1.data analysing: analysing the data. the process of analysing the large data set. 2. data set: collection of information or values which are particular to onr subject or group 3. data sci: processing large data

In [3]:

```
import pandas as pd
df=pd.read_csv("crimeJ.csv")
df.head() #SAMPLE VALUES FROM TOP
```

Out[3]:

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
0	1/1/06 0:00	3108 OCCIDENTAL DR	3	3C	1115	10851(A)VC TAKE VEH W/O OWNER	2404	38.550420	-121.391416
1	1/1/06 0:00	2082 EXPEDITION WAY	5	5A	1512	459 PC BURGLARY RESIDENCE	2204	38.473501	-121.490186
2	1/1/06 0:00	4 PALEN CT	2	2A	212	10851(A)VC TAKE VEH W/O OWNER	2404	38.657846	-121.462101
3	1/1/06 0:00	22 BECKFORD CT	6	6C	1443	476 PC PASS FICTICIOUS CHECK	2501	38.506774	-121.426951
4	1/1/06 0:00	3421 AUBURN BLVD	2	2A	508	459 PC BURGLARY-UNSPECIFIED	2299	38.637448	-121.384613

In [4]:

```
df.tail() #SAMPLE VALUES FROM BOTTOM
```

Out[4]:

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
7579	1/31/06 23:36	26TH ST / G ST	3	3B	728	594(B)(2)(A) VANDALISM/ -$400	2999	38.577832	-121.470460
7580	1/31/06 23:40	4011 FREEPORT BLVD	4	4A	957	459 PC BURGLARY BUSINESS	2203	38.537591	-121.492591
7581	1/31/06 23:41	30TH ST / K ST	3	3C	841	TRAFFIC-ACCIDENT INJURY	5400	38.572030	-121.467012
7582	1/31/06 23:45	5303 FRANKLIN BLVD	4	4B	969	3056 PAROLE VIO - I RPT	7000	38.527187	-121.471248
7583	1/31/06 23:50	COBBLE COVE LN / COBBLE SHORES DR	4	4C	1294	TRAFFIC-ACCIDENT-NON INJURY	5400	38.479628	-121.528634

In [5]:

```
df.head(3)#TO GET PARTICULAR NO.OF TOP VALUES
```

Out[5]:

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
0	1/1/06 0:00	3108 OCCIDENTAL DR	3	3C	1115	10851(A)VC TAKE VEH W/O OWNER	2404	38.550420	-121.391416
1	1/1/06 0:00	2082 EXPEDITION WAY	5	5A	1512	459 PC BURGLARY RESIDENCE	2204	38.473501	-121.490186
2	1/1/06 0:00	4 PALEN CT	2	2A	212	10851(A)VC TAKE VEH W/O OWNER	2404	38.657846	-121.462101

In [6]:

```
df.tail(2) #TO GET PARTICULAR NO.OF LAST VALUES
```

Out[6]:

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
7582	1/31/06 23:45	5303 FRANKLIN BLVD	4	4B	969	3056 PAROLE VIO - I RPT	7000	38.527187	-121.471248
7583	1/31/06 23:50	COBBLE COVE LN / COBBLE SHORES DR	4	4C	1294	TRAFFIC-ACCIDENT-NON INJURY	5400	38.479628	-121.528634

In [7]:

```
df.sort_values("district") #SORT BY DISTRICT ORDER
```

Out[7]:

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
4367	1/18/06 22:00	1671 W EL CAMINO AVE	1	1B	434	10851(A)VC TAKE VEH W/O OWNER	2404	38.613539	-121.499806
1425	1/7/06 0:30	4601 BLACKROCK DR	1	1A	136	459 PC BURGLARY VEHICLE	2299	38.657333	-121.495213
1420	1/7/06 0:01	5948 MEEKS WAY	1	1A	106	1708 US THEFT OF MAIL	2310	38.683789	-121.496349
1411	1/7/06 0:00	3260 NORDYKE DR	1	1C	418	459 PC BURGLARY VEHICLE	2299	38.626506	-121.476455
1409	1/7/06 0:00	3410 SHADOW TREE DR	1	1B	401	10851(A)VC TAKE VEH W/O OWNER	2404	38.628525	-121.505960
...	...	...	...	...	...	...	...	...	...
3108	1/13/06 18:00	3264 RAMONA AVE	6	6C	1113	10851(A)VC TAKE VEH W/O OWNER	2404	38.545224	-121.415719
3105	1/13/06 18:00	15 CARTHAGE CT	6	6C	1443	MISSING PERSON	7000	38.505769	-121.418114
6430	1/27/06 9:45	8151 POWER RIDGE RD	6	6C	1145	10851 VC AUTO THEFT LOCATE	2404	38.531579	-121.407542
1553	1/7/06 16:10	3946 2ND AVE	6	6A	1014	459 PC BURGLARY RESIDENCE	2204	38.550676	-121.461860
3791	1/16/06 22:00	5961 13TH AVE	6	6B	1056	459 PC BURGLARY VEHICLE	2299	38.540424	-121.436176

7584 rows × 9 columns

In [8]:

```
df["grid"].mean() #MEAN
```

Out[8]:

916.2507911392405

In [9]:

```
df["grid"].max() #MAX VALUE
```

Out[9]:

1661

In [10]:

```
df["grid"].min() #MIN VALUE
```

Out[10]:

102

In [11]:

```
df.describe() #DESCRIBE
```

Out[11]:

	district	grid	ucr_ncic_code	latitude	longitude
count	7584.000000	7584.000000	7584.000000	7584.000000	7584.000000
mean	3.574631	916.250791	4275.068829	38.559809	-121.463832
std	1.642512	407.436310	2171.593193	0.056101	0.034621
min	1.000000	102.000000	909.000000	38.437999	-121.555832
25%	2.000000	567.000000	2309.000000	38.518476	-121.489543
50%	3.000000	899.000000	3532.000000	38.559523	-121.465459
75%	5.000000	1264.000000	7000.000000	38.610361	-121.435947
max	6.000000	1661.000000	8102.000000	38.683789	-121.365238

In [12]:

```
df['beat'] #COLOUMN
```

Out[12]:

```
0       3C
1       5A
2       2A
3       6C
4       2A
        ...
7579    3B
7580    4A
7581    3C
7582    4B
7583    4C
Name: beat, Length: 7584, dtype: object
```

In [13]:

```
df[['district','address']]#COLOUMN SLICING
```

Out[13]:

	district	address
0	3	3108 OCCIDENTAL DR
1	5	2082 EXPEDITION WAY
2	2	4 PALEN CT
3	6	22 BECKFORD CT
4	2	3421 AUBURN BLVD
...	...	...
7579	3	26TH ST / G ST
7580	4	4011 FREEPORT BLVD
7581	3	30TH ST / K ST
7582	4	5303 FRANKLIN BLVD
7583	4	COBBLE COVE LN / COBBLE SHORES DR

7584 rows × 2 columns

In [15]:

```
#TO GET INFORMATION OF THE DATA TYPE
df=pd.read_csv('crimeJ.csv')
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 7584 entries, 0 to 7583
Data columns (total 9 columns):
cdatetime        7584 non-null object
address          7584 non-null object
district         7584 non-null int64
beat             7584 non-null object
grid             7584 non-null int64
crimedescr       7584 non-null object
ucr_ncic_code    7584 non-null int64
latitude         7584 non-null float64
longitude        7584 non-null float64
dtypes: float64(2), int64(3), object(4)
memory usage: 533.4+ KB
```

Out[15]:

	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
0	1/1/06 0:00	3108 OCCIDENTAL DR	3	3C	1115	10851(A)VC TAKE VEH W/O OWNER	2404	38.550420	-121.391416
1	1/1/06 0:00	2082 EXPEDITION WAY	5	5A	1512	459 PC BURGLARY RESIDENCE	2204	38.473501	-121.490186
2	1/1/06 0:00	4 PALEN CT	2	2A	212	10851(A)VC TAKE VEH W/O OWNER	2404	38.657846	-121.462101
3	1/1/06 0:00	22 BECKFORD CT	6	6C	1443	476 PC PASS FICTICIOUS CHECK	2501	38.506774	-121.426951
4	1/1/06 0:00	3421 AUBURN BLVD	2	2A	508	459 PC BURGLARY-UNSPECIFIED	2299	38.637448	-121.384613

In [16]:

```
#LISTS
import pandas as pd
name=['elena','bonnie','jake','ian','paul']
df=pd.DataFrame(name)
df
```

Out[16]:

	0
0	elena
1	bonnie
2	jake
3	ian
4	paul

In [17]:

```
#column and row acccess using list
data=[['elena',30],['ian',40],['paul',37]]
pd.DataFrame(data,index=[1,2,3],columns=['name','age'])
```

Out[17]:

	name	age
1	elena	30
2	ian	40
3	paul	37

In [18]:

```
#DICTIONARIES
import pandas as pd
food={'name':['pizza','noodles','biriyani','ice cream'],'price':[225,200,270,110]}
df=pd.DataFrame(food,index=[1,2,3,4])
```

```
df=pd.DataFrame(food,index=[1,2,3,4])
df
```

Out[18]:

	name	price
1	pizza	225
2	noodles	200
3	biriyani	270
4	ice cream	110

In [19]:

```
#TO DELETE A ROW
df.drop(3)
```

Out[19]:

	name	price
1	pizza	225
2	noodles	200
4	ice cream	110

In [20]:

```
#TO DELETE A COLUMN
del df['price']
df
```

Out[20]:

	name
1	pizza
2	noodles
3	biriyani
4	ice cream

In [22]:

```
 #cleaning data
import pandas as pd
exam={'s1':[15,13,20],'s2':[12,17,10],'s3':[13,15,19]}# CREATING NESTED DICTIONARY
df=pd.DataFrame(exam,index=['elena','damon','stefan'])# GIVING NAME TO THR ROWS
df
```

Out[22]:

	s1	s2	s3
elena	15	12	13
damon	13	17	15
stefan	20	10	19

In [23]:

```
#reindex() function conform DataFrame to new index with optional filling logic
df=df.reindex(['elena','damon','stefan','jermy'])
df
```

Out[23]:

	s1	s2	s3
elena	15.0	12.0	13.0
damon	13.0	17.0	15.0
stefan	20.0	10.0	19.0
jermy	NaN	NaN	NaN

In [24]:

```
 #fillna allows the user replace NaN values with some value of their own
df.fillna('absent')
```

Out[24]:

	s1	s2	s3
elena	15	12	13
damon	13	17	15
stefan	20	10	19
jermy	absent	absent	absent

In [25]:

```
#dropna used to manage and remove Null values from a data frame
df.dropna()
```

Out[25]:

	s1	s2	s3
elena	15.0	12.0	13.0
damon	13.0	17.0	15.0
stefan	20.0	10.0	19.0

In [26]:

```
# to replace a value
df.replace({19:100,15:200})
```

Out[26]:

	s1	s2	s3
elena	200.0	12.0	13.0
damon	13.0	17.0	200.0
stefan	20.0	10.0	100.0
jermy	NaN	NaN	NaN

In [27]:

```
# groupby:
import pandas as pd
df = pd.read_csv("ndata.csv")
df
```

Out[27]:

	name	age	salary
0	elena	17	12000

	name	age	salary
1	damon	19	15000
2	elena	17	30000
3	damon	19	45000
4	stafan	20	50000

In [28]:

```
name=df.groupby('name')
```

In [29]:

```
name
```

Out[29]:

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000149D1186CC8>

In [30]:

```
name.mean()# grouping by name
```

Out[30]:

name	age	salary
damon	19	30000
elena	17	21000
stafan	20	50000

In [31]:

```
name.max()# groupby max
```

Out[31]:

name	age	salary
damon	19	45000
elena	17	30000
stafan	20	50000

In [32]:

```
name.min()# groupby min
```

Out[32]:

name	age	salary
damon	19	15000
elena	17	12000
stafan	20	50000

In [33]:

```
name.sum()
```

Out[33]:

	age	salary
name		
damon	38	60000
elena	34	42000
stafan	20	50000

In [34]:

```
name.std
```

Out[34]:

```
<bound method GroupBy.std of <pandas.core.groupby.generic.DataFrameGroupBy object at
0x00000149D1186CC8>>
```

In [35]:

```
name.count()
```

Out[35]:

	age	salary
name		
damon	2	2
elena	2	2
stafan	1	1

In [36]:

```
name.describe()
```

Out[36]:

	age								salary							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
name																
damon	2.0	19.0	0.0	19.0	19.0	19.0	19.0	19.0	2.0	30000.0	21213.203436	15000.0	22500.0	30000.0	37500.0	45000.0
elena	2.0	17.0	0.0	17.0	17.0	17.0	17.0	17.0	2.0	21000.0	12727.922061	12000.0	16500.0	21000.0	25500.0	30000.0
stafan	1.0	20.0	NaN	20.0	20.0	20.0	20.0	20.0	1.0	50000.0	NaN	50000.0	50000.0	50000.0	50000.0	50000.0

In [37]:

```
name.describe().transpose()
```

Out[37]:

	name	damon	elena	stafan
age	count	2.000000	2.000000	1.0
	mean	19.000000	17.000000	20.0
	std	0.000000	0.000000	NaN
	min	19.000000	17.000000	20.0
	25%	19.000000	17.000000	20.0
	50%	19.000000	17.000000	20.0
	75%	19.000000	17.000000	20.0
	max	19.000000	17.000000	20.0

	name	damon	elena	stafan
salary	count	2.000000	2.000000	1.0
	mean	30000.000000	21000.000000	50000.0
	std	21213.203436	12727.922061	NaN
	min	15000.000000	12000.000000	50000.0
	25%	22500.000000	16500.000000	50000.0
	50%	30000.000000	21000.000000	50000.0
	75%	37500.000000	25500.000000	50000.0
	max	45000.000000	30000.000000	50000.0

In [38]:

```
#THREE SHEETS IN ONE EXCEL FILE:
import pandas as pd
d1={'v':['stefan','ian','paul','damon'],
     'm1':[4,2,3,5]}
d2={'h':['jermy','elena','nina','matt'],
    'm2':[3,5,3,4]}
d3={'w':['bonnie','cal','ben','mick'],
    'm3':[1,3,5,2]}
data1=pd.DataFrame(d1)
data2=pd.DataFrame(d2)
data3=pd.DataFrame(d3)
with pd.ExcelWriter('assignment.xlsx') as w:
    data1.to_excel(w,sheet_name='sheet1')
    data2.to_excel(w,sheet_name='sheet2')
    data3.to_excel(w,sheet_name='sheet3')
```

In [39]:

```
data1
```

Out[39]:

	v	m1
0	stefan	4
1	ian	2
2	paul	3
3	damon	5

In [40]:

```
data2
```

Out[40]:

	h	m2
0	jermy	3
1	elena	5
2	nina	3
3	matt	4

In [41]:

```
data3
```

Out[41]:

	w	m3
0	bonnie	1

1	w	m3
2	ben	5
3	mick	2

In [42]:

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
v      4 non-null object
m1     4 non-null int64
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
```

In [43]:

```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
h      4 non-null object
m2     4 non-null int64
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
```

In [44]:

```
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
w      4 non-null object
m3     4 non-null int64
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
```

In [45]:

```
data1.describe()
```

Out[45]:

	m1
count	4.000000
mean	3.500000
std	1.290994
min	2.000000
25%	2.750000
50%	3.500000
75%	4.250000
max	5.000000

In [46]:

```
data2.describe()
```

Out[46]:

	m2
count	4.000000
mean	3.750000
std	0.957427
min	3.000000
25%	3.000000
50%	3.500000
75%	4.250000
max	5.000000

In [47]:

```
data3.describe()
```

Out[47]:

	m3
count	4.000000
mean	2.750000
std	1.707825
min	1.000000
25%	1.750000
50%	2.500000
75%	3.500000
max	5.000000

In [48]:

```
# data scaling:
## Load digit
from sklearn.datasets import load_digits
from sklearn import preprocessing
digits = load_digits()
print(digits.data.shape)
```

```
(1797, 64)
```

In [49]:

```
x1=digits.data
y1=digits.target
n1=preprocessing.normalize(x1)
print(n1)
```

```
[[0.         0.         0.09024036 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.15413829 0.         0.        ]
 [0.         0.         0.         ... 0.24153867 0.1358655  0.        ]
 ...
 [0.         0.         0.0140138  ... 0.08408278 0.         0.        ]
 [0.         0.         0.03044313 ... 0.18265877 0.         0.        ]
 [0.         0.         0.14230641 ... 0.17076769 0.01423064 0.        ]]
```

In [50]:

```
preprocessing.scale(x1)
```

Out[50]:

```
array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
```

```
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
         1.6951369 , -0.19600752],
       ...,
       [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
        -0.26113572, -0.19600752]])
```

NUMPY

In [66]:

```
import numpy as np #importing the numpy pakeage
```

In [67]:

```
arr=[1,2,3,5,6,9] #creating a list of numbers
np.array(arr) #THIS IS TO PRINT THE LIST AS A 1D ARRAY
```

Out[67]:

```
array([1, 2, 3, 5, 6, 9])
```

In [68]:

```
mat=[[1,3,5],[2,4,6],[7,8,9]]# CREATING USING A NESTED LIST
np.array(mat)# THIS IS TO PRINT THE LIST AS A 2D ARRAY
```

Out[68]:

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

In [69]:

```
a=np.arange(0,20,2) #np.arange(start,stop,step)to return evenly spaced values within a given
intervals
```

In [70]:

```
a
```

Out[70]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [71]:

```
# IT RETURNS A ARRAY OF GIVEN SHAPE AND TYPE WITH ZEROS
np.zeros(5)#1D ARRAY
```

Out[71]:

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

In [72]:

```
np.zeros((2,2))#2D ARRAY
```

Out[72]:

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

In [73]:

```
# IT RETURNS A ARRAY OF GIVEN SHAPE AND TYPE WITH ONES
np.ones(3)#1D ARRAY
```

Out[73]:

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

In [74]:

```
np.ones((3,3))# 2D ARRAY
```

Out[74]:

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

In [75]:

```
#Return a matrix having 1's on the diagonal and 0's elsewhere(identity matrix)
np.eye(4,4)
```

Out[75]:

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

In [76]:

```
#numpy. linspace (start, stop, num=50)
#creating numeric sequences,similar to arange function.
#it prints the linearly spaced vectors
np.linspace(0,5,10)
```

Out[76]:

```
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

In [77]:

```
# creates an array of specified shape and fills it with positive random values.
np.random.rand(5)
```

Out[77]:

```
array([0.43237861, 0.55843269, 0.49656912, 0.56702275, 0.71339312])
```

In [78]:

```
#creates an array of specified shape and fills it with random values both positive and negative.
np.random.randn(4)
```

Out[78]:

```
array([-1.03105556, -2.06307735, -0.49439508,  0.07424895])
```

In [79]:

```
#It returns an array of specified shape and fills it with random integers from low to high
np.random.randint(1,10,5)
```

Out[79]:

```
array([7, 4, 8, 1, 9])
```

In [80]:

```
#reshape function is used to give a new shape to an array without changing its data.J
a.reshape(5,2)
```

Out[80]:

```
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10],
       [12, 14],
       [16, 18]])
```

In [81]:

```
a.max()# prints the maximum value
```

Out[81]:

```
18
```

In [82]:

```
a.argmax()# prints the index of the maximum value
```

Out[82]:

```
9
```

In [83]:

```
a.min()# print the minimum value
```

Out[83]:

```
0
```

In [84]:

```
a.argmin()#print the index of the minimum value
```

Out[84]:

```
0
```

In [85]:

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

Out[85]:

```
(10,)
```

In [86]:

```
a.dtype
```

Out[86]:

```
dtype('int32')
```

In [87]:

```
# numpy indexing and selection
import numpy as np
```

```python
import numpy as np
```

In [88]:

```python
arr=np.arange(10)
```

In [89]:

```python
arr
```

Out[89]:

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

In [90]:

```python
arr[2:7] #indexing the list from 2-6 leaving 7
```

Out[90]:

```
array([2, 3, 4, 5, 6])
```

In [91]:

```python
arr>2 #to get a boolean value (i.e) numbers greater than 2 is TRUE, while others are FALSE
```

Out[91]:

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

In [92]:

```python
arr[arr>2] #TO print all the numbers greater than two
```

Out[92]:

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

In [93]:

```python
# 2d array
import  numpy as np
```

In [94]:

```python
mat=np.array([[1,2,3],[4,5,6],[7,8,9]])
```

In [95]:

```python
mat
```

Out[95]:

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

In [96]:

```python
mat[0] #TO PRINT THE FIRST ROW
```

Out[96]:

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

In [97]:

```
mat[1,2] #TO PRINT THE ELEMENT IN THE 1ST ROW,2ND COL
```

Out[97]:

```
6
```

In [98]:

```
mat[:2,2:]
```

Out[98]:

```
array([[3],
       [6]])
```

In [99]:

```
mat[1:,1:]
```

Out[99]:

```
array([[5, 6],
       [8, 9]])
```

In [100]:

```
# numpy arithmetic operations
import numpy as np
```

In [101]:

```
arr=np.arange(0,20)
```

In [102]:

```
arr
```

Out[102]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [103]:

```
arr+arr#diff of the arr
```

Out[103]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38])
```

In [104]:

```
arr-arr #diff of the arr
```

Out[104]:

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

In [105]:

```
arr*arr #pdt of the arr
```

Out[105]:

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
```

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225, 256, 289, 324, 361])
```

In [106]:

```python
arr+100 # adding 100 to arr
```

Out[106]:

```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
       113, 114, 115, 116, 117, 118, 119])
```

In [107]:

```python
arr-100 #subrating 100 from arr
```

Out[107]:

```
array([-100,  -99,  -98,  -97,  -96,  -95,  -94,  -93,  -92,  -91,  -90,
        -89,  -88,  -87,  -86,  -85,  -84,  -83,  -82,  -81])
```

In [108]:

```python
arr*2 # multiplying 100 to arr
```

Out[108]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38])
```

In [109]:

```python
arr**2 #squaring the arr
```

Out[109]:

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225, 256, 289, 324, 361], dtype=int32)
```

In [110]:

```python
np.sqrt(arr) #sqr root of arr
```

Out[110]:

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ,
       3.16227766, 3.31662479, 3.46410162, 3.60555128, 3.74165739,
       3.87298335, 4.        , 4.12310563, 4.24264069, 4.35889894])
```

In [111]:

```python
np.exp(arr) #exponential of arr
```

Out[111]:

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03, 2.20264658e+04, 5.98741417e+04,
       1.62754791e+05, 4.42413392e+05, 1.20260428e+06, 3.26901737e+06,
       8.88611052e+06, 2.41549528e+07, 6.56599691e+07, 1.78482301e+08])
```

In [112]:

```python
np.max(arr) # max value of arr
```

Out[112]:

```
19
```

In [113]:

```
np.min(arr) #min value of arr
```

Out[113]:

```
0
```

In [114]:

```
np.sin(arr) # sine values of arr
```

Out[114]:

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
       -0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736,
        0.65028784, -0.28790332, -0.96139749, -0.75098725,  0.14987721])
```

In [115]:

```
np.cos(arr)# cosine values of arr
```

Out[115]:

```
array([ 1.        ,  0.54030231, -0.41614684, -0.9899925 , -0.65364362,
        0.28366219,  0.96017029,  0.75390225, -0.14550003, -0.91113026,
       -0.83907153,  0.0044257 ,  0.84385396,  0.90744678,  0.13673722,
       -0.75968791, -0.95765948, -0.27516334,  0.66031671,  0.98870462])
```

In [116]:

```
np.tan(arr)# tan values of arr
```

Out[116]:

```
array([ 0.00000000e+00,  1.55740772e+00, -2.18503986e+00, -1.42546543e-01,
        1.15782128e+00, -3.38051501e+00, -2.91006191e-01,  8.71447983e-01,
       -6.79971146e+00, -4.52315659e-01,  6.48360827e-01, -2.25950846e+02,
       -6.35859929e-01,  4.63021133e-01,  7.24460662e+00, -8.55993401e-01,
        3.00632242e-01,  3.49391565e+00, -1.13731371e+00,  1.51589471e-01])
```

In [117]:

```
np.log(arr)# log values of arr
```

```
C:\Users\Anbu Nambi\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: divide by
zero encountered in log
  """Entry point for launching an IPython kernel.
```

Out[117]:

```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458,
       2.30258509, 2.39789527, 2.48490665, 2.56494936, 2.63905733,
       2.7080502 , 2.77258872, 2.83321334, 2.89037176, 2.94443898])
```

MATPLOTLIB

In [126]:

```
# functional method:
```

In [127]:

```
import matplotlib.pyplot as plt #importing the matplotlib package
```

In [128]:

```
%matplotlib inline
# this is to see the plots created inside the jupyter notebook
```

In [129]:

```
import numpy as np #importin the numpy package
```

In [130]:

```
x=np.arange(0,10) #to spaced the values evenly within a given intervals of the x axis
y=x**2 #the square of the x values
```

In [131]:

```
x
```

Out[131]:

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

In [132]:

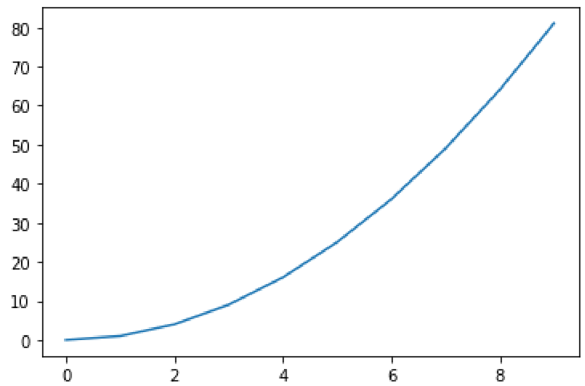
```
y
```

Out[132]:

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81], dtype=int32)
```

In [133]:

```
plt.plot(x,y)# it is used to shows the plot
plt.show()# it is used to print the plot
```

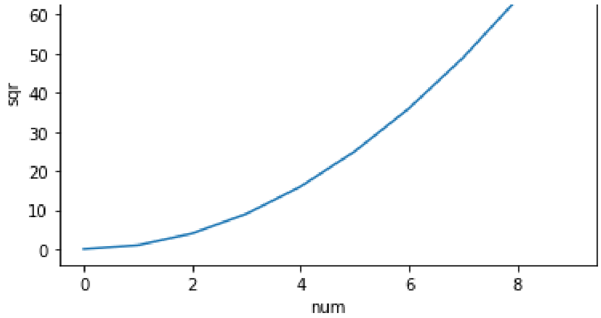


In [134]:

```
plt.plot(x,y)
plt.xlabel('num')#labeling the xaxis
plt.ylabel('sqr')#labeling the yaxis
plt.title('sample')# naming the graph
```

Out[134]:

```
Text(0.5, 1.0, 'sample')
```

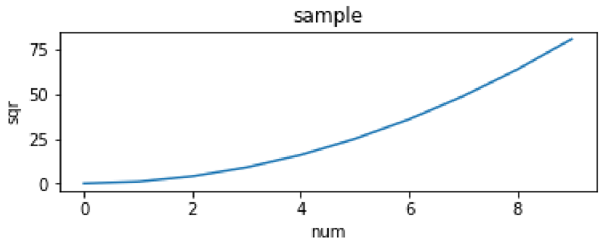


In [135]:

```
plt.subplot(2,1,1) #plt.subplot(no.of row,no.of col,ref of plot no.)
plt.plot(x,y)
plt.xlabel('num')
plt.ylabel('sqr')
plt.title('sample')
```

Out[135]:

```
Text(0.5, 1.0, 'sample')
```



In [141]:

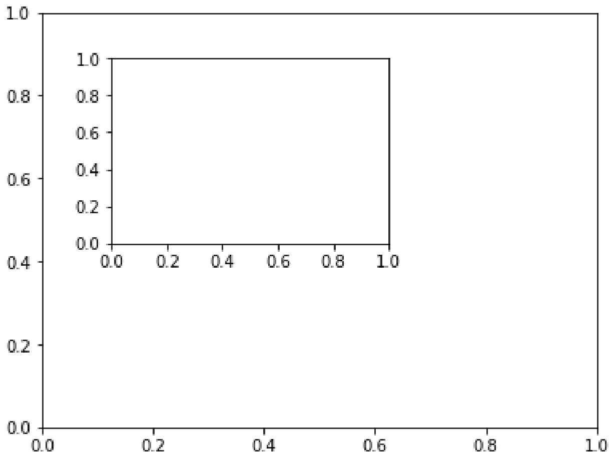
```
#object oreinted method:
```

In [142]:

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [143]:

```
fig=plt.figure()
axes1=fig.add_axes([.1,.2,.8,.9])
axes2=fig.add_axes([.2,.6,.4,.4])
```

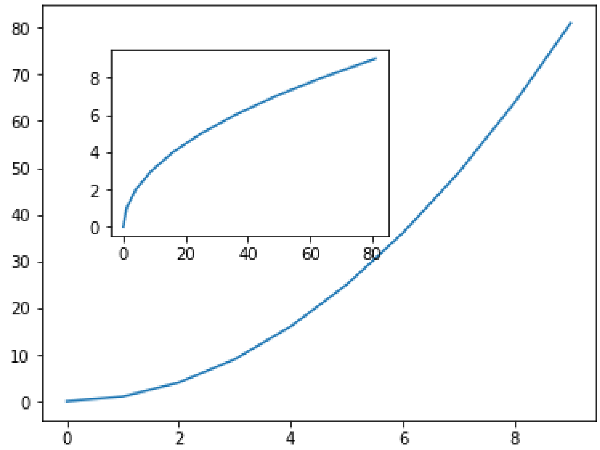


In [144]:

```
fig=plt.figure()
axes1=fig.add_axes([.1,.2,.8,.9])
axes2=fig.add_axes([.2,.6,.4,.4])
axes1.plot(x,y)
axes2.plot(y,x)
```

Out[144]:

```
[<matplotlib.lines.Line2D at 0x149d432e248>]
```

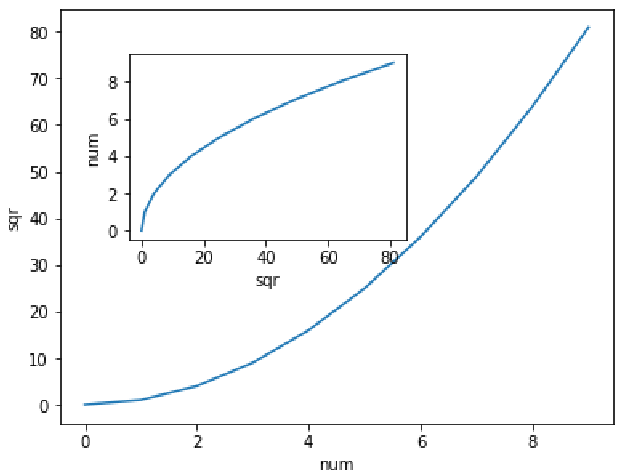


In [140]:

```
fig=plt.figure()
axes1=fig.add_axes([.1,.2,.8,.9])
axes2=fig.add_axes([.2,.6,.4,.4])
axes1.plot(x,y)
axes2.plot(y,x)
axes1.set_xlabel('num')#big graph x axis
axes1.set_ylabel('sqr')#big graph y axis
axes2.set_ylabel('num')#small graph y axis
axes2.set_xlabel('sqr')#small graph x axis
```

Out[140]:

```
Text(0.5, 0, 'sqr')
```



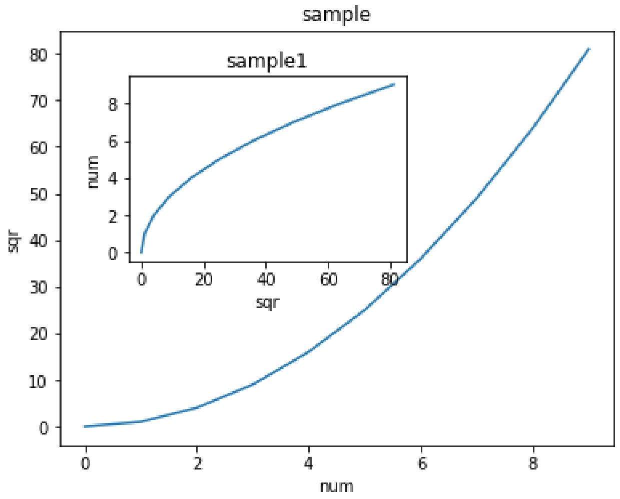
In [145]:

```
fig=plt.figure()
axes1=fig.add_axes([.1,.2,.8,.9])
axes2=fig.add_axes([.2,.6,.4,.4])
axes1.plot(x,y)
axes2.plot(y,x)
axes1.set_xlabel('num')
axes1.set_ylabel('sqr')
axes2.set_ylabel('num')
```

```
axes1.set_ylabel('num')
axes2.set_xlabel('sqr')
axes1.set_title('sample') # naming the big graph
axes2.set_title('sample1') #naming the small graph
```

Out[145]:

```
Text(0.5, 1.0, 'sample1')
```

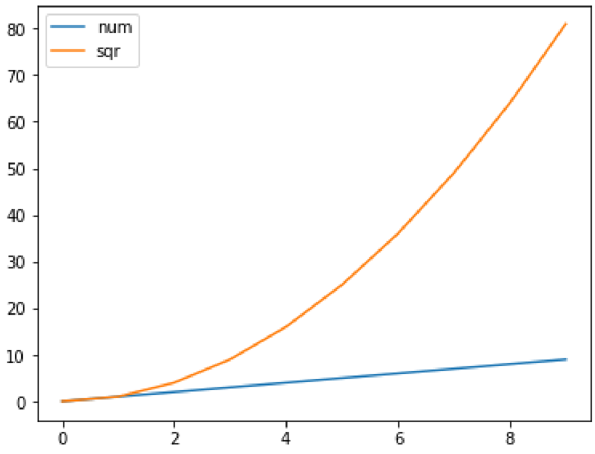


In [146]:

```
fig=plt.figure()
axes1=fig.add_axes([.1,.2,.8,.9])
axes1.plot(x,label='num')
axes1.plot(y,label='sqr')
plt.legend(loc='best')
# choosing the best location to print the key/legend of the respective graph
```

Out[146]:

```
<matplotlib.legend.Legend at 0x149d448d388>
```

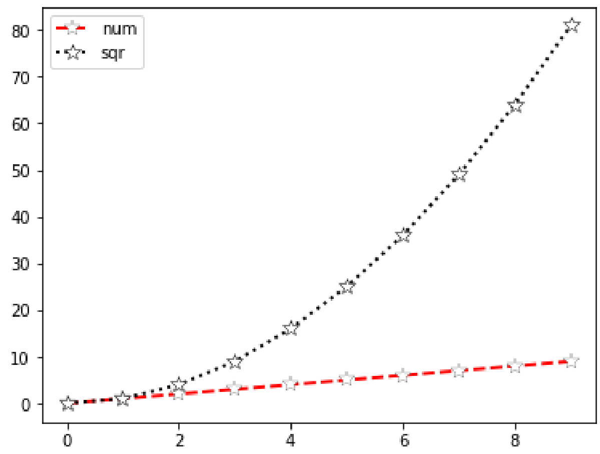


In [147]:

```
fig=plt.figure()
axes1=fig.add_axes([.1,.2,.8,.9])
axes1.plot(x,label='num',color='red',lw=2,ls='--',marker='*',markersize='10',markerfacecolor='w',ma
rkeredgecolor='k',markeredgewidth='.2',)
axes1.plot(y,label='sqr',color='black',lw=2,ls=':',marker='*',markersize='10',markerfacecolor='w',m
arkeredgecolor='k',markeredgewidth='.5',)
plt.legend(loc='best')
```

Out[147]:

```
<matplotlib.legend.Legend at 0x149d4510a48>
```



THE GRAPH CAN BE MADE MORE ATTRACTIVE AND EASILY UNDERSTOOD UDING THESE FUNCTIONS: *FROM THE ABOVE GRAPH I COMPARE THE NUMBERS AND THE SQUARE OF THE NUMBERS*LABEL: IT IS TO LABEL THE LINE I THE GRAPH *COLOUR: TO CHOOSE THE COLOR OF THE LINE*LINE WIDTH: TO SELECT THE WIDTH OF THE LINE *LINE STYLE: TO SELECT THE STYLE OF THE LINE*MARKER: TO SELECT THE SHAPE OF THE MARKER *MARKER SIZE: TO SELECT THE SIZE OF THE MARKER*MARKER FACE COLOR:TO CHOOSE THE COLOUR OF THE MARKER *MARKER EDGE COLOR:TO CHOOSE THE OUTLINE COLOR OF THE MARKER*MARKER EDGE WIDTH:TO CHOOSE THE WIDTH OF THE MARKER OUTLINE

- USING ALL THE FUNCTION THE GARPH CAN BE MADE MORE PRESENTABLE AND UNDERSTANGING.

seaborn

In [151]:

```
#distribution plots
```

In [152]:

```
import seaborn as sns
%matplotlib inline
a=sns.load_dataset('car_crashes')
a.head()
```

Out[152]:

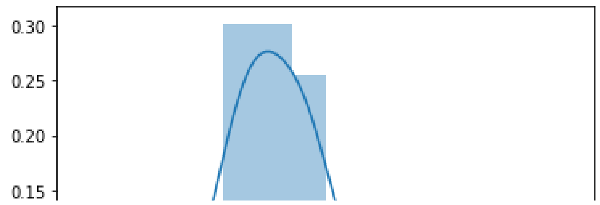
	total	speeding	alcohol	not_distracted	no_previous	ins_premium	ins_losses	abbrev
0	18.8	7.332	5.640	18.048	15.040	784.55	145.08	AL
1	18.1	7.421	4.525	16.290	17.014	1053.48	133.93	AK
2	18.6	6.510	5.208	15.624	17.856	899.47	110.35	AZ
3	22.4	4.032	5.824	21.056	21.280	827.34	142.39	AR
4	12.0	4.200	3.360	10.920	10.680	878.41	165.63	CA

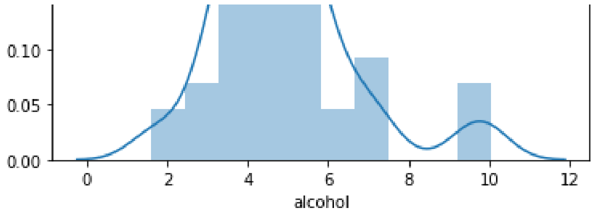
In [153]:

```
sns.distplot(a['alcohol'])
```

Out[153]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d5716f48>





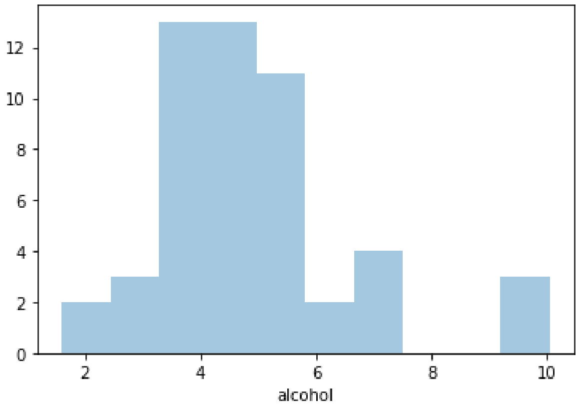
The most convenient way to take a quick look at a univariate distribution in seaborn is the distplot() function. By default, this will draw a histogram and fit a kde. if kde= false the it will not be printed, *bins is used to adjust the size of the bar according to the graph

In [154]:

```
sns.distplot(a['alcohol'], kde=False)
```

Out[154]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x149d57a7388>
```

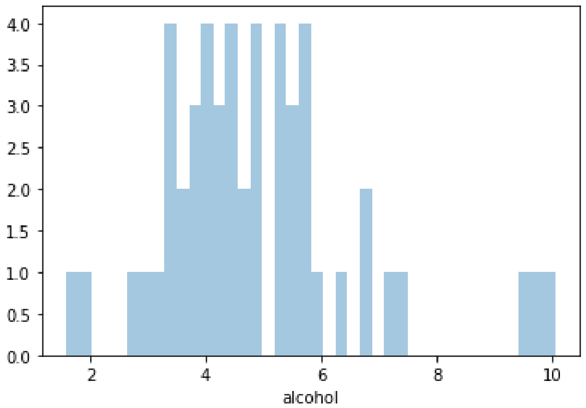


In [155]:

```
sns.distplot(a['alcohol'], kde=False,bins=40)
```

Out[155]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x149d4219408>
```



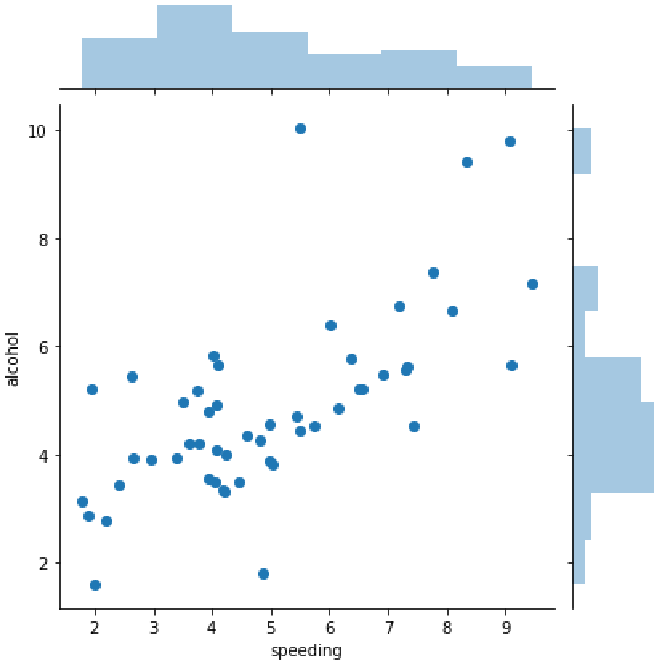
the jointplot function creates a multi-figure that shows both the joint relationship between two variables along with the marginal) distribution of each axes

In [156]:

```
sns.jointplot(x='speeding',y='alcohol',data=a)
```

Out[156]:

```
<seaborn.axisgrid.JointGrid at 0x149d407d8c8>
```

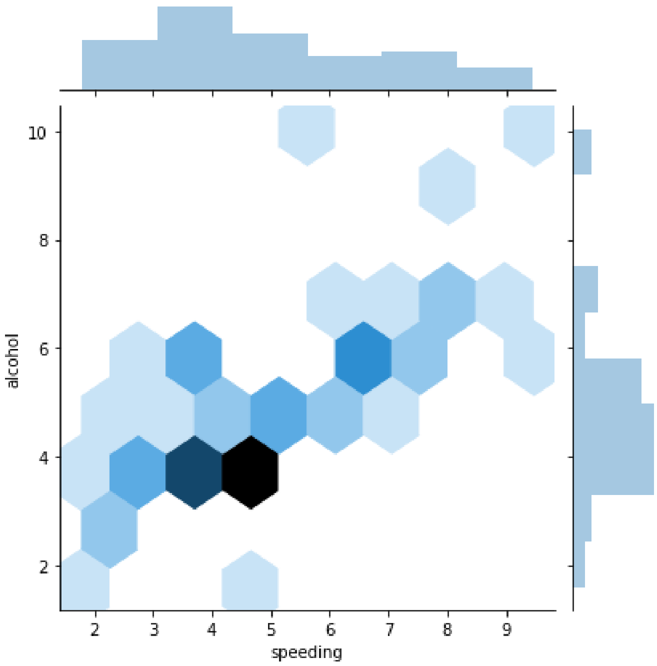


In [157]:

```
sns.jointplot(x='speeding',y='alcohol',data=a,kind='hex')
```

Out[157]:

<seaborn.axisgrid.JointGrid at 0x149d58b2b08>

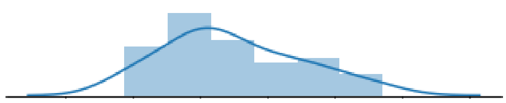


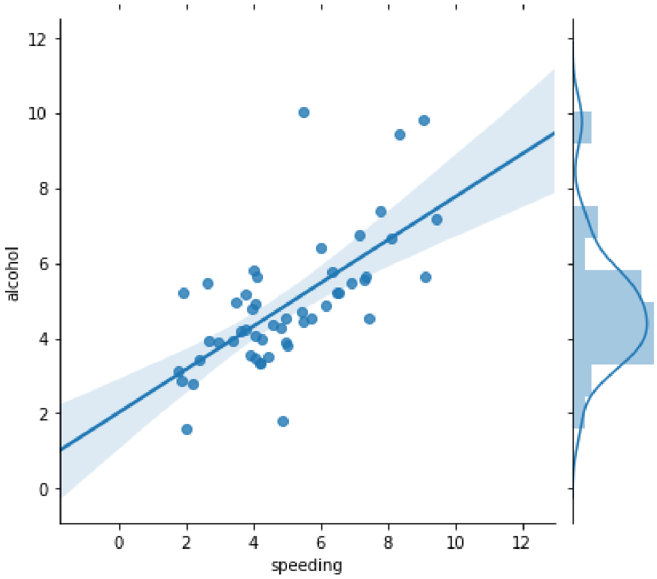
In [158]:

```
sns.jointplot(x='speeding',y='alcohol',data=a,kind='reg')
```

Out[158]:

<seaborn.axisgrid.JointGrid at 0x149d5a096c8>



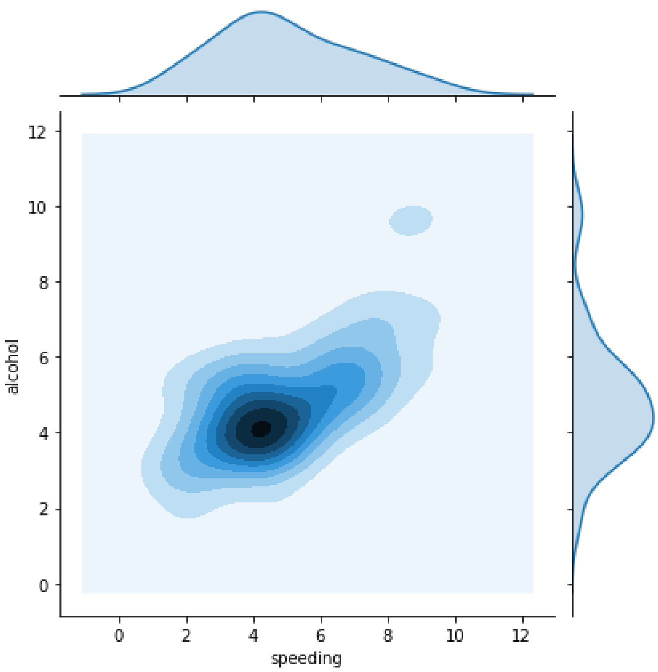


In [159]:

```
sns.jointplot(x='speeding',y='alcohol',data=a,kind='kde')
```

Out[159]:

<seaborn.axisgrid.JointGrid at 0x149d59f9a48>



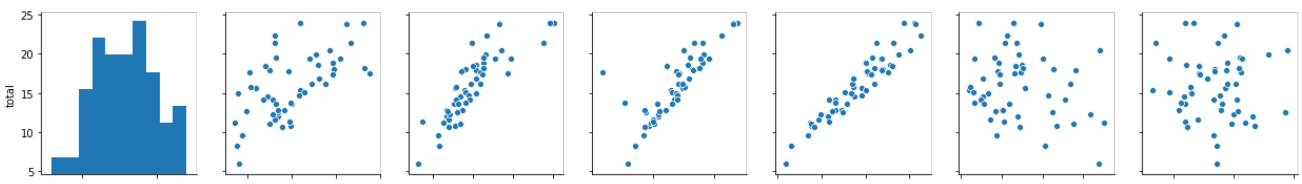
the pairplot function creates a matrix of axes and shows the relationship for each pair of columns in a DataFrame.

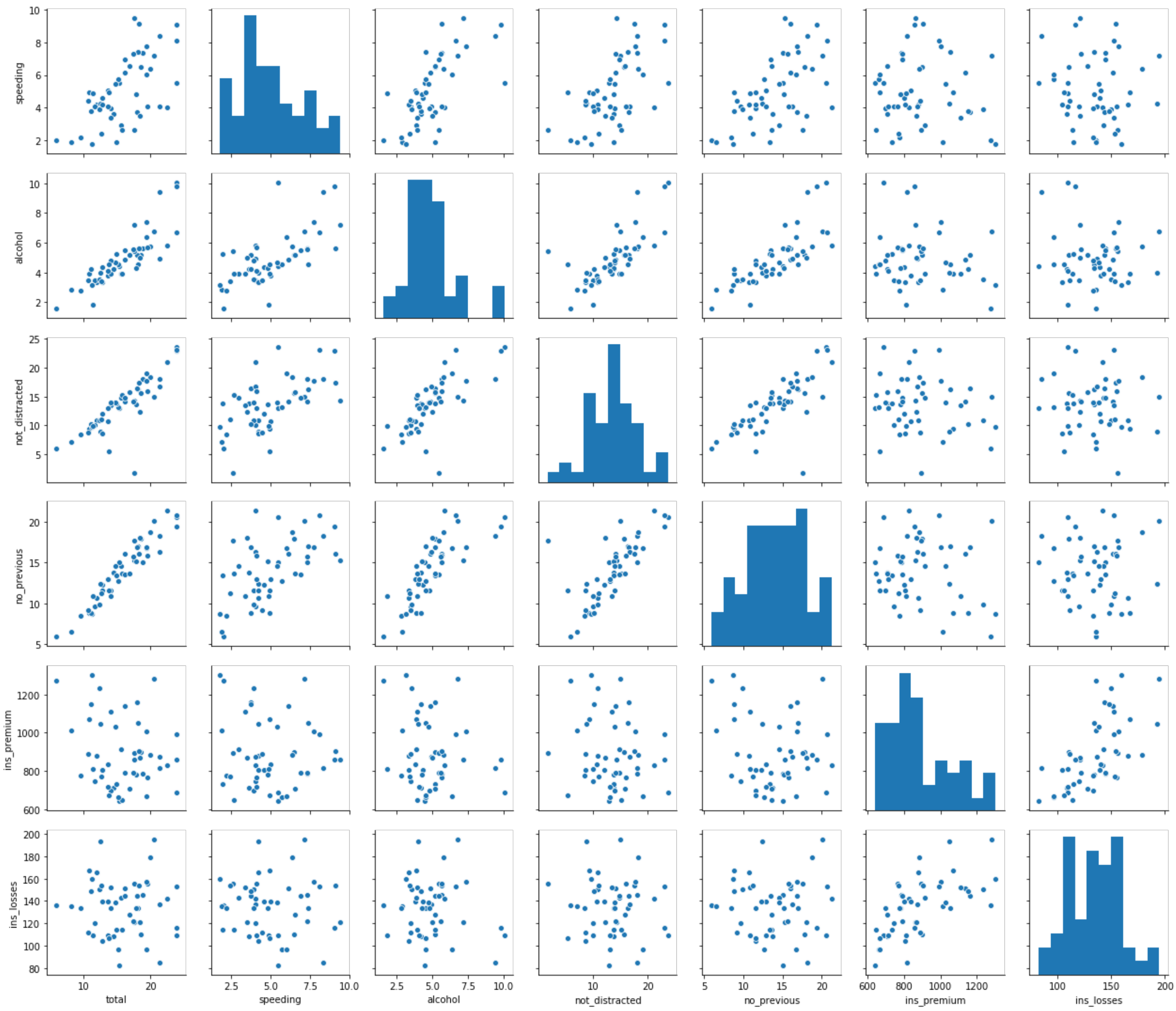
In [163]:

```
sns.pairplot(a)
```

Out[163]:

<seaborn.axisgrid.PairGrid at 0x149d5d1e548>





In [164]:

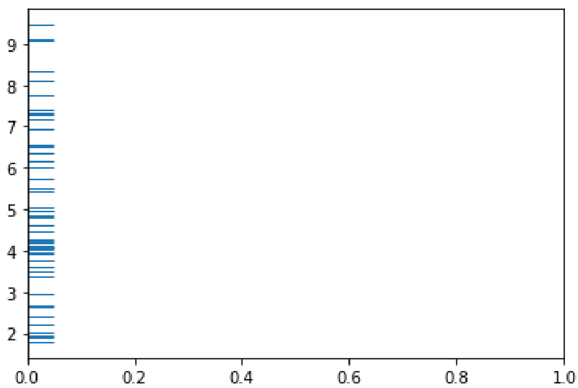
```
# rug plot
```

In [165]:

```
sns.rugplot(a['speeding'],axis='y')
```

Out[165]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d8232148>

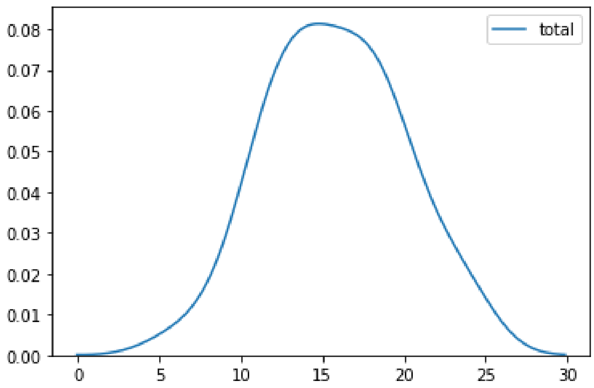


In [166]:

```
sns.kdeplot(a['total'])
```

Out[166]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d84ffac8>



In [167]:

```
#categorical plot:
```

In [168]:

```
import seaborn as sns
%matplotlib inline
a=sns.load_dataset("dots")
a.head()
```

Out[168]:

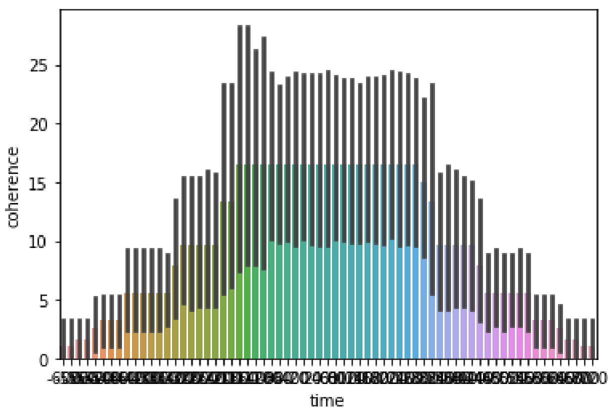
	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487

In [169]:

```
import numpy as np
sns.barplot(x='time',y='coherence',data=a)#the graph seems to be gradually inceasing and decreasing
```

Out[169]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d8525b48>

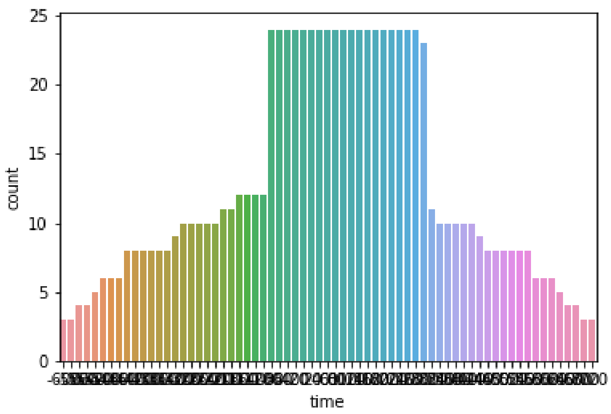


In [170]:

```
sns.countplot(x='time',data=a)#here the time increases gradually in the middle and fall down back
```

Out[170]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d8f3b9c8>

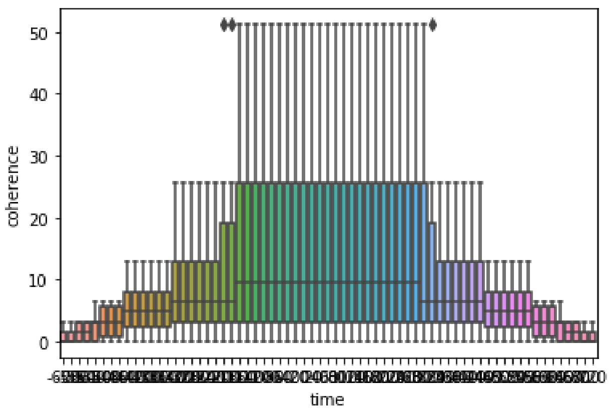


In [171]:

```
sns.boxplot(x='time',y='coherence',data=a)
```

Out[171]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d90f9e08>

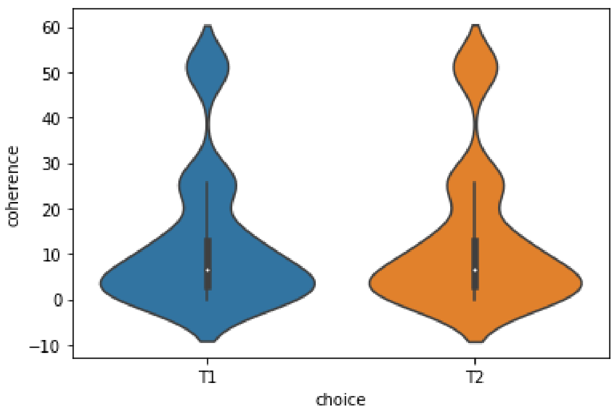


In [172]:

```
sns.violinplot(x='choice',y='coherence',data=a)
```

Out[172]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d96c6a08>

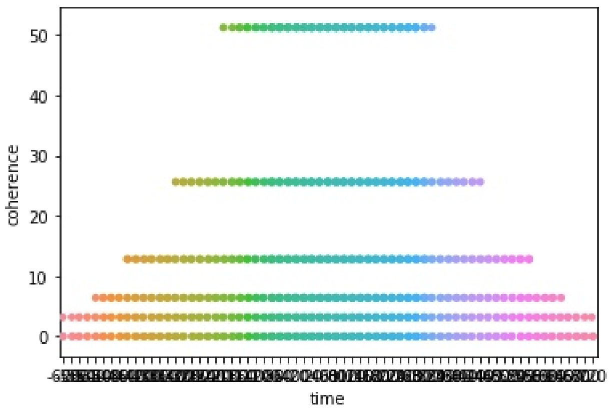


In [173]:

```
sns.stripplot(x='time',y='coherence',data=a,jitter=True)
```

Out[173]:

<matplotlib.axes._subplots.AxesSubplot at 0x149d97d94c8>

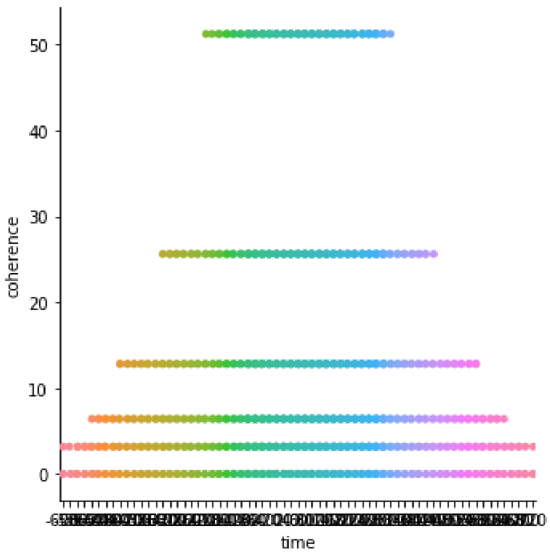


In [175]:

```
sns.catplot(x='time',y='coherence',data=a)
```

Out[175]:

<seaborn.axisgrid.FacetGrid at 0x149da92d2c8>



matrix plots

In [180]:

```
import seaborn as sns
%matplotlib inline
a=sns.load_dataset('car_crashes')
b=sns.load_dataset('dots')
```

In [181]:

```
a
```

Out[181]:

	total	speeding	alcohol	not_distracted	no_previous	ins_premium	ins_losses	abbrev
0	18.8	7.332	5.640	18.048	15.040	784.55	145.08	AL
1	18.1	7.421	4.525	16.290	17.014	1053.48	133.93	AK
2	18.6	6.510	5.208	15.624	17.856	899.47	110.35	AZ
3	22.4	4.032	5.824	21.056	21.280	827.34	142.39	AR
4	12.0	4.200	3.360	10.920	10.680	878.41	165.63	CA
5	13.6	5.032	3.808	10.744	12.920	835.50	139.91	CO
6	10.8	4.968	3.888	9.396	8.856	1068.73	167.02	CT
7	16.2	6.156	4.860	14.094	16.038	1137.87	151.48	DE
8	5.9	2.006	1.593	5.900	5.900	1273.89	136.05	DC
9	17.9	3.759	5.191	16.468	16.826	1160.13	144.18	FL
10	15.6	2.964	3.900	14.820	14.508	913.15	142.80	GA
11	17.5	9.450	7.175	14.350	15.225	861.18	120.92	HI
12	15.3	5.508	4.437	13.005	14.994	641.96	82.75	ID
13	12.8	4.608	4.352	12.032	12.288	803.11	139.15	IL
14	14.5	3.625	4.205	13.775	13.775	710.46	108.92	IN
15	15.7	2.669	3.925	15.229	13.659	649.06	114.47	IA
16	17.8	4.806	4.272	13.706	15.130	780.45	133.80	KS
17	21.4	4.066	4.922	16.692	16.264	872.51	137.13	KY
18	20.5	7.175	6.765	14.965	20.090	1281.55	194.78	LA
19	15.1	5.738	4.530	13.137	12.684	661.88	96.57	ME
20	12.5	4.250	4.000	8.875	12.375	1048.78	192.70	MD
21	8.2	1.886	2.870	7.134	6.560	1011.14	135.63	MA
22	14.1	3.384	3.948	13.395	10.857	1110.61	152.26	MI
23	9.6	2.208	2.784	8.448	8.448	777.18	133.35	MN
24	17.6	2.640	5.456	1.760	17.600	896.07	155.77	MS
25	16.1	6.923	5.474	14.812	13.524	790.32	144.45	MO
26	21.4	8.346	9.416	17.976	18.190	816.21	85.15	MT
27	14.9	1.937	5.215	13.857	13.410	732.28	114.82	NE
28	14.7	5.439	4.704	13.965	14.553	1029.87	138.71	NV
29	11.6	4.060	3.480	10.092	9.628	746.54	120.21	NH
30	11.2	1.792	3.136	9.632	8.736	1301.52	159.85	NJ
31	18.4	3.496	4.968	12.328	18.032	869.85	120.75	NM
32	12.3	3.936	3.567	10.824	9.840	1234.31	150.01	NY
33	16.8	6.552	5.208	15.792	13.608	708.24	127.82	NC
34	23.9	5.497	10.038	23.661	20.554	688.75	109.72	ND
35	14.1	3.948	4.794	13.959	11.562	697.73	133.52	OH
36	19.9	6.368	5.771	18.308	18.706	881.51	178.86	OK
37	12.8	4.224	3.328	8.576	11.520	804.71	104.61	OR
38	18.2	9.100	5.642	17.472	16.016	905.99	153.86	PA
39	11.1	3.774	4.218	10.212	8.769	1148.99	148.58	RI
40	23.9	9.082	9.799	22.944	19.359	858.97	116.29	SC
41	19.4	6.014	6.402	19.012	16.684	669.31	96.87	SD
42	19.5	4.095	5.655	15.990	15.795	767.91	155.57	TN
43	19.4	7.760	7.372	17.654	16.878	1004.75	156.83	TX
44	11.3	4.859	1.808	9.944	10.848	809.38	109.48	UT
45	13.6	4.080	4.080	13.056	12.920	716.20	109.61	VT
46	12.7	2.413	3.429	11.049	11.176	768.95	153.72	VA
47	10.6	4.452	3.498	8.692	9.116	890.03	111.62	WA
48	23.8	8.092	6.664	23.086	20.706	992.61	152.56	WV

	total	speeding	alcohol	not_distracted	no_previous	ins_premium	ins_losses	abbrev
49	13.8	4.968	4.554	5.382	11.592	670.31	106.62	WI
50	17.4	7.308	5.568	14.094	15.660	791.14	122.04	WY

In [182]:

```
b
```

Out[182]:

	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487
...	...	...	...	...	...
843	sacc	T2	300	3.2	33.281734
844	sacc	T2	300	6.4	27.583979
845	sacc	T2	300	12.8	28.511530
846	sacc	T2	300	25.6	27.009804
847	sacc	T2	300	51.2	30.959302

848 rows × 5 columns

In [183]:

```
x=a.corr()
x
```

Out[183]:

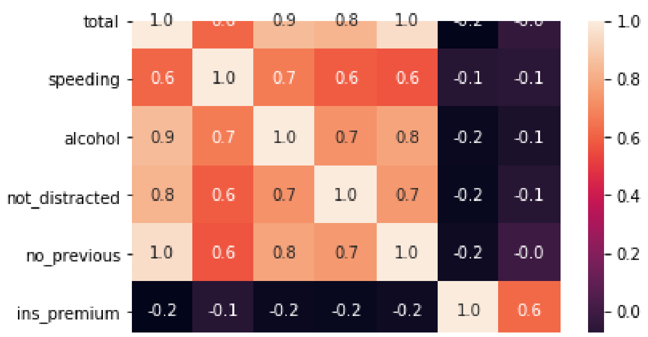
	total	speeding	alcohol	not_distracted	no_previous	ins_premium	ins_losses
total	1.000000	0.611548	0.852613	0.827560	0.956179	-0.199702	-0.036011
speeding	0.611548	1.000000	0.669719	0.588010	0.571976	-0.077675	-0.065928
alcohol	0.852613	0.669719	1.000000	0.732816	0.783520	-0.170612	-0.112547
not_distracted	0.827560	0.588010	0.732816	1.000000	0.747307	-0.174856	-0.075970
no_previous	0.956179	0.571976	0.783520	0.747307	1.000000	-0.156895	-0.006359
ins_premium	-0.199702	-0.077675	-0.170612	-0.174856	-0.156895	1.000000	0.623116
ins_losses	-0.036011	-0.065928	-0.112547	-0.075970	-0.006359	0.623116	1.000000

In [184]:

```
sns.heatmap(x,annot=True,fmt='.1f')
```

Out[184]:

<matplotlib.axes._subplots.AxesSubplot at 0x149dad8e2c8>



In [185]:

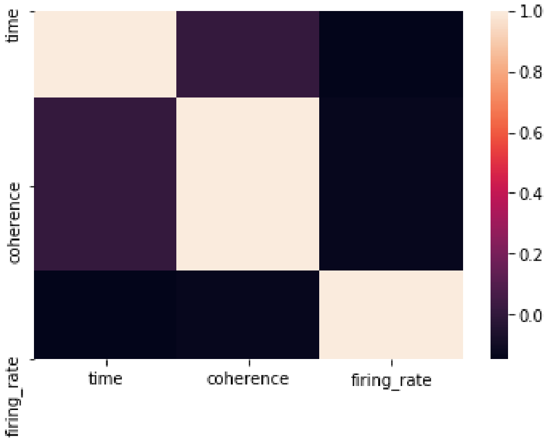
```
y=b.corr()
```

In [186]:

```
sns.heatmap(y)
```

Out[186]:

<matplotlib.axes._subplots.AxesSubplot at 0x149dae46888>

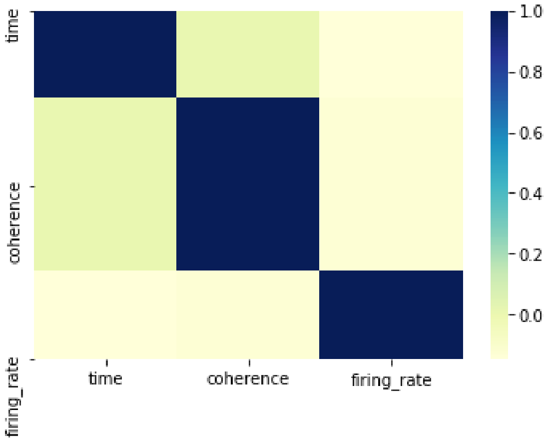


In [187]:

```
sns.heatmap(y,cmap='YlGnBu')
```

Out[187]:

<matplotlib.axes._subplots.AxesSubplot at 0x149dae2b248>

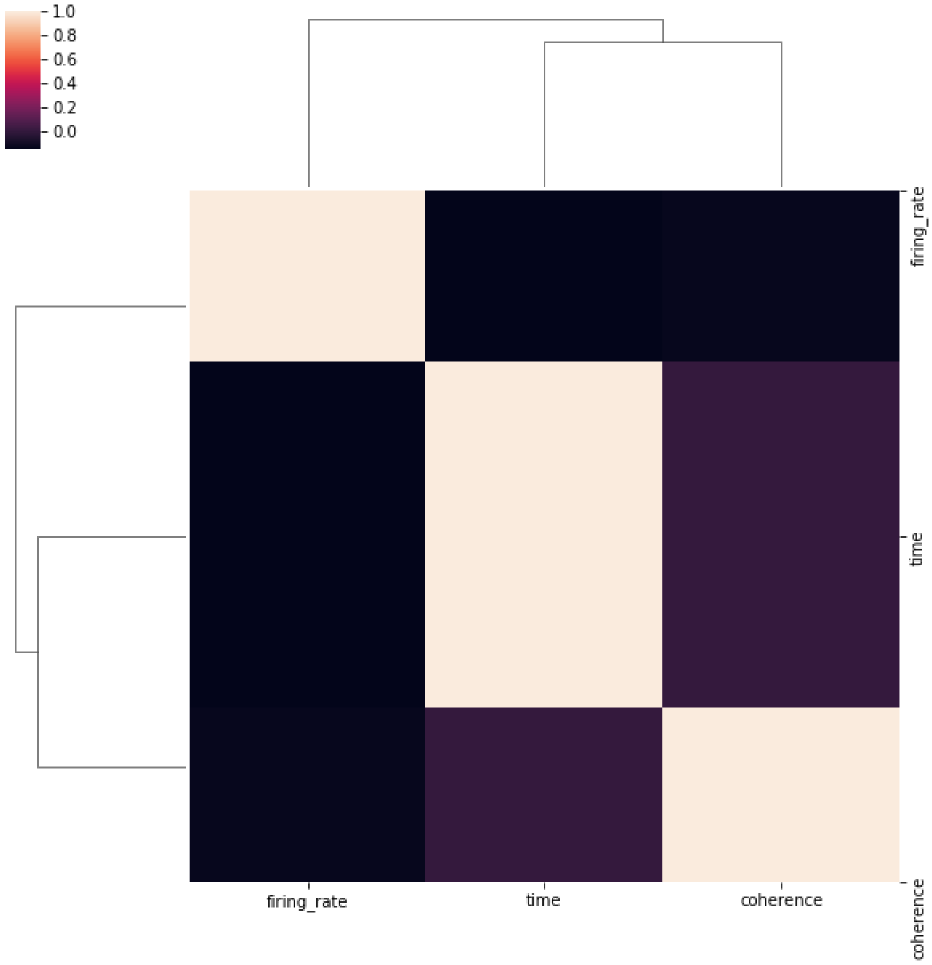


In [189]:

```
sns.clustermap(y)
```

Out[189]:

<seaborn.matrix.ClusterGrid at 0x149db0df9c8>


1.0
0.8
0.6
0.4
0.2
0.0
firing_rate
time
coherence
firing_rate
time
coherence

In [ ]: