In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('Poker.csv')
df#suits are s, c = value of poker hand
```

Out[2]:

	s1	c1	s2	c2	s3	c3	s4	c4	s5	c5
0	4	10	4	4	3	2	4	9	3	5
1	1	4	4	9	2	8	2	2	2	5
2	2	8	3	5	2	6	1	5	4	3
3	1	4	1	8	3	9	2	1	3	10
4	3	7	4	8	2	5	3	13	4	6
5	3	13	1	9	1	10	3	7	2	1
6	4	6	4	9	4	5	2	8	3	8
7	2	5	4	10	2	4	4	3	4	13
8	2	4	1	7	4	11	4	13	3	2
9	4	5	3	11	1	6	4	7	1	10
10	2	8	4	8	3	3	2	7	4	13
11	1	3	2	5	3	2	1	7	4	5
12	3	13	4	8	3	5	4	10	3	2
13	3	10	3	9	4	13	3	1	4	2
14	4	3	3	6	1	13	1	8	2	10
15	4	1	4	13	4	4	4	6	3	1
16	4	10	4	4	4	3	3	9	2	5
17	1	13	4	10	1	2	2	6	4	7
18	3	10	3	6	2	4	2	12	1	11
19	2	7	2	12	1	5	2	9	4	8
20	3	4	3	13	2	9	2	2	4	10
21	1	7	2	11	4	8	2	5	2	13
22	3	11	3	9	2	2	1	13	1	1
23	1	4	1	7	1	6	1	1	3	11
24	2	13	1	4	4	4	3	6	1	1
25	1	11	3	13	4	1	2	7	1	2
26	4	12	4	10	1	6	3	11	4	2
27	3	2	1	1	3	1	1	2	1	5
28	3	4	1	3	2	6	4	2	4	12
29	2	13	2	8	2	7	2	2	4	8
...	...	...	...	...	...	...	...	...	...	...
199970	2	13	2	10	2	4	1	1	3	6

	s1	c1	s2	c2	s3	c3	s4	c4	s5	c5
199971	4	1	2	7	3	7	4	3	1	8
199972	4	2	4	13	4	6	2	12	3	6
199973	3	2	3	3	2	9	2	11	1	12
199974	3	12	4	2	4	12	3	4	2	13
199975	3	11	2	7	1	10	4	1	2	1
199976	4	12	1	8	2	6	3	13	1	1
199977	4	1	4	5	3	3	3	12	1	6
199978	1	6	4	5	2	8	2	12	4	1
199979	1	13	2	10	1	7	3	1	1	12
199980	3	1	4	13	2	5	2	1	1	12
199981	2	3	4	9	4	4	3	7	2	2
199982	3	3	2	6	4	6	3	13	2	5
199983	3	3	1	12	2	3	1	10	1	9
199984	3	4	2	8	1	10	1	7	4	1
199985	1	13	2	8	4	7	3	2	2	11
199986	4	13	3	10	4	2	3	11	2	4
199987	1	1	2	5	3	1	4	13	4	3
199988	2	6	4	10	4	5	3	6	3	4
199989	3	7	1	7	1	1	2	9	4	7
199990	1	4	3	2	4	13	2	6	4	7
199991	4	1	2	9	2	1	2	3	4	13
199992	4	11	2	6	3	7	2	12	1	7
199993	1	8	3	4	1	3	4	2	3	7
199994	2	3	3	8	3	5	2	7	2	1
199995	2	7	2	12	1	11	4	4	3	1
199996	2	3	3	3	4	9	4	7	3	5
199997	2	2	3	1	4	11	2	10	3	6
199998	2	8	4	6	1	3	1	11	4	7
199999	4	8	1	6	3	2	1	4	3	12

200000 rows × 10 columns

In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_digits
```

In [4]:
```
from sklearn.model_selection import train_test_split

X = df

#y = df['c5']
X.drop(['c5'], axis = 1, inplace = True)
print(X.shape)
#print(y.shape)
```

```
(200000, 9)
```

In [ ]:

In [ ]:

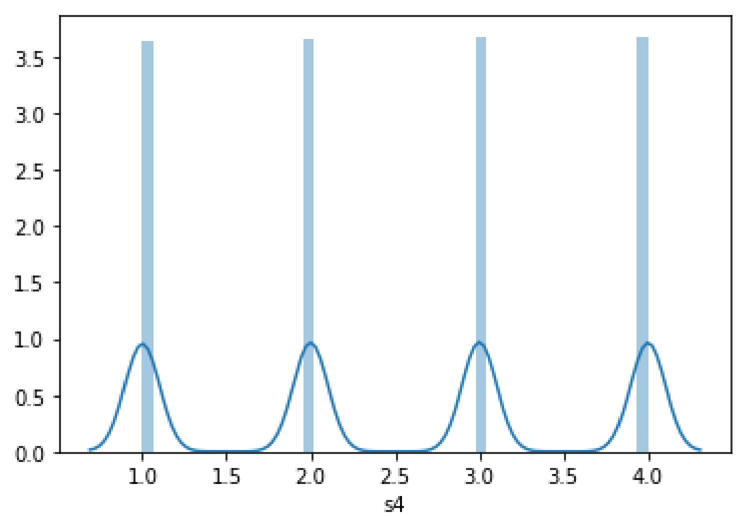
In [5]:
```
X.columns
```

Out[5]:
```
Index(['s1', 'c1', 's2', 'c2', 's3', 'c3', 's4', 'c4', 's5'], dtype='obje
ct')
```

Now we check for any anomalies in the data

In [6]:
```
sns.distplot(df['s4'])# distribution of the suits
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11aff16d0>
```

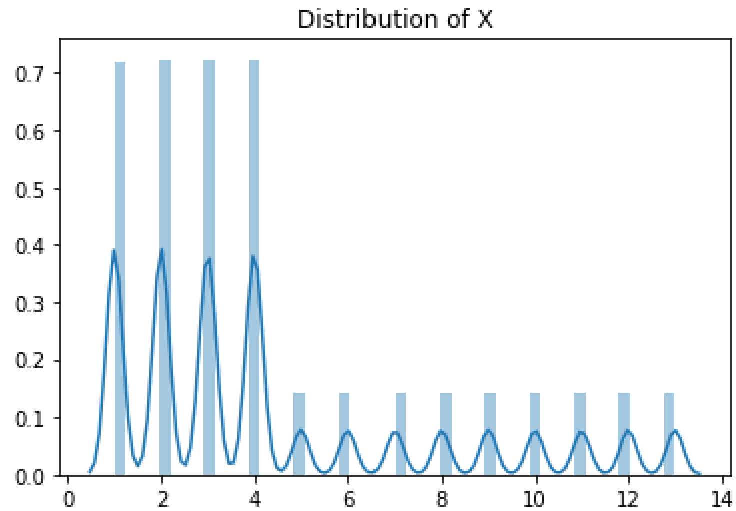


In [ ]:

Looks standardized so far

In [7]:
```
sns.distplot(X)#distribution of all columns of X
plt.title('Distribution of X')
```

Out[7]: `Text(0.5, 1.0, 'Distribution of X')`



Now we start making methods to classify hands

we aslo need to classify the hands in a new column (if we can)

In [8]:
```
import numpy as np
X['classification'] = np.arange(0,200000)*0
X.head()# ok, so now we have an ordered list
```

Out[8]:

	s1	c1	s2	c2	s3	c3	s4	c4	s5	classification
0	4	10	4	4	3	2	4	9	3	0
1	1	4	4	9	2	8	2	2	2	0
2	2	8	3	5	2	6	1	5	4	0
3	1	4	1	8	3	9	2	1	3	0
4	3	7	4	8	2	5	3	13	4	0

In [9]:

```
l = []
def flush():
    count = 0
    for i in range(len(X)//20):

        if (X['s1'][i] == X['s2'][i] == X['s3'][i] == X['s4'][i] ):# flush
            count += 1
            X['classification'][i:i+1].replace(0 ,1, inplace = True)
            #X['classification'][i] == 1

    return count
print(flush())
#if (abs(c1+1) == abs(c2 + 2) == abs(c3 + 3) == abs(c4 + 4)):#straights
    #None
flush()
l.append(flush())#added a function to a list
```

```
103
```

In [10]:

```
def straight():
    count = 0
    for i in range(len(df)//20):

        if (abs(X['c1'][i]+1) == abs(X['c2'][i] + 2) == abs(X['c3'][i] + 3)
            count += 1
            X['classification'][i:i+1].replace(0 ,2, inplace = True)
            #X['classification'][i] == 2
    print(count)
    return count

#5- 4 card straights in the first 200,000/20 = 10,000
straight()
l.append(straight())
```

```
5
5
```

In [11]:
```
def p1():
    count = 0
    for i in range(len(df)//20):

        if (X['c1'][i] == X['c2'][i]):#first set of pairs
            count += 1
            X['classification'][i:i+1].replace(0 ,3, inplace = True)
            #X['classification'][i] == 3
    print(count)
    return count

#5- 4 card straights in the first 200,000/20 = 10,000
p1()
l.append(p1())
```

```
606
606
```

In [12]:
```
l #checkpoint for list
```

Out[12]:
```
[103, 5, 606]
```

In [13]:
```
def p2():
    count = 0
    for i in range(len(df)//20):

        if (X['c1'][i] == X['c3'][i]):#second set of pairs
            count += 1
            X['classification'][i:i+1].replace(0 ,4, inplace = True)
            #X['classification'][i] == 4
    print(count)
    return count

p2()
l.append(p2())
```

```
571
571
```

In [14]:
```
def p3():
    count = 0
    for i in range(len(df)//20):

         if (X['c1'][i] == X['c4'][i]):#third set of pairs
            count += 1
            X['classification'][i:i+1].replace(0 ,5, inplace = True)
            #X['classification'][i] == 5
    print(count)
    return count


l.append(p3())


p3()
```

```
580
580
```

Out[14]:
```
580
```

In [15]:
```
def p4():
    count = 0
    for i in range(len(df)//20):

         if (X['c1'][i] == X['c4'][i]):#fourth set of pairs
            count += 1
            X['classification'][i:i+1].replace(0 ,6, inplace = True)
            #X['classification'][i] == 6
    print(count)
    return count


l.append(p4())
p4()
```

```
580
580
```

Out[15]:
```
580
```

In [16]:
```
def p5():
    count = 0
    for i in range(len(df)//20):

        if (X['c2'][i] == X['c4'][i]):#fifth set of pairs
           count += 1
           X['classification'][i:i+1].replace(0 ,7, inplace = True)
           #X['classification'][i] == 7
    print(count)
    return count


l.append(p5())


p5()
```

```
574
574
```

Out[16]: 574

In [17]:
```
def p6():
    count = 0
    for i in range(len(df)//20):

        if (X['c3'][i] == X['c4'][i]):#sixth set of pairs
           count += 1
           X['classification'][i:i+1].replace(0 ,8, inplace = True)
           #X['classification'][i] == 8
    print(count)
    return count


l.append(p6())

p6()
```

```
571
571
```

Out[17]: 571

In [18]:
```
sum(l)
```

Out[18]: 3590

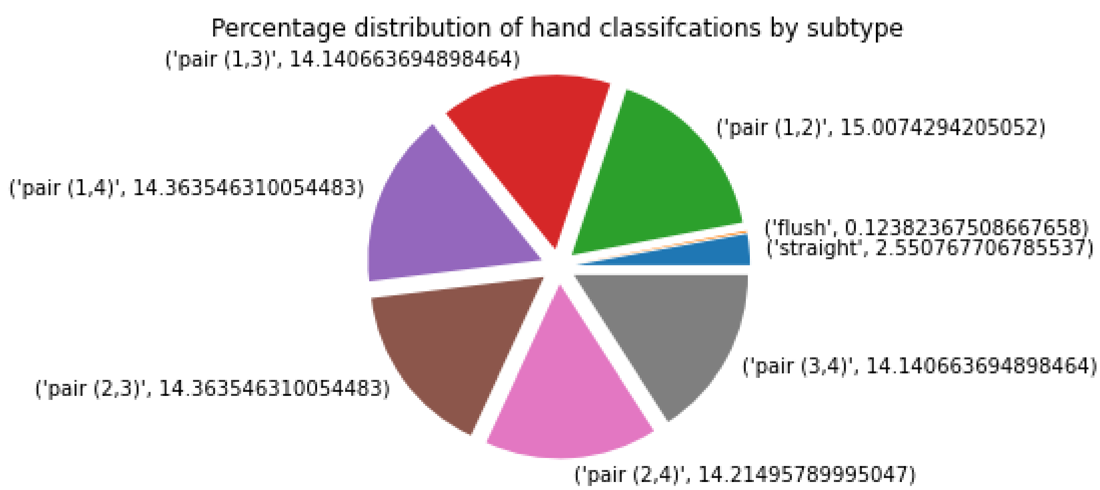
1748 special hands (not high card) so far (omitting three of a kind and full house)

In [19]:
```
len(l)
```
Out[19]:
```
8
```

In [20]:
```
perc = []
for elem in l:
    perc.append((elem*100)/(4038))
print(perc)#hands in percentage form
```
```
[2.550767706785537, 0.12382367508667658, 15.0074294205052, 14.14066369489
8464, 14.363546310054483, 14.363546310054483, 14.21495789995047, 14.14066
3694898464]
```

In [21]:
```
#plt.plot(l)
```

In [22]:
```
labs = [('straight',perc[0]), ('flush',perc[1]), ('pair (1,2)',perc[2]), ('
plt.pie(perc, labels = labs, explode = [.1,.1,.1,.1,.1,.1,.1,.1])
plt.title('Percentage distribution of hand classifcations by subtype')
```
Out[22]:
```
Text(0.5, 1.0, 'Percentage distribution of hand classifcations by subtyp
e')
```



Tuples as labels with percentages and classifcations

In [23]:
```
# as you can see, the pairs are roughly equally distributed, with flushes a
```

All 4 card hands in tuple zip form

In [24]:
```
dy = df[:10000]
listCombined = zip(dy['c1'],dy['c2'],dy['c3'],dy['c4'])
```

In [25]:
```
print(tuple(listCombined))
```

```
1, 5), (3, 5, 12, 4), (13, 11, 10, 9), (11, 11, 4, 2), (3, 1, 3, 2), (1,
6, 6, 13), (3, 7, 12, 9), (3, 4, 10, 4), (1, 2, 9, 5), (3, 7, 8, 6), (4,
3, 8, 8), (1, 11, 6, 7), (9, 6, 3, 5), (13, 9, 7, 8), (12, 5, 1, 13), (3,
12, 13, 10), (6, 11, 10, 3), (9, 7, 1, 1), (1, 4, 12, 11), (5, 12, 3, 1
2), (5, 1, 12, 10), (4, 12, 2, 4), (13, 4, 7, 5), (10, 9, 1, 2), (6, 12,
11, 10), (4, 6, 11, 1), (2, 10, 10, 7), (11, 10, 8, 4), (11, 12, 1, 6),
(13, 2, 6, 11), (5, 3, 6, 5), (10, 10, 12, 6), (3, 8, 6, 7), (9, 4, 11,
1), (1, 8, 1, 6), (3, 5, 7, 9), (11, 10, 13, 11), (1, 10, 4, 9), (11, 2,
11, 10), (3, 3, 2, 8), (1, 9, 13, 9), (2, 12, 9, 6), (10, 1, 11, 11), (1
2, 8, 9, 7), (7, 9, 2, 1), (7, 7, 13, 12), (10, 12, 11, 6), (2, 11, 4,
3), (9, 1, 8, 12), (6, 12, 7, 5), (3, 2, 4, 7), (8, 2, 1, 11), (12, 3, 1
3, 9), (9, 10, 13, 5), (6, 12, 1, 3), (8, 13, 9, 9), (6, 6, 8, 7), (11, 1
1, 2, 13), (5, 12, 2, 3), (5, 12, 6, 9), (12, 5, 10, 3), (10, 6, 8, 12),
(3, 5, 8, 9), (10, 3, 10, 3), (7, 11, 10, 8), (13, 1, 3, 10), (8, 13, 1,
4), (4, 3, 2, 12), (8, 12, 9, 10), (1, 8, 3, 2), (11, 2, 9, 10), (4, 6,
6, 12), (7, 3, 10, 2), (11, 10, 9, 12), (12, 9, 7, 9), (3, 11, 4, 5), (7,
12, 3, 2), (13, 13, 3, 5), (11, 6, 3, 8), (7, 9, 12, 8), (12, 1, 8, 3),
(13, 9, 10, 1), (12, 1, 3, 2), (2, 11, 6, 5), (5, 9, 12, 1), (10, 5, 6, 1
0), (13, 6, 12, 5), (3, 12, 4, 11), (10, 3, 3, 5), (3, 12, 13, 2), (11,
```

In [26]:
```
y1 = ([0,1,2,3,4,5,6,7,8])# all the different hand classifcations and pair
```

In [27]:
```
#len(tuple(listCombined)) - 0??
```

can we figure out the fourth card from the hand classifcation and the first 3 cards?

In [47]:
```
#X.where(X['classifcation']==0 , X['classifcation'] == 1, axis = 0)
flush()
straight()
p1()
p2()
p3()
p4()
p5()
p6()
X.head(n = 20)

#X_train, X_test, y1_train, y1_test = train_test_split(X, y1, test_size = .
```

```
5
606
571
580
580
574
571
```

Out[47]:

	s1	c1	s2	c2	s3	c3	s4	c4	s5	classification
0	4	10	4	4	3	2	4	9	3	0
1	1	4	4	9	2	8	2	2	2	0
2	2	8	3	5	2	6	1	5	4	7
3	1	4	1	8	3	9	2	1	3	0
4	3	7	4	8	2	5	3	13	4	0
5	3	13	1	9	1	10	3	7	2	0
6	4	6	4	9	4	5	2	8	3	0
7	2	5	4	10	2	4	4	3	4	0
8	2	4	1	7	4	11	4	13	3	0
9	4	5	3	11	1	6	4	7	1	0
10	2	8	4	8	3	3	2	7	4	3
11	1	3	2	5	3	2	1	7	4	0
12	3	13	4	8	3	5	4	10	3	0
13	3	10	3	9	4	13	3	1	4	0
14	4	3	3	6	1	13	1	8	2	0
15	4	1	4	13	4	4	4	6	3	1
16	4	10	4	4	4	3	3	9	2	0
17	1	13	4	10	1	2	2	6	4	0
18	3	10	3	6	2	4	2	12	1	0
19	2	7	2	12	1	5	2	9	4	0

WE DID IT - CLASSIFIED HANDS BASED ON RANK

In [30]:
```
y2 = X['classification']

X_train, X_test, y2_train, y2_test = train_test_split(X, y2, test_size = .2
```

Now we move to KNN machine learning

In [32]:
```
print(X_train.shape)
print(X_test.shape)
print(y2_train.shape)
print(y2_test.shape)
```

```
(150000, 10)
(50000, 10)
(150000,)
(50000,)
```

In [33]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
```

In [35]:
```
print(knn.fit(X_train, y2_train))
```

```
KNeighborsClassifier()
```

In [36]:
```
yPredict = knn.predict(X_test)
yPredict
```

Out[36]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [37]:
```
expectedYvalues = y2_test
expectedYvalues
```

Out[37]:
```
41850     0
179391    0
140190    0
167550    0
85069     0
87539     0
114435    0
106552    0
128414    0
121434    0
130392    0
6958      8
121454    0
117004    0
4423      3
128704    0
143072    0
61397     0
126124    0
11898     0
91523     0
196610    0
44082     0
174988    0
147196    0
31058     0
139701    0
97685     0
180184    0
49597     0
         ..
60674     0
123914    0
85242     0
77868     0
158413    0
141293    0
10142     0
174042    0
19050     0
191435    0
5702      0
47027     0
14424     0
182637    0
64832     0
5905      3
71022     0
71161     0
106160    0
137238    0
79669     0
35844     0
198490    0
```

```
137251    0
161552    0
70926     0
79406     0
184217    0
186833    0
147158    0
Name: classification, Length: 50000, dtype: int64
```

In [40]:
```
#print(knn.score(X_test, y2_test))
```

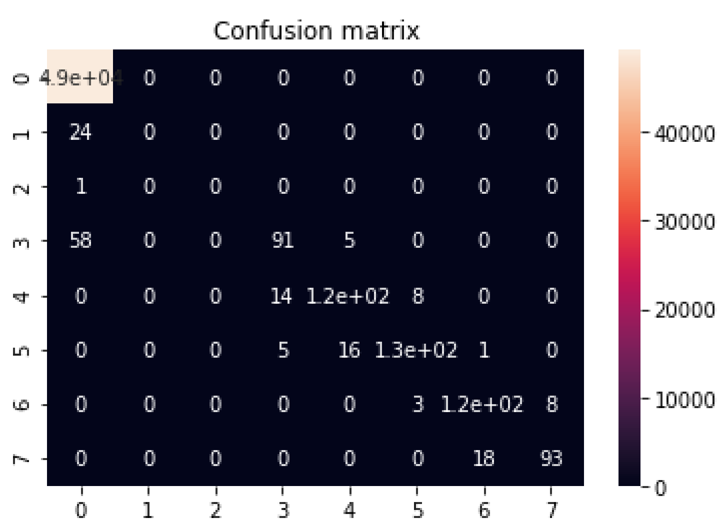
now we will analyze the confusion matrix and the classification report

In [41]:
```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(expectedYvalues, yPredict))
```

```
[[49280     0     0     0     0     0     0     0]
 [   24     0     0     0     0     0     0     0]
 [    1     0     0     0     0     0     0     0]
 [   58     0     0    91     5     0     0     0]
 [    0     0     0    14   122     8     0     0]
 [    0     0     0     5    16   131     1     0]
 [    0     0     0     0     0     3   122     8]
 [    0     0     0     0     0     0    18    93]]
```

In [45]:
```
sns.heatmap(confusion_matrix(expectedYvalues, yPredict), annot = True)
plt.title('Confusion matrix')
```

Out[45]: Text(0.5, 1.0, 'Confusion matrix')



In [56]:
```
from sklearn.metrics import classification_report
print(classification_report(expectedYvalues, yPredict))
```

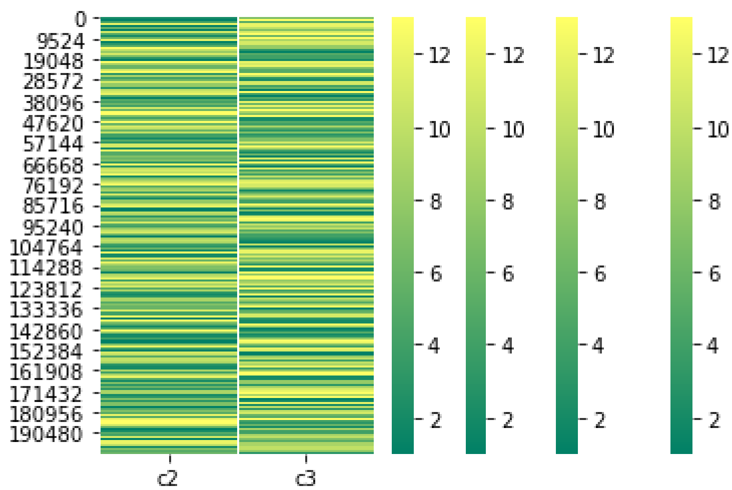
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49280
           1       0.00      0.00      0.00        24
           2       0.00      0.00      0.00         1
           3       0.83      0.59      0.69       154
           4       0.85      0.85      0.85       144
           5       0.92      0.86      0.89       153
           7       0.87      0.92      0.89       133
           8       0.92      0.84      0.88       111

    accuracy                           1.00     50000
   macro avg       0.67      0.63      0.65     50000
weighted avg       1.00      1.00      1.00     50000
```

So we are perfectly accurate at predicting high cards, pretty good at pairs, and terrible at the more rare hands - all of which makes sense

__ _ __ ____ __ __

Now we move to a logistic regression

In [61]:
```
import random
for i in range (len(X['c1'])//50000):
    r = random.randint(0,4)
    row = random.randint(0,10000)
    if r == 1:
        sns.heatmap(X[['c1', 'c2']], cmap = 'winter')
    elif r == 2:
        sns.heatmap(X[['c2', 'c3']], cmap = 'summer')
    elif r == 3:
        sns.heatmap(X[['c3', 'c4']], cmap = 'spring')
    else:
        sns.heatmap(X[['c1', 'c4']], cmap = 'autumn')
#print(digits['images'])
```



I have created art

In [62]:
```
from sklearn.model_selection import cross_val_score
```

In [63]:
```
from sklearn.model_selection import KFold
```

In [64]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=11)# shuffle is impor
```

In [78]:
```
X = X[:1000]

scores = cross_val_score(estimator = knn, X = X, y = X['classification'], c
```

In [79]:
```
scores
```

Out[79]:
```
array([0.87, 0.91, 0.92, 0.89, 0.89, 0.91, 0.86, 0.89, 0.82, 0.89])
```

very solid scores here - much better than KNN

moving on to the logistic regression analysis

In [80]:
```
cation']
t, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=690)
```

In [81]:
```
from sklearn.linear_model import LogisticRegression
```

In [82]:
```
print(X_train.shape)
print(y_train.shape)
# smaller subset of the full dataframe as an experiment
```

```
(500, 10)
(500,)
```

In [83]:
```
lr = LogisticRegression()
```

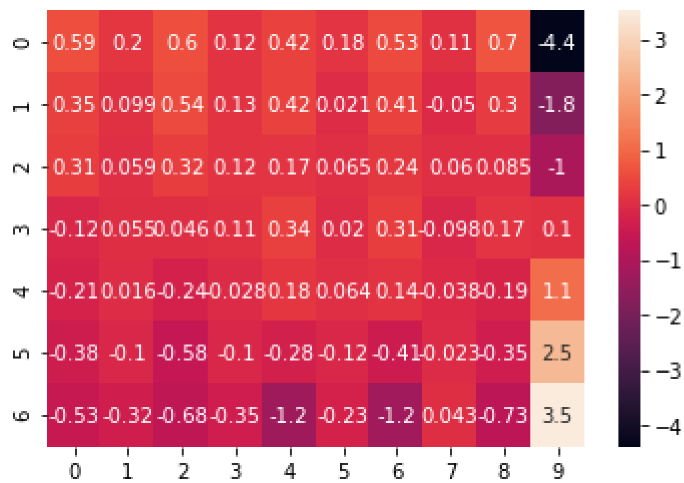
In [84]:
```
result = lr.fit(X_train,y_train)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  n_iter_i = _check_optimize_result(
```

In [89]:
```
print(result.coef_)
sns.heatmap(result.coef_, annot = True)
```

```
[[ 0.58640039  0.19751058  0.59705734  0.12060936  0.41798579  0.17545765
   0.53127812  0.1058985   0.70422309 -4.39193108]
 [ 0.34786895  0.09907909  0.53517587  0.12549271  0.41544105  0.0207972
   0.41496811 -0.04969794  0.29680738 -1.7524122 ]
 [ 0.30649637  0.05899169  0.32046457  0.12143929  0.17002569  0.06532975
   0.2364601   0.06007509  0.08465585 -1.0315097 ]
 [-0.12061756  0.05530928  0.04628421  0.11214341  0.33977423  0.02031089
   0.30545239 -0.09807297  0.17476984  0.10297707]
 [-0.20541451  0.01595248 -0.23808964 -0.02819568  0.18036997  0.06365031
   0.13836364 -0.03793757 -0.18529159  1.07696713]
 [-0.38465644 -0.10486068 -0.58352832 -0.10447678 -0.27706972 -0.12030076
  -0.41461454 -0.02337189 -0.34685825  2.47918455]
 [-0.53007718 -0.32198244 -0.67736403 -0.3470123  -1.24652701 -0.22524504
  -1.21190782  0.04310678 -0.72830634  3.51672423]]
```

Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x14ffb4a90>



In [86]:
```
result.intercept_
```

Out[86]:
```
array([ 1.13017594,  0.81203529,  1.95714464,  0.52831089, -0.91333352,
       -2.16568244, -1.3486508 ])
```

In [90]:
```
yPredict = lr.predict(X_test)

yPredict[:50]
```

Out[90]:
```
array([0, 3, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 7, 3, 8, 0, 3, 0, 0, 8, 0,
       0, 7, 5, 8, 0, 0, 0, 5, 0, 0, 7, 4, 0, 0, 7, 0, 0, 4, 3, 3, 0, 5,
       0, 0, 3, 0, 0, 3])
```

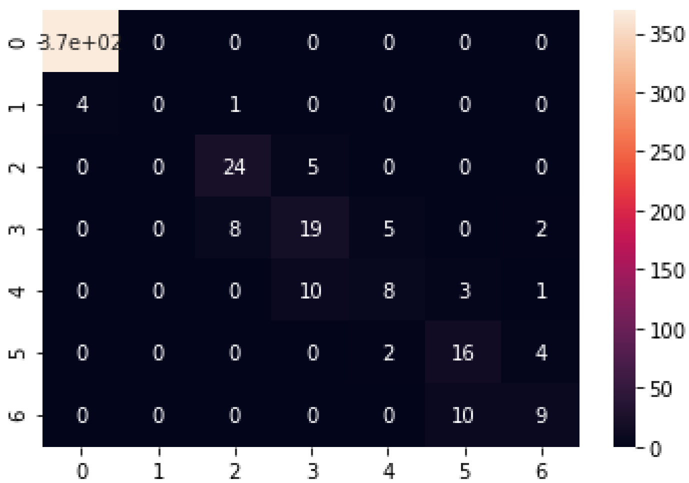
seems reasonable, about equal distributions of pairs (3-8); no 1's or 2's (rare)

In [93]:
```
# already have confusion matrix imported
confusion_matrix(y_test, yPredict)
```

Out[93]:
```
array([[369,   0,   0,   0,   0,   0,   0],
       [  4,   0,   1,   0,   0,   0,   0],
       [  0,   0,  24,   5,   0,   0,   0],
       [  0,   0,   8,  19,   5,   0,   2],
       [  0,   0,   0,  10,   8,   3,   1],
       [  0,   0,   0,   0,   2,  16,   4],
       [  0,   0,   0,   0,   0,  10,   9]])
```

In [95]:
```
sns.heatmap(confusion_matrix(y_test, yPredict), annot = True)
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x122213fd0>



again, high cards are well predicted, pairs less so

In [97]:
```
X = X[:200]# now trying with an even smaller subset
```

In [110]:
```
y = X['classification'][:200]
```

In [116]:
```
print(X_train.shape)
print(y_train.shape)
X_train , X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, r
```

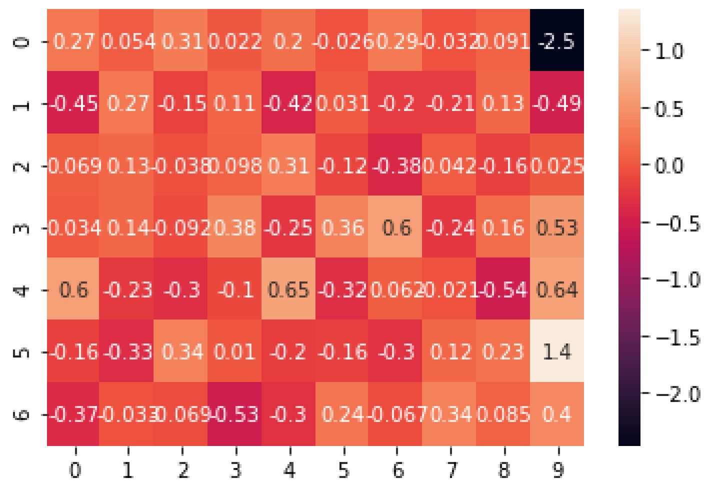
```
(100, 10)
(100,)
```

In [117]:
```
nresult = lr.fit(X_train,y_train)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logisti
c.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  n_iter_i = _check_optimize_result(
```

In [118]:
```
nresult.coef_
sns.heatmap(nresult.coef_, annot = True)
```

Out[118]: <matplotlib.axes._subplots.AxesSubplot at 0x1161dca30>



a bit strange that the model was better on a smaller sample size...

One final note to wrap this up:

in poker, there are hands which I omitted which include: '2-pair', 'three-of-a-kind', 'full house', and 'straight flush'

In [ ]:
```
#
```