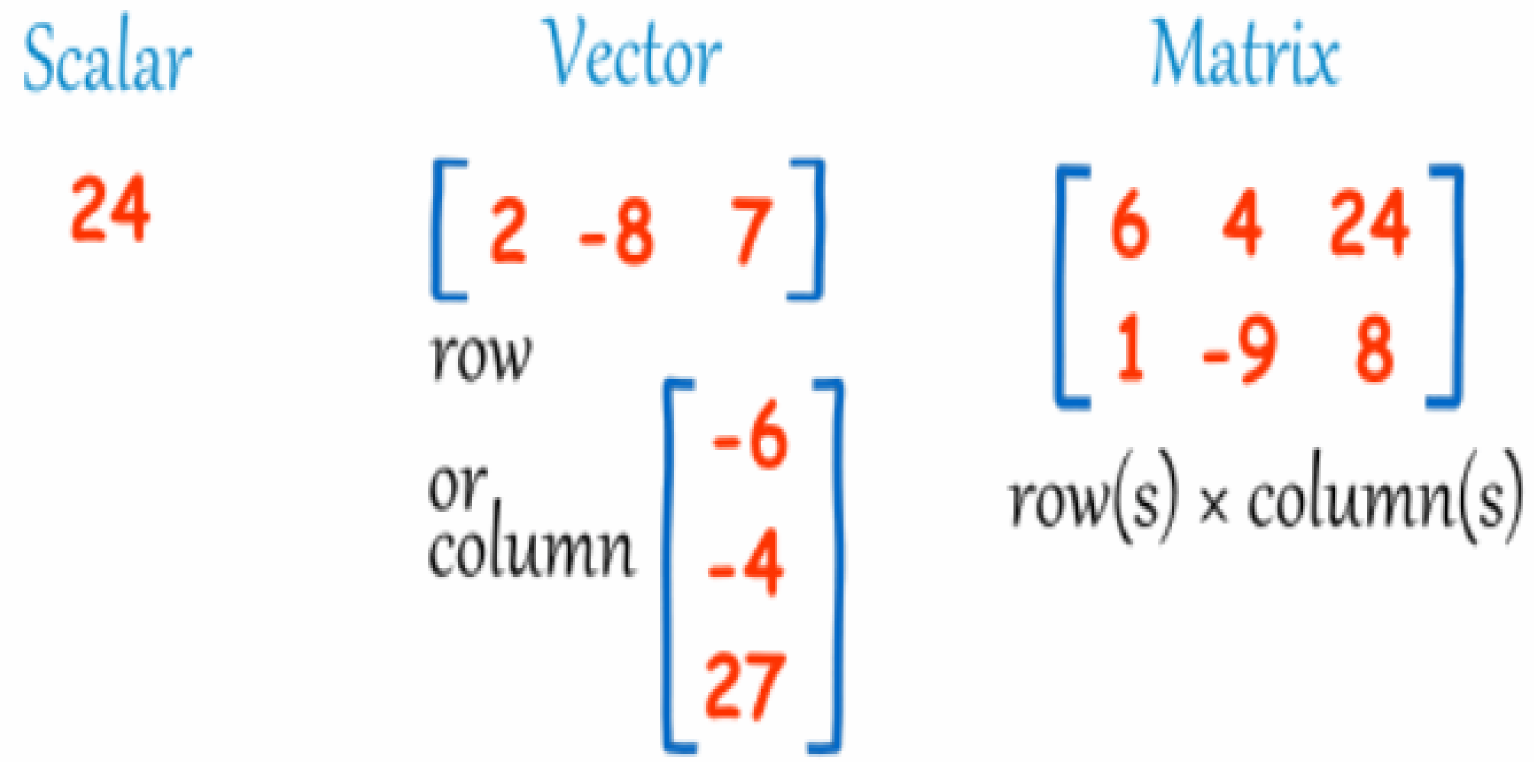
## Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

## MATPLOTLIB

matplotlib is a plotting library for the Python programming language and its NumPy numerical mathematics extension. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits.

# Scalars, Vectors and Matrices



Scalar

$24$

Vector

$\begin{bmatrix} 2 & -8 & 7 \end{bmatrix}$

row

or column

$\begin{bmatrix} -6 \\ -4 \\ 27 \end{bmatrix}$

Matrix

$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$

row(s) × column(s)

```
conda install -c anaconda numpy
```

# Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

```
import numpy as np
a = np.array([1, 2, 3])
print(type(a))
print(a.shape)
print(a[0], a[1], a[2])
a[0] = 5
print(a)
```

```
b = np.array([[1,2,3],[4,5,6]])
print(b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
```

```
import numpy as np
a=np.array([1,2,3,4,5,6],float)
print(a)
print(type(a))
```

```
import numpy as np
a=np.array([1,2,3,4,5,6],float)
print(a)
print(type(a))
print(a[3])
print(a[:3])
a[3]=54
print(a)
```

# Array

```
import numpy as np
array1=np.array([2,3,4,5,6])
array2=array1*2
print(array1)
print(array2)
```

```
import numpy as np
a=np.array([1,2,3,4,5,6,7,8,9],float)
print(a)
print(a.shape)
print(all(a))
print(any(a))
print(a[:1])
print(a[:3])
print(a[1:])
print(a[3:])
print(a[:-1])
```

```
import numpy as np
a=np.array([[1,2,3,4,5,6],[9,11,22,33,44,55]],float)
print(a)
print(type(a))
print(a[0,1])
print(a[1,1])
print(a[1:,])
print(a[:,2])
print(a.shape)
print(a.dtype)
print(len(a))
print(34 in a)
print(22 in a)
```

```
import numpy as np
a=np.array([2,3,4],float)
print(a)
s=a.tostring()
print(s)
print(np.fromstring(s))
a.fill(0)
print(a)
```

```
import numpy as np
a = np.array(range(6), float).reshape((2, 3))
print(a)
a.transpose()
print(a)
```

```
import numpy as np
a=np.array([1,2],float)
b=np.array([3,4,5],float)
c=np.array([6,7,8,9,0])
d=np.concatenate((a,b,c),float)
print(a)
print(b)
print(c)
print(d)
```

```
import numpy as np
a=np.array([[1,2],[3,4]],float)
b=np.array([[5,6],[7,8]],float)
c=np.concatenate((a,b))
print(c)
d=np.concatenate((a,b),axis=1)
print(d)
```

# Numpy functions to create arrays

```
import numpy as np
print(np.eye(4))
```

```
import numpy as  np
d=np.ones(4)
print(type(d))
```

```
import numpy as np
d=np.zeros(3)
d
```

```
c = np.full((2,2), 7)
print(c)
```

# Basic array operations

```
import numpy as np
a=np.array([1,2,4,5,68,9,0],float)
print(a)
print(a.sum())
print(a.prod())
print(a)
print(a.mean())
print(a.var())
print(a.std())
print(a.max())
print(a.min())
print(a.argmax())
print(a.argmin)
```

```
print(np.sum(a))
```

# Arange

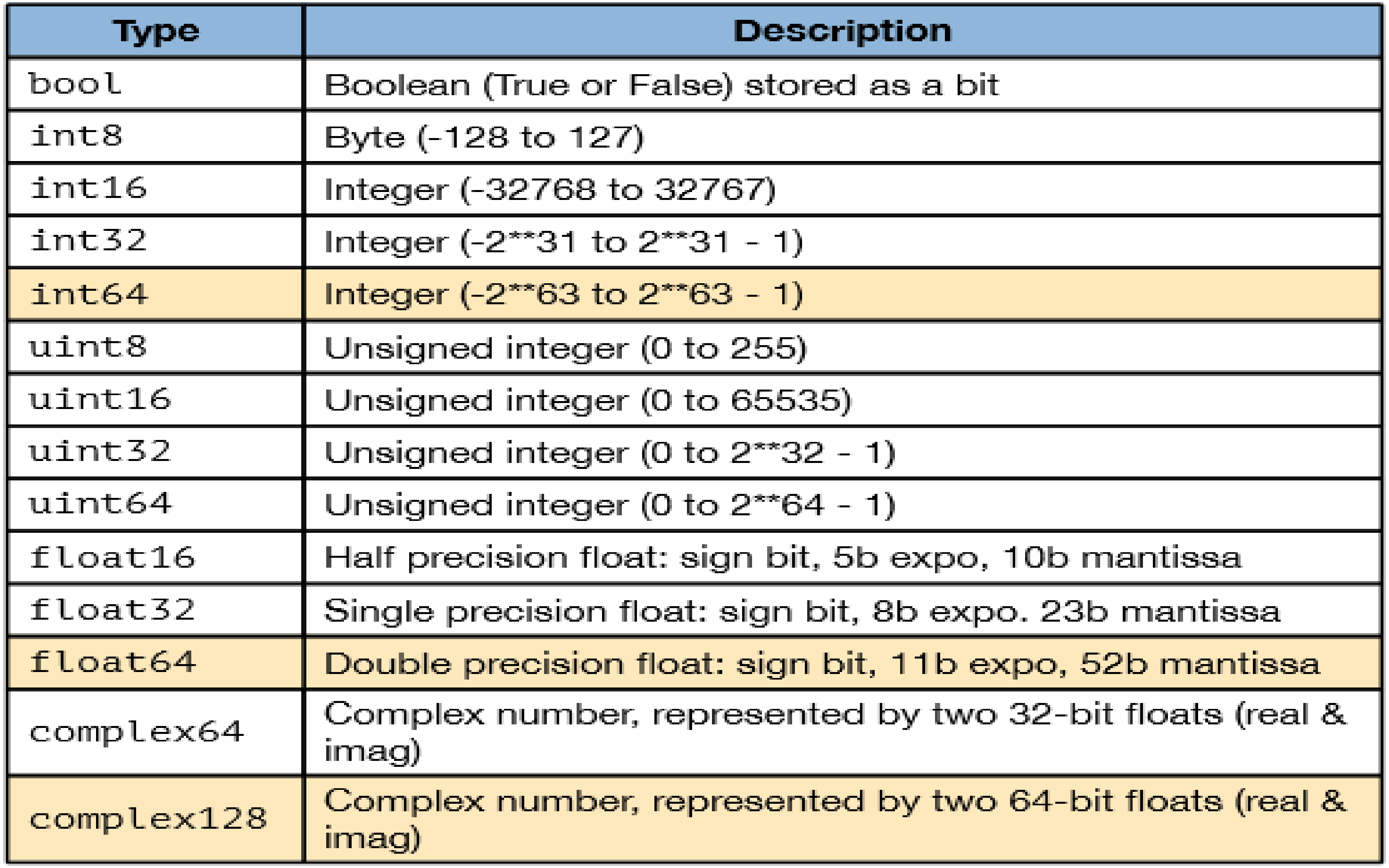
With NumPy we can get an array based on ranges. The arange method receives 1, 2 or 3 arguments

```
import numpy as np
array1=np.arange(5)
print(array1)
array1=np.arange(5,10)
print(array1)
array1=np.arange(0,10,2)
print(array1)
```

# Random numbers

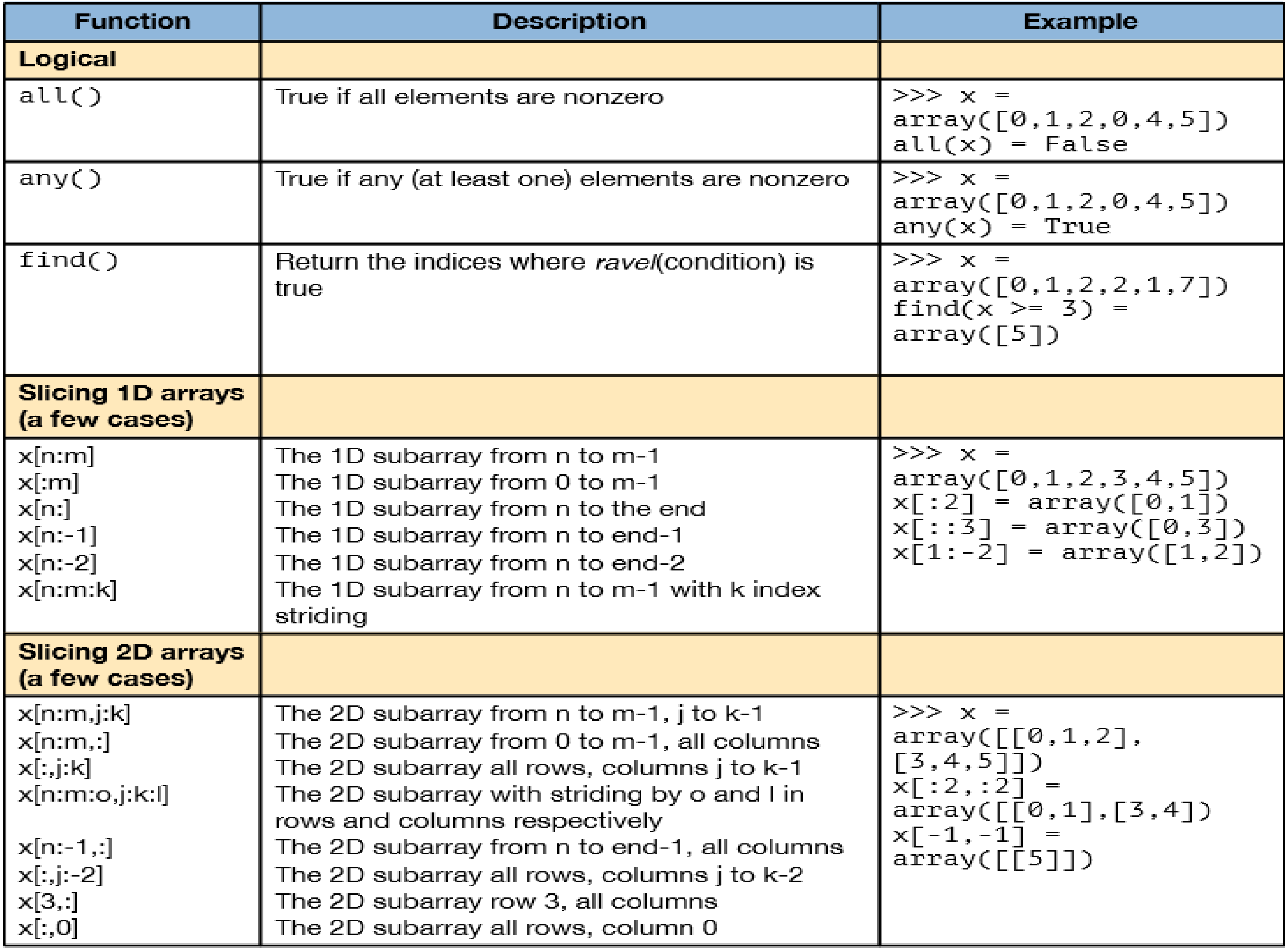
```
import numpy as np
a=np.random.rand(5)
print(a)
b=np.random.rand(2,3)
print(b)
c=np.random.random()
print(c)
d=np.random.randint(2,45)
print(d)
```

# ndarray data type



| Type | Description |
|---|---|
| bool | Boolean (True or False) stored as a bit |
| int8 | Byte (-128 to 127) |
| int16 | Integer (-32768 to 32767) |
| int32 | Integer (-2**31 to 2**31 - 1) |
| int64 | Integer (-2**63 to 2**63 - 1) |
| uint8 | Unsigned integer (0 to 255) |
| uint16 | Unsigned integer (0 to 65535) |
| uint32 | Unsigned integer (0 to 2**32 - 1) |
| uint64 | Unsigned integer (0 to 2**64 - 1) |
| float16 | Half precision float: sign bit, 5b expo, 10b mantissa |
| float32 | Single precision float: sign bit, 8b expo. 23b mantissa |
| float64 | Double precision float: sign bit, 11b expo, 52b mantissa |
| complex64 | Complex number, represented by two 32-bit floats (real & imag) |
| complex128 | Complex number, represented by two 64-bit floats (real & imag) |

## Popular methods for working with ndarrays



| Function | Description | Example |
|---|---|---|
| **Logical** | | |
| all() | True if all elements are nonzero | `>>> x = array([0,1,2,0,4,5])`<br>`all(x) = False` |
| any() | True if any (at least one) elements are nonzero | `>>> x = array([0,1,2,0,4,5])`<br>`any(x) = True` |
| find() | Return the indices where *ravel*(condition) is true | `>>> x = array([0,1,2,2,1,7])`<br>`find(x >= 3) = array([5])` |
| **Slicing 1D arrays (a few cases)** | | |
| x[n:m]<br>x[:m]<br>x[n:]<br>x[n:-1]<br>x[n:-2]<br>x[n:m:k] | The 1D subarray from n to m-1<br>The 1D subarray from 0 to m-1<br>The 1D subarray from n to the end<br>The 1D subarray from n to end-1<br>The 1D subarray from n to end-2<br>The 1D subarray from n to m-1 with k index striding | `>>> x = array([0,1,2,3,4,5])`<br>`x[:2] = array([0,1])`<br>`x[::3] = array([0,3])`<br>`x[1:-2] = array([1,2])` |
| **Slicing 2D arrays (a few cases)** | | |
| x[n:m,j:k]<br>x[n:m,:]<br>x[:,j:k]<br>x[n:m:o,j:k:l]<br>x[n:-1,:]<br>x[:,j:-2]<br>x[3,:]<br>x[:,0] | The 2D subarray from n to m-1, j to k-1<br>The 2D subarray from 0 to m-1, all columns<br>The 2D subarray all rows, columns j to k-1<br>The 2D subarray with striding by o and l in rows and columns respectively<br>The 2D subarray from n to end-1, all columns<br>The 2D subarray all rows, columns j to k-2<br>The 2D subarray row 3, all columns<br>The 2D subarray all rows, column 0 | `>>> x = array([[0,1,2],[3,4,5]])`<br>`x[:2,:2] = array([[0,1],[3,4])`<br>`x[-1,-1] = array([[5]])` |

| Function | Description | Example |
|---|---|---|
| **Shape & Concatenation** | | |
| `reshape()` | Reshape a 1D or 2D array to a new shape; the new shape must be consistent. | `>>> x = arange(0,5)`<br>`1D 6 elements`<br>`y = reshape(x,(2,3))`<br>`2D 2x3 elements` |
| `concatenate()` | Join a sequence of arrays together. The arrays must have the same shape except in the axis used for combining. axis=0 is rows, axis=1 is columns. | `>>> x = array([[0,1,2,3,4,5]])`<br>`2D 1x6 elements`<br>`concatenate((x,x), axis=0)`<br>`2D 1x6 elements`<br>`concatenate((x,x)), axis=1)`<br>`2D 1x12 elements` |
| `hstack()` | Stack arrays horizontally. A subset of `concatenate()` | `>>> x = array([[0,1,2,3,4,5]])`<br>`2D 1x6 elements`<br>`x = x.T #transpose`<br>`y=hstack((x,x))`<br>`2D 6x2 columns` |
| `vstack()` | Stack arrays vertically. A subset of `concatenate()` | `>>> x = array([[0,1,2,3,4,5]])`<br>`2D 1x6 elements`<br>`y=vstack((x,x))`<br>`2D 2x6 columns` |
| `flatten()` | Values of the argument array become a 1D array. May be done in-place with x.flatten() | `>>> x = array([[0,1,2,3,4,5]])`<br>`x.flatten()`<br>`1D 6 element` |
| `transpose()` or `array.T` | Like matrix transpose for 2D arrays. In-place via `x.T`. | `>>> x = array([[0,1,2,3,4,5]])`<br>`x.T`<br>`2D 6x1 array` |

| Math | Many other standard functions, e.g., trig, are also available for array operations | |
|---|---|---|
| mean(x) | The sample mean of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>mean(x) = 2.5 |
| var(x) | The sample variance of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>var(x) = 2.9167 |
| std(x) | The sample standard deviation of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>std(x) = 1.7078 |
| sum(x) | The sum of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>sum(x) = 15 |
| prod(x) | The sample mean of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>prod(x) = 0 |
| cumsum(x) | The sample mean of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>cumsum(x) = array([0,<br>1,3,6,10,15]) |
| cumprod(x) | The sample mean of the values contained in array x. | >>> x =<br>array([1,1,2,3,4,5])<br>cumprod(x) = array([1,<br>1,2,6,24,120]) |
| min(x) | The sample mean of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>min(x) = 0 |
| max(x) | The sample mean of the values contained in array x. | >>> x =<br>array([0,1,2,3,4,5])<br>max(x) = 5 |
| conj(x) | The sample mean of the values contained in array x. | >>> x = array([2+5j])<br>conj(x) =<br>array([2+5j]) |
| x.real & x.imag | The real or imaginary part of the values contained in array x. Also real(x), imag(x) | >>> x = array([2+5j])<br>x.real = array([ 2.])<br>imag(x) = array([ 5.]) |

$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

```
import numpy as np
m1=np.array([3.,3.])
m2=np.array([2.,2.])
print(10 * np.dot(m1,m2))
```

# Matplotlib

The library is generally used as follows:

1.Call a plotting function with some data (e.g. plot()).
2.Call many functions to setup the properties of the plot (e.g. labels and colors).
3.Make the plot visible (e.g. show()).

```
from matplotlib import pyplot as plt
plt.plot([1,2,3],[4,5,1])
plt.show()
```

```
import matplotlib.pyplot as plt plt.plot([1,2,3,4], [1,4,9,16],
'ro')
plt.axis([0, 6, 0, 20]) plt.show()
```

```
from matplotlib import pyplot as plt from matplotlib
import style
 import numpy as np
style.use('ggplot')
 x,y = np.loadtxt('exampleFile.csv', unpack=True,
delimiter = ',')
plt.plot(x,y)
plt.title('Epic Info')
 plt.ylabel('Y axis')
plt.xlabel('X axis')
 plt.show()
```

```
import matplotlib.pyplot as plt
import numpy
myarray = numpy.array([1, 2, 3])
plt.plot(myarray)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy
x = numpy.array([1, 2, 3])
y = numpy.array([2, 4, 6])
plt.scatter(x,y)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```

# Pandas data types

**Series**

**DataFrame**

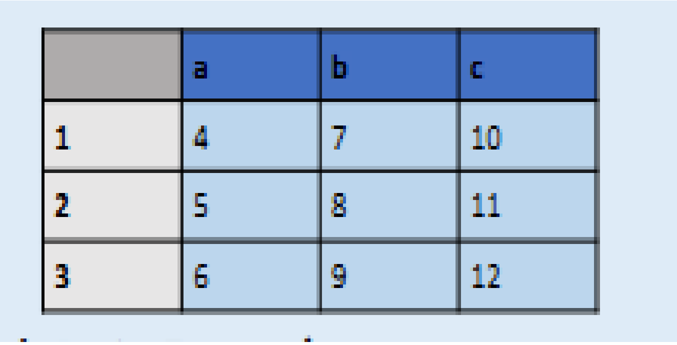
**Panel**

**Panel4D**

```
import pandas as pd
c=pd.Series([1,2,3,4,5])
c
```

```
c.values
```

# Creating DataFrame



| | a | b | c |
|---|---|---|---|
| 1 | 4 | 7 | 10 |
| 2 | 5 | 8 | 11 |
| 3 | 6 | 9 | 12 |

```
import pandas as pd
df=pd.DataFrame({"a":[4,5,6],"b":[7,8,9],"c":[10,11,12]},
index=[1,2,3])
```

```
df=pd.DataFrame([[4,7,10],[5,8,11],[6,9,12]],index=[1,2,3],
columns=["a","b","c"])
```

```
states ={'State' :['Gujarat', 'Tamil Nadu', ' Andhra', 'Karnataka',
'Kerala'], 'Population': [36, 44, 67,89,34],'Language' :['Gujarati',
'Tamil', 'Telugu', 'Kannada', 'Malayalam']}
```

```
State=pd.DataFrame(states)
State
```

```
df=pd.DataFrame({"x1":['a','b','c'],"x2":[1,2,3]})
df
```

```
df1=pd.DataFrame({"x1":['A','B','C'],"x3":["T","F","T"]})
df1
```

```
pd.merge(df,df1,how='left',on='x1')
```

```
pd.merge(df,df1,how='right',on='x1')
```

```
pd.merge(df,df1,how='inner',on='x1')
```

```
pd.merge(df,df1,how='outer',on='x1')
```

# Pandas indexing

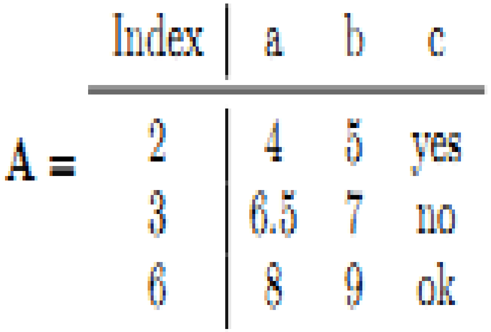
For indexing the rows loc, iloc and ix

**A.loc[3,:]**

**A.ix[2,:]**

**A.iloc[2,:]**

**A.loc[:,'b']**



A =

| Index | a | b | c |
|---|---|---|---|
| 2 | 4 | 5 | yes |
| 3 | 6.5 | 7 | no |
| 6 | 8 | 9 | ok |

```
import pandas as pd
A = pd.DataFrame([[4, 5, 'yes'], [6.5, 7, 'no'], [8, 9, 'ok']],
index=[2, 3, 6], columns=['a', 'b', 'c'])
```

```
import pandas as pd
data = pd.DataFrame({'group': ['a', 'a', 'a', 'b','b', 'b', 'c', 'c','c'],
                     'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
```

```
data.describe()
```

df.info() shows data types, number of rows and columns, and memory usage of your data frame

**Sorting**

```
data.sort_values(by=['group','ounces'], ascending=[False, True], inplace=True)
data
```

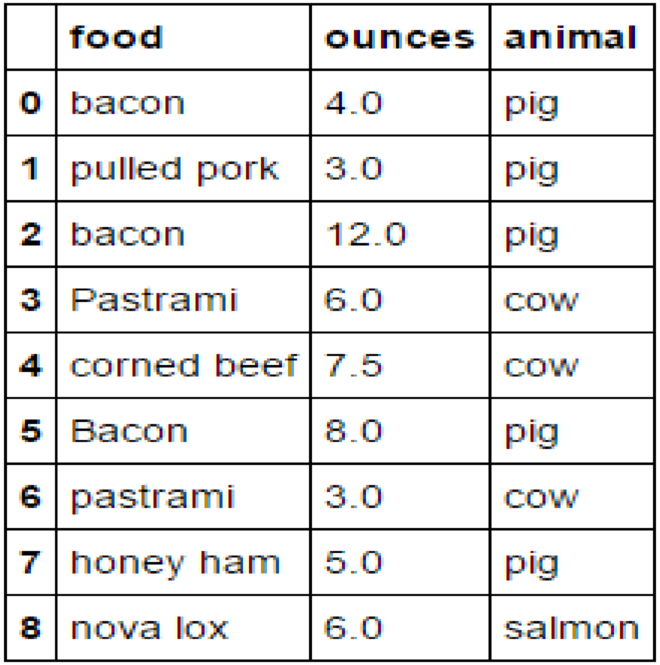
## Removing duplicates

```
import pandas as pd
data = pd.DataFrame({'k1': ['one'] * 3 + ['two'] * 4, 'k2': [3, 2, 1, 3,
3, 4, 4]})
```

```
data.drop_duplicates()
data.drop_duplicates(subset='k1')
```

**Creating a new column based on values from another column**

```
import pandas as pd
data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
'Pastrami','corned beef', 'Bacon', 'pastrami', 'honey ham','nova
lox'],'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
```

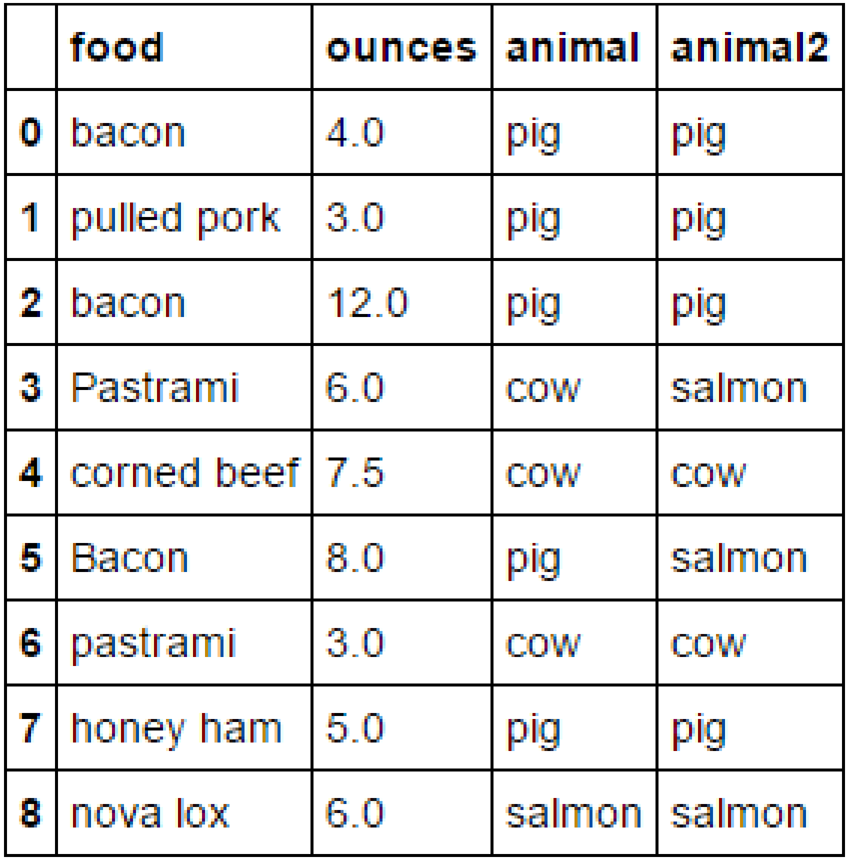


| | food | ounces | animal |
|---|---|---|---|
| 0 | bacon | 4.0 | pig |
| 1 | pulled pork | 3.0 | pig |
| 2 | bacon | 12.0 | pig |
| 3 | Pastrami | 6.0 | cow |
| 4 | corned beef | 7.5 | cow |
| 5 | Bacon | 8.0 | pig |
| 6 | pastrami | 3.0 | cow |
| 7 | honey ham | 5.0 | pig |
| 8 | nova lox | 6.0 | salmon |

```
meat_to_animal = {
'bacon': 'pig',
'pulled pork': 'pig',
'pastrami': 'cow',
'corned beef': 'cow',
'honey ham': 'pig',
'nova lox': 'salmon'
}
```

```
def meat2animal(series):
    if series["food"]=='bacon':
        return 'pig'
    elif series["food"]=='pulled pork':
        return 'pig'
    elif series["food"]=='pastrami':
        return 'cow'
    elif series["food"]=='corned beef':
        return 'cow'
    elif series["food"]=='honey ham':
        return 'pig'
    else:
        return 'salmon'
```

```
data['animal'] =
data['food'].map(str.lower).map(meat_to_animal)
data
```



| | food | ounces | animal | animal2 |
|---|---|---|---|---|
| 0 | bacon | 4.0 | pig | pig |
| 1 | pulled pork | 3.0 | pig | pig |
| 2 | bacon | 12.0 | pig | pig |
| 3 | Pastrami | 6.0 | cow | salmon |
| 4 | corned beef | 7.5 | cow | cow |
| 5 | Bacon | 8.0 | pig | salmon |
| 6 | pastrami | 3.0 | cow | cow |
| 7 | honey ham | 5.0 | pig | pig |
| 8 | nova lox | 6.0 | salmon | salmon |

```
data['animal2'] = data.apply(meat2animal,axis='columns')
data
```

## Removing or dropping a column

```
data.drop('animal', axis='columns', inplace=True)
data
```

```
import pandas as pd
import numpy as np
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data
```

```
data.replace(-999, np.nan, inplace=True)
data
```

```
import pandas as pd
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data
```

```
data.replace([-999, -1000], np.nan, inplace=True)
data
```

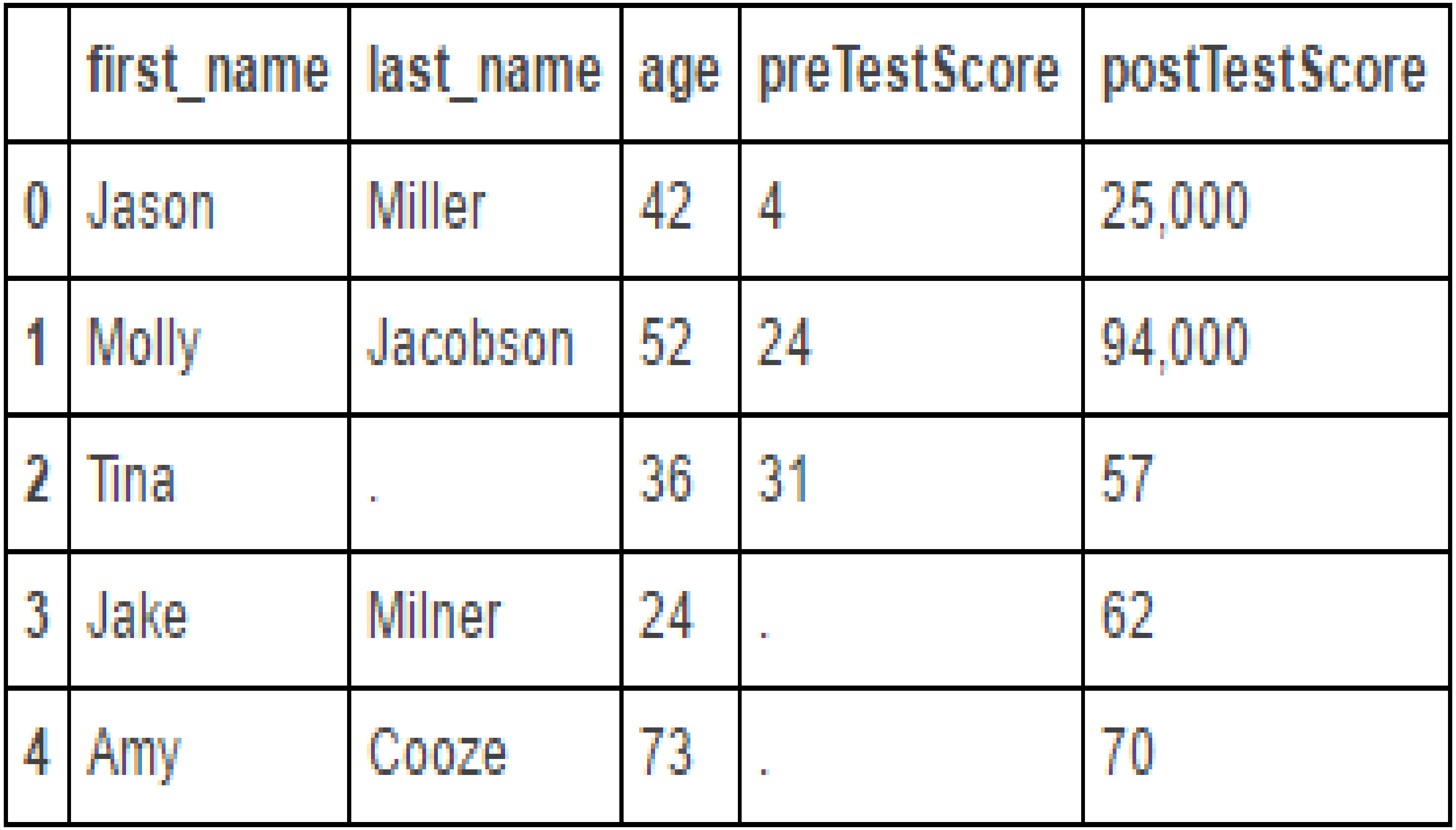
**Dropping NaN**

```
data.dropna()
```

Filling/replacing NaN values with something else (replace NaN with 0/zero)

```
data.dropna()
```

# Create dataframe



| | first_name | last_name | age | preTestScore | postTestScore |
|---|---|---|---|---|---|
| 0 | Jason | Miller | 42 | 4 | 25,000 |
| 1 | Molly | Jacobson | 52 | 24 | 94,000 |
| 2 | Tina | . | 36 | 31 | 57 |
| 3 | Jake | Milner | 24 | . | 62 |
| 4 | Amy | Cooze | 73 | . | 70 |

```
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', 'Jacobson', ".", 'Milner', 'Cooze'],
        'age': [42, 52, 36, 24, 73],
        'preTestScore': [4, 24, 31, ".", "."],
        'postTestScore': ["25,000", "94,000", 57, 62, 70]}
```

```
df = pd.DataFrame(raw_data, columns = ['first_name',
'last_name', 'age', 'preTestScore', 'postTestScore'])
df
```

Save dataframe as csv in the working director

```
df.to_csv('C:/Users/shubh/Desktop/python/data.csv')
```

Load a csv

```
df=pd.read_csv('C:/Users/shubh/Desktop/python/data.csv')
```

Load a csv with no headers

```
df=pd.read_csv('C:/Users/shubh/Desktop/python/data.csv')
df
```

# Pivot table

In data processing, a ***pivot table*** is a data summarization tool found in data visualization programs such as spreadsheets or business intelligence software. Among other functions, a pivot table can automatically sort, count, total or average the data stored in one table or spreadsheet, displaying the results in a second table showing the summarized data.

```
pandas.pivot_table(data, values=None, index=None,
columns=None, aggfunc='mean', fill_value=None, margins=False,
dropna=True, margins_name='All')
```

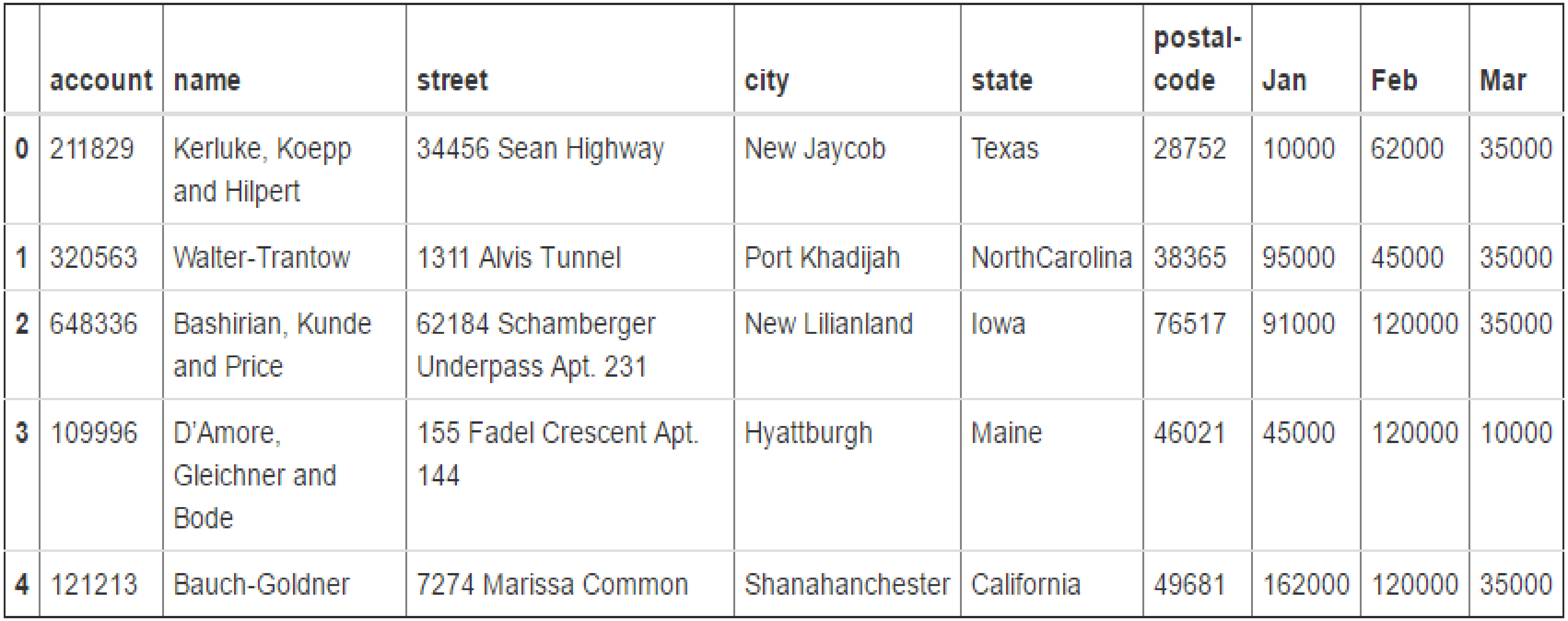
# EXCEL/CSV File

```
import pandas as pd
df=pd.read_excel('C:/Users/shubh/Desktop/data/sales-funnel.xlsx')
```

```
import pandas as pd
df=pd.read_csv('C:/Users/shubh/Desktop/data/sales-funnel.xlsx')
```

```
df.head()
```

```
import pandas as pd
import numpy as np
df=pd.read_excel('C:/Users/shubh/Desktop/data/
excel-comp-data.xlsx')
df.head()
```



|  | account | name | street | city | state | postal-code | Jan | Feb | Mar |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 211829 | Kerluke, Koepp and Hilpert | 34456 Sean Highway | New Jaycob | Texas | 28752 | 10000 | 62000 | 35000 |
| 1 | 320563 | Walter-Trantow | 1311 Alvis Tunnel | Port Khadijah | NorthCarolina | 38365 | 95000 | 45000 | 35000 |
| 2 | 648336 | Bashirian, Kunde and Price | 62184 Schamberger Underpass Apt. 231 | New Lilianland | Iowa | 76517 | 91000 | 120000 | 35000 |
| 3 | 109996 | D'Amore, Gleichner and Bode | 155 Fadel Crescent Apt. 144 | Hyattburgh | Maine | 46021 | 45000 | 120000 | 10000 |
| 4 | 121213 | Bauch-Goldner | 7274 Marissa Common | Shanahanchester | California | 49681 | 162000 | 120000 | 35000 |

```
df["total"] = df["Jan"] + df["Feb"] + df["Mar"]
df.head()
```

Performing column level analysis is easy in pandas

```
df["Jan"].sum(), df["Jan"].mean(),df["Jan"].min(),df["Jan"].max()
```

```
sum_row=df[["Jan","Feb","Mar","total"]].sum()
sum_row
```

```
df_sum=pd.DataFrame(data=sum_row).T
df_sum
```

# PANDAS Example #

```
import numpy as np
import matplotlib as p
import pdb
from pandas import *
data = read_csv('C:/Users/shubh/Desktop/data/train.csv')
data
```

```
data.describe()
```

```
import numpy as np
import matplotlib as p
import pdb
from pandas import *
data = read_csv('C:/Users/shubh/Desktop/data/train.csv')
data
```

```
men = data[data.Sex == 'male']
  women = data[data.Sex =='female']
```

We can then find the proportion of men and women that survive

```
proportion_women_survived =
float(sum(women.Survived))/len(women)
proportion_men_survived = float(sum(men.Survived))/len(men)
```

```
data.groupby('Sex').Survived.mean()
```

```
men.Age
women.Age
```

```
data.columns
```

Now we might want to create a new column that stores our prediction for whether someone survived or not. Let's call this new column 'prediction'

```
data['prediction']=0
```

```
data.prediction[data.sex == 'female']=1
```

# Visualization

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

```
#import matplotlib libary
import matplotlib.pyplot as plt
#define some data
x = [1,2,3,4]
y = [20, 21, 20.5, 20.8]
#plot data
plt.plot(x, y)
#show plot
plt.show()
```

# Regression analysis using Python

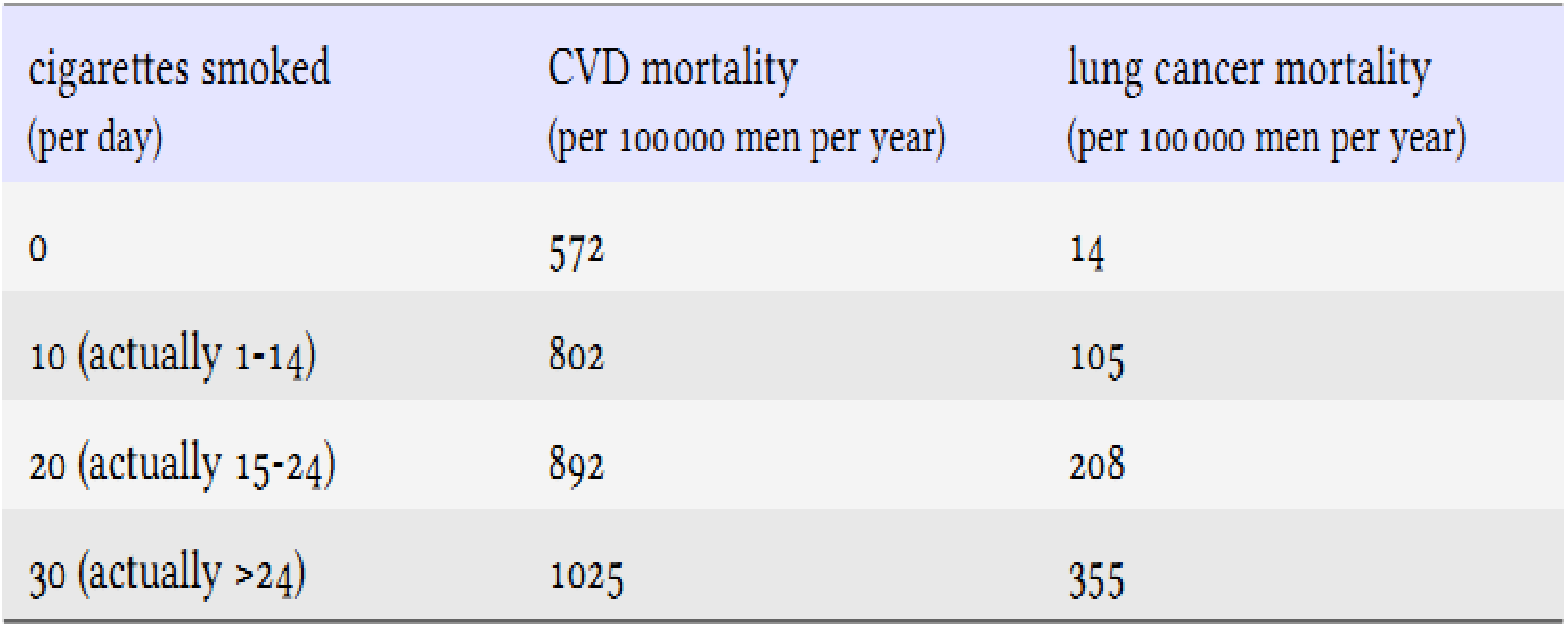
Linear regression analysis means “fitting a straight line to data”

- also called linear modelling

It’s a widely used technique to help model and understand real-world phenomena

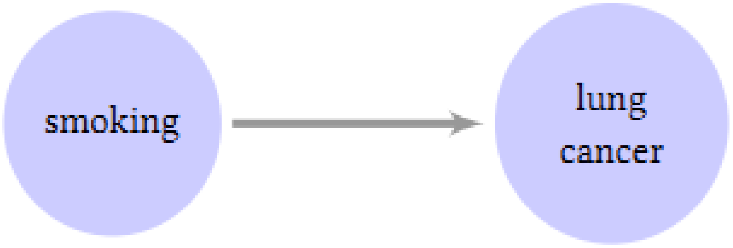
- easy to use
- easy to understand intuitively

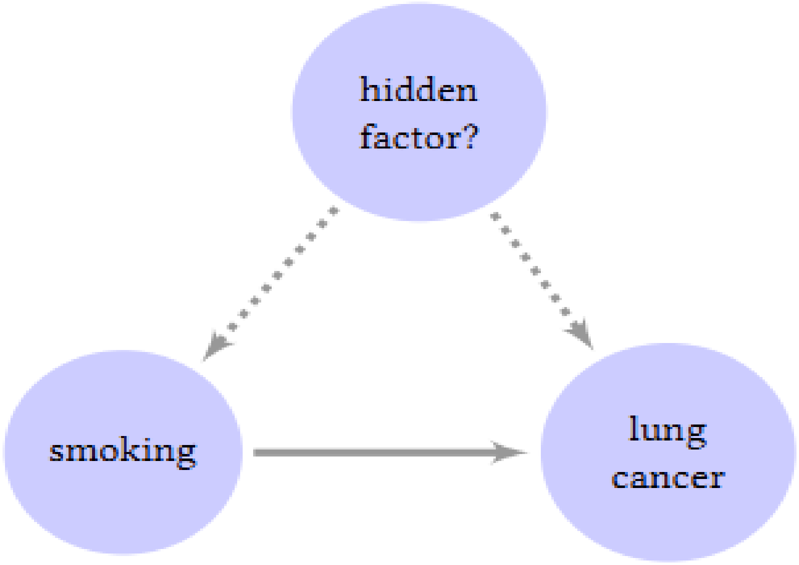
**Allows prediction**



| cigarettes smoked<br>(per day) | CVD mortality<br>(per 100 000 men per year) | lung cancer mortality<br>(per 100 000 men per year) |
|---|---|---|
| 0 | 572 | 14 |
| 10 (actually 1-14) | 802 | 105 |
| 20 (actually 15-24) | 892 | 208 |
| 30 (actually >24) | 1025 | 355 |

```
import pandas
import matplotlib.pyplot as plt
data = pandas.DataFrame({'cigarettes': [0,10,20,30],
'CVD': [572,802,892,1025],
'lung': [14,105,208,355]});
data.plot('cigarettes', 'CVD', kind='scatter')
plt.title("Deaths for different smoking intensities")
plt.xlabel("Cigarettes smoked per day")
plt.ylabel("CVD deaths")
plt.show()
```


smoking
lung cancer


hidden factor?
smoking
lung cancer

```
import numpy, pandas
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
df = pandas.DataFrame({'cigarettes': [0,10,20,30],
'CVD': [572,802,892,1025],
'lung': [14,105,208,355]});
df.plot('cigarettes', 'CVD', kind='scatter')
lm = smf.ols("CVD ~ cigarettes", data=df).fit()
xmin = df.cigarettes.min()
xmax = df.cigarettes.max()
X = numpy.linspace(xmin, xmax, 100)
# params[0] is the intercept (beta0)
# params[1] is the slope (beta1)
Y = lm.params[0] + lm.params[1] * X
plt.plot(X, Y, color="darkgreen")
plt.show()
```

```
ipython qtconsole --pylab=inline
```