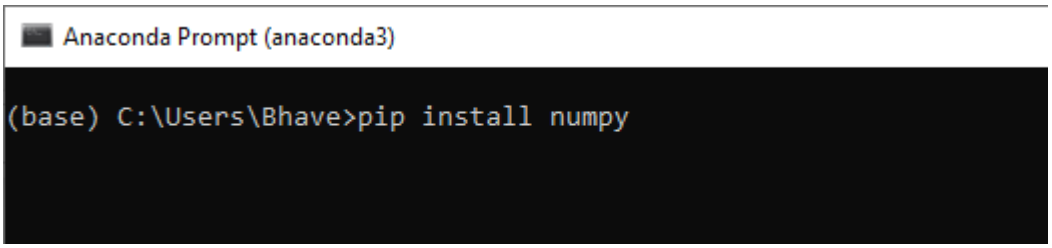
DAY 2

NUMPY LIBRARY

Numpy Arrays

NumPy arrays are the main way we will use Numpy throughout the course. Numpy arrays essentially are of two types: vectors and matrices. Vectors are strictly 1-d arrays and matrices are 2-d (but a matrix can still have only one row or one column).

INSTALLING NUMPY LIBRARY

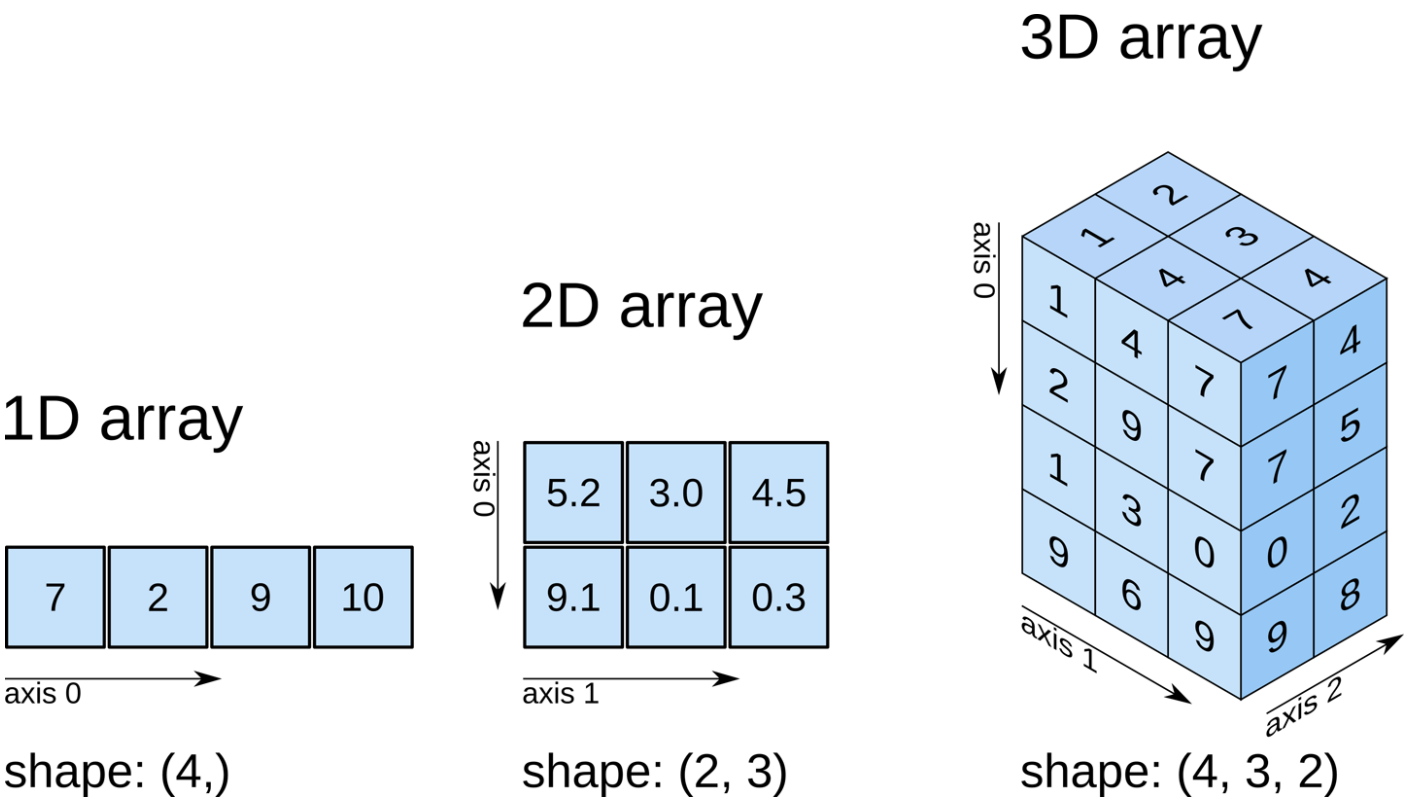


IMPORT NUMPY LIBRARY

Once NumPy is installed, import it in your applications by adding the import keyword:

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

Creating NumPy Arrays



From a Python List

We can create an array by directly converting a list or list of lists:

In [9]:
```
my_list = [1,2,3]
my_list
```

Out[9]:
```
[1, 2, 3]
```

In [10]:
```
arr=np.array(my_list)
```

In [11]:
```
type(arr)
```

Out[11]:
```
numpy.ndarray
```

we will create 2-D array using list of list

In [15]:
```
arr=np.array([[1,2,3],[4,5,6],[5,6,7],[8,9,10]])
```

array object has an atribute as shape which gives you shape of created array

number of rows and columns and channnels information

In [17]:
```
arr.shape
```

Out[17]:
```
(4, 3)
```

Creating array from tuple

In [12]:
```
arr = np.array((1, 2, 3, 4, 5))

arr
```

Out[12]:
```
array([1, 2, 3, 4, 5])
```

In [13]:
```
type(arr)
```

Out[13]:
```
numpy.ndarray
```

Built in methods to generate arrays

Create an array of 10 zeros

In [ ]:
```
np.zeros(10)
```

Create an array of 10 ones

In [ ]:
```
np.ones(10)
```

Create an array of the integers from 10 to 50

In [ ]:
```
np.arange(10,51)
```

Create an array of all the even integers from 10 to 50

In [ ]:
```
np.arange(10,51,2)
```

Create a 3x3 matrix with values ranging from 0 to 8

In [3]:
```
### to generate below matrix
```

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

In [5]:
```
arr=np.arange(9)
print(arr)
```

```
[0 1 2 3 4 5 6 7 8]
```

In [ ]:
```

```

In [6]:
```
arr.reshape(3,3)
```

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

rules of reshaping

In [9]:
```
arr=np.arange(0,13)
print(arr)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12]
```

In [10]:
```
arr.shape
```

Out[10]:
```
(13,)
```

In [11]:
```
arr.reshape((2,3,2))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-9260ae725992> in <module>
----> 1 arr.reshape((2,3,2))

ValueError: cannot reshape array of size 13 into shape (2,3,2)
```

Create a 3x3 identity matrix

In [7]:
```
np.eye(3)
```

Out[7]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Use NumPy to generate a random number between 0 and 1

In [8]:
```
np.random.rand(1)
```

Out[8]:
```
array([0.1964267])
```

Use NumPy to generate an array of 25 random numbers sampled from a standard normal distribution

In [12]:
```
np.random.randn(25)
```

Out[12]:
```
array([-3.20014652e-01,  9.84046590e-01,  1.70752606e+00, -5.43141571e-01,
       -1.23075072e+00, -6.80607169e-01, -2.62885050e+00, -4.99637118e-01,
        5.64847171e-01,  1.87240328e+00,  6.45229507e-01, -6.15585449e-01,
       -1.54986535e-01,  1.31486318e-03,  5.40980780e-01, -1.46380321e-01,
       -7.79668537e-01, -2.81761249e+00,  6.37305957e-01, -9.58376807e-02,
        8.65414340e-01,  6.23169621e-01, -5.21952911e-01, -6.43645057e-01,
        4.39230484e-01])
```

matrix using arange method

In [13]:
```
np.arange(1,101).reshape(10,10) / 100
```

Out[13]:
```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

Create an array of 20 linearly spaced points between 0 and 1:

In [14]:
```
np.linspace(0,1,20)
```

Out[14]:
```
array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

Numpy Indexing and Selection

In [15]:
```
mat = np.arange(1,26).reshape(5,5)
mat
```

Out[15]:
```
array([[ 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]])
```

Indexing rules

- 1. check the shape off given array
- 1. if it is 1-D array then indexing is similar to indexing in a list
- 1. if its a 2-D array then indexing is done as [row,col]
- 1. if its a 3-D array then indexing is done as [channel,row,col]

In [16]:
```
mat[3:5]
```

Out[16]:
```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

In [17]:
```
mat[2:,1:]
```

Out[17]:
```
array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

In [18]:
```
mat[:3,1:2]
```

Out[18]:
```
array([[ 2],
       [ 7],
       [12]])
```

Get the sum of all the values in mat

In [19]:
```
mat
```

Out[19]:
```
array([[ 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]])
```

In [20]:
```
mat.sum()
```

Out[20]:
```
325
```

In [21]:
```
mat.sum(axis=1)
```

Out[21]:
```
array([ 15,  40,  65,  90, 115])
```

In [22]:
```
mat.sum(axis=0)
```

Out[22]:
```
array([55, 60, 65, 70, 75])
```

In [23]:
```
mat.sum(axis=-1)
```

Out[23]:
```
array([ 15,  40,  65,  90, 115])
```

In [25]:
```
mat1=np.arange(0,27).reshape((3,3,3))
```

In [26]:
```
mat1
```

Out[26]:
```
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]]])
```

In [27]:
```
mat1.sum(axis=-1)
```

Out[27]:
```
array([[ 3, 12, 21],
       [30, 39, 48],
       [57, 66, 75]])
```

In [28]:
```
mat1.sum(axis=1)
```

Out[28]:
```
array([[ 9, 12, 15],
       [36, 39, 42],
       [63, 66, 69]])
```

In [29]:
```
mat1.sum(axis=0)
```

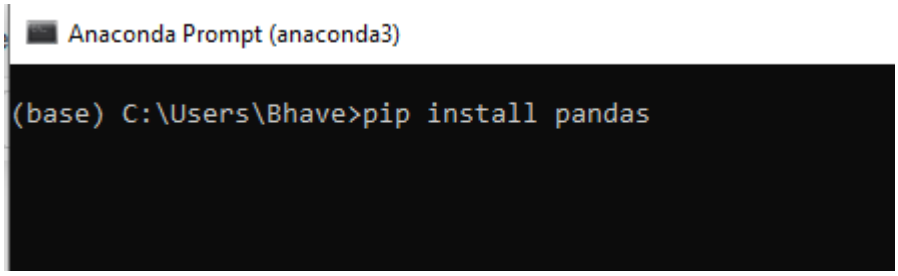
Out[29]:
```
array([[27, 30, 33],
       [36, 39, 42],
       [45, 48, 51]])
```

In [30]:
```
mat.std()
```

Out[30]:
```
7.211102550927978
```

PANDAS LIBRARY

INSTALLING PANDAS



In [31]:
```
import pandas as pd
```

DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language.

A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.)

The DataFrame has both a row and column index

In [32]:
```
from numpy.random import randn
np.random.seed(101)
```

There are many ways to construct a DataFrame, though one of the most common is from a dict of equal-length lists or NumPy arrays

In [36]:
```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
       'popu': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
```

In [37]:
```
frame = pd.DataFrame(data)
```

In [38]:
```
frame
```

Out[38]:

	state	year	popu
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

For large DataFrames, the head method selects only the first five rows:

In [39]:
```
frame.head()
```

Out[39]:

	state	year	popu
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

In [40]:
```
frame.tail()
```

Out[40]:

	state	year	popu
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

In [41]:
```
pd.DataFrame(data, columns=['year', 'state', 'popu'])
```

Out[41]:

	year	state	popu
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

In [43]:
```
frame.columns
```

Out[43]:
```
Index(['state', 'year', 'popu'], dtype='object')
```

Pull one state column Method: 1

In [44]:
```
frame.state
```

Out[44]:
```
0      Ohio
1      Ohio
2      Ohio
3    Nevada
4    Nevada
5    Nevada
Name: state, dtype: object
```

In [45]:
```
type(frame.state)
```

Out[45]:
```
pandas.core.series.Series
```

Pull one state column Method: 2

In [ ]:
```
frame2['state']
```

Pull (index) out multiple columns

In [47]:
```
frame[['state','year']]
```

Out[47]:

	state	year
0	Ohio	2000
1	Ohio	2001
2	Ohio	2002
3	Nevada	2001
4	Nevada	2002
5	Nevada	2003

In [48]:
```
type(frame[['state','year']])
```

Out[48]:
```
pandas.core.frame.DataFrame
```

In [ ]:
```
### Removing row and column
```

In [49]:
```
frame2 = pd.DataFrame(data, columns=['year', 'state', 'popu', 'debt'],
                      index=['one', 'two', 'three', 'four',
                             'five', 'six'])
```

In [50]:
```
frame2
```

Out[50]:

	year	state	popu	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

Drop row with row name six

its not permanent

In [51]:
```
frame2.drop('six')
```

Out[51]:

	year	state	popu	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN

In [52]:
```
frame2
```

Out[52]:

	year	state	popu	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

for permanent removal use inplace =True

In [53]:
```
frame2.drop('six',inplace=True)
```

In [54]:
```
frame2
```

Out[54]:

	year	state	popu	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN

Removing Columns

In [55]:
```
frame2.drop('debt',axis=1)
```

Out[55]:

	year	state	popu
one	2000	Ohio	1.5
two	2001	Ohio	1.7
three	2002	Ohio	3.6
four	2001	Nevada	2.4
five	2002	Nevada	2.9

the empty 'debt' column could be assigned a scalar value or an array of values:

In [57]:
```
frame2['debt'] = [12.2,13.0,14.2,14.2,15.0]
```

In [58]:
```
frame2
```

Out[58]:

	year	state	popu	debt
one	2000	Ohio	1.5	12.2
two	2001	Ohio	1.7	13.0
three	2002	Ohio	3.6	14.2
four	2001	Nevada	2.4	14.2
five	2002	Nevada	2.9	15.0

INDEXING METHOD : to take subset out

In [59]:
```
frame2.loc[['three','four'],['state','popu','debt']]
```

Out[59]:

	state	popu	debt
three	Ohio	3.6	14.2
four	Nevada	2.4	14.2

using numerical index numbering

In [60]:
```
frame2.iloc[3:5,1:3]
```

Out[60]:

	state	popu
four	Nevada	2.4
five	Nevada	2.9

Read CSV file

SF Salaries Exercise

Welcome to a quick exercise for you to practice your pandas skills! We will be using the SF Salaries Dataset from Kaggle! Just follow along and complete the tasks outlined in bold below. The tasks will get harder and harder as you go along.

Read Salaries.csv as a dataframe called sal.

In [62]:
```
sal=pd.read_csv('Salaries.csv')
```

In [63]:
```
sal.head()
```

Out[63]:

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay
0	1	NATHANIEL FORD	GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY	167411.18	0.00	400184.25	NaN	567595.43
1	2	GARY JIMENEZ	CAPTAIN III (POLICE DEPARTMENT)	155966.02	245131.88	137811.38	NaN	538909.28
2	3	ALBERT PARDINI	CAPTAIN III (POLICE DEPARTMENT)	212739.13	106088.18	16452.60	NaN	335279.91
3	4	CHRISTOPHER CHONG	WIRE ROPE CABLE MAINTENANCE MECHANIC	77916.00	56120.71	198306.90	NaN	332343.61
4	5	PATRICK GARDNER	DEPUTY CHIEF OF DEPARTMENT, (FIRE DEPARTMENT)	134401.60	9737.00	182234.59	NaN	326373.19

In [64]:
```
sal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148654 entries, 0 to 148653
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Id                148654 non-null  int64  
 1   EmployeeName      148654 non-null  object 
 2   JobTitle          148654 non-null  object 
 3   BasePay           148045 non-null  float64
 4   OvertimePay       148650 non-null  float64
 5   OtherPay          148650 non-null  float64
 6   Benefits          112491 non-null  float64
 7   TotalPay          148654 non-null  float64
 8   TotalPayBenefits  148654 non-null  float64
 9   Year              148654 non-null  int64  
 10  Notes             0 non-null       float64
 11  Agency            148654 non-null  object 
 12  Status            0 non-null       float64
dtypes: float64(8), int64(2), object(3)
memory usage: 14.7+ MB
```

Notes and status column has too many NULL values we wll drop them

In [66]:
```
salary=sal.copy()
salary=salary.drop(['Notes','Status'],axis=1)
```

In [67]:
```
salary.head()
```

Out[67]:

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay
0	1	NATHANIEL FORD	GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY	167411.18	0.00	400184.25	NaN	567595.43
1	2	GARY JIMENEZ	CAPTAIN III (POLICE DEPARTMENT)	155966.02	245131.88	137811.38	NaN	538909.28
2	3	ALBERT PARDINI	CAPTAIN III (POLICE DEPARTMENT)	212739.13	106088.18	16452.60	NaN	335279.91
3	4	CHRISTOPHER CHONG	WIRE ROPE CABLE MAINTENANCE MECHANIC	77916.00	56120.71	198306.90	NaN	332343.61
4	5	PATRICK GARDNER	DEPUTY CHIEF OF DEPARTMENT, (FIRE DEPARTMENT)	134401.60	9737.00	182234.59	NaN	326373.19

Q1.What is the average BasePay ?

total null values present in BasePay column

In [68]:
```
salary['BasePay'].isnull().sum()
```

Out[68]: 609

all aggregating methods skip the missing value within the column

In [69]:
```
Avg_basePay=sal_1['BasePay'].mean()
print(Avg_basePay)
```

```
66325.44884050643
```

Q2. What is the highest amount of OvertimePay in the dataset ?

In [70]:
```
salary['OvertimePay'].max()
```

Out[70]: 245131.88

Q3.What is the job title of JOSEPH DRISCOLL ? Note: Use all caps, otherwise you may get an answer that doesn't match up (there is also a lowercase Joseph Driscoll).

In [71]:
```
salary[salary['EmployeeName']=='JOSEPH DRISCOLL'][['EmployeeName','JobTitle']]
```

Out[71]:

	EmployeeName	JobTitle
24	JOSEPH DRISCOLL	CAPTAIN, FIRE SUPPRESSION

Q4.What is the name of lowest paid person (including benefits)?

In [72]:
```
salary[sal_1['TotalPayBenefits']== salary['TotalPayBenefits'].min()]
```

Out[72]:

	Id	EmployeeName	JobTitle	BasePay	OvertimePay	OtherPay	Benefits	TotalPay
148653	148654	Joe Lopez	Counselor, Log Cabin Ranch	0.0	0.0	-618.13	0.0	-618.13

Do you notice something strange about how much he or she is paid?

Q5. What was the average (mean) BasePay of all employees per year? (2011-2014) ?

In [73]:
```
sal_1.groupby('Year').mean().BasePay
```

Out[73]:
```
Year
2011    63595.956517
2012    65436.406857
2013    69630.030216
2014    66564.421924
Name: BasePay, dtype: float64
```

How many unique job titles are there?

In [74]:
```
salary['JobTitle'].nunique()
```

Out[74]:
```
2159
```

What are the top 5 most common jobs?

In [75]:
```
salary['JobTitle'].value_counts().head(5)
```

Out[75]:
```
Transit Operator                7036
Special Nurse                   4389
Registered Nurse                3736
Public Svc Aide-Public Works    2518
Police Officer 3                2421
Name: JobTitle, dtype: int64
```

How many Job Titles were represented by only one person in 2013? (e.g. Job Titles with only one occurence in 2013?)

In [76]:
```
sum(sal_1[sal_1['Year']==2013]['JobTitle'].value_counts() == 1)
```

Out[76]: 202

How many people have the word Chief in their job title? (A little tricky)

METHOD: 1

In [79]:
```
contain=[]
title_l=list(salary['JobTitle'])

for title in title_l:
    if 'chief' in title.lower():
        contain.append(True)
print(len(contain))
```

627

METHOD: 2

In [83]:
```
sum(salary['JobTitle'].apply(lambda x: 'chief' in x.lower()))
```

Out[83]: 627

GREAT JOB !

In [ ]:
```

```