exploration

May 16, 2016

In [1]:
```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Make the graphs a bit prettier, and bigger
pd.set_option('display.mpl_style', 'default')
plt.rcParams['figure.figsize'] = (15, 5)
plt.rcParams['font.family'] = 'sans-serif'

# This is necessary to show lots of columns in pandas 0.12.
# Not necessary in pandas 0.13.
pd.set_option('display.width', 5000)
pd.set_option('display.max_columns', 60)
```

In [2]:
```
df=pd.read_csv('sample2v.csv', header=None)
```

In [3]:
```
df
```

Out[3]:

	0	1	2	3	4	5	
0	964	C2374$@DOM1	C2374$@DOM1	C2375	C586	?	
1	2011	C1652$@DOM1	C1652$@DOM1	C1652	C612	Kerberos	Networ
2	3081	C538$@DOM1	C538$@DOM1	C539	C523	Kerberos	Networ
3	4104	C1008$@DOM1	C1008$@DOM1	C1008	C625	Kerberos	Networ
4	5118	U292@DOM1	U292@DOM1	C1737	C1737	?	
5	6134	C287$@DOM1	C287$@DOM1	C529	C529	?	Networ
6	7198	C2491$@DOM1	C2491$@DOM1	C457	C457	?	Networ
7	8145	C1367$@DOM1	C1367$@DOM1	C612	C612	?	Networ
8	9170	C1656$@DOM1	C1656$@DOM1	C1656	C1656	?	
9	10208	U22@DOM1	U22@DOM1	C506	C586	Kerberos	Networ
10	11221	C3816$@DOM1	C3816$@DOM1	C528	C528	?	Networ
11	12243	C3071$@DOM1	C3071$@DOM1	C3071	C1065	?	
12	13264	C3116$@DOM1	C3116$@DOM1	C3116	C625	?	
13	14324	C599$@DOM1	C599$@DOM1	C1619	C599	?	
14	15298	U22@DOM1	U22@DOM1	C477	C625	Kerberos	Networ
15	16303	C2070$@DOM1	C2070$@DOM1	C2071	C457	?	
16	17342	C1487$@DOM1	C1487$@DOM1	C1487	C586	Kerberos	Networ
17	18372	U9@DOM1	U9@DOM1	C223	C223	?	Networ
18	19370	ANONYMOUS LOGON@C586	ANONYMOUS LOGON@C586	C746	C586	NTLM	Networ
19	20376	C3319$@DOM1	C3319$@DOM1	C3319	C529	?	
20	21318	C6093$@DOM1	C6093$@DOM1	C6093	C585	?	
21	22165	U18@DOM1	U18@DOM1	C2086	C528	?	
22	22925	C2873$@DOM1	C2873$@DOM1	C457	C457	?	Networ

23	23635	C6231$@DOM1	C6231$@DOM1	C6231	C706	?		
24	24237	C2092$@DOM1	C2092$@DOM1	C2093	C467	Kerberos	Networ	
25	24818	C7210$@DOM1	C7210$@DOM1	C2106	C2106	?	Networ	
26	25309	C114$@DOM1	C114$@DOM1	C114	C586	Kerberos	Networ	
27	25761	C5232$@DOM1	C5232$@DOM1	C5232	C5232	?		
28	26191	U1900@DOM1	U1900@DOM1	C1444	C2327	?		
29	26607	C2475$@DOM1	C2475$@DOM1	C2475	C1065	Kerberos	Networ	
...	...	...	...	...	...	...	...	
10484	4997209	C16469$@DOM1	C16469$@DOM1	C612	C612	?	Networ	
10485	4997656	U10905@DOM3	U10905@DOM3	C3429	C3429	?	Networ	
10486	4998120	C1114$@DOM1	C1114$@DOM1	C1115	C1114	?		
10487	4998590	U6@DOM1	U6@DOM1	C423	C529	Kerberos	Networ	
10488	4999074	U8646@DOM1	U8646@DOM1	C17626	C1025	?		
10489	4999556	C480$@DOM1	C480$@DOM1	C625	C625	?	Networ	
10490	5000037	C3547$@DOM1	C3547$@DOM1	C3547	C3547	Kerberos	Networ	
10491	5000536	ANONYMOUS LOGON@C586	ANONYMOUS LOGON@C586	C586	C586	?	Networ	
10492	5001017	C15068$@DOM1	C15068$@DOM1	C15068	C15068	?	Networ	
10493	5001481	C11875$@DOM1	C11875$@DOM1	C457	C457	?	Networ	
10494	5001951	C1567$@DOM1	C1567$@DOM1	C625	C625	?	Networ	
10495	5002418	C599$@DOM1	C599$@DOM1	C1619	C523	Kerberos	Networ	
10496	5002895	C4552$@DOM1	C4552$@DOM1	C4552	C4552	?	Networ	
10497	5003383	C1710$@DOM1	C1710$@DOM1	C1710	C612	Kerberos	Networ	
10498	5003870	NETWORK SERVICE@C4501	NETWORK SERVICE@C4501	C4501	C4501	Negotiate	Servic	
10499	5004351	C1827$@DOM1	C1827$@DOM1	C1065	C1065	?	Networ	
10500	5004811	C21514$@DOM1	C21514$@DOM1	C586	C586	?	Networ	
10501	5005276	C14670$@DOM1	C14670$@DOM1	C14670	C801	Kerberos	Networ	
10502	5005758	C1665$@DOM1	C1665$@DOM1	C529	C529	?	Networ	
10503	5006238	ANONYMOUS LOGON@C586	ANONYMOUS LOGON@C586	C21566	C586	NTLM	Networ	
10504	5006720	C612$@DOM1	C612$@DOM1	C612	C612	Kerberos	Networ	
10505	5007201	C13862$@DOM1	C13862$@DOM1	C625	C625	?	Networ	
10506	5007705	U873@DOM1	U873@DOM1	C612	C612	?	Networ	
10507	5008181	U3145@DOM1	U3145@DOM1	C529	C529	?	Networ	
10508	5008643	C18896$@DOM1	C18896$@DOM1	C18896	C612	Kerberos	Networ	
10509	5009119	C429$@DOM1	C429$@DOM1	C429	TGT	?		
10510	5009595	U132@DOM1	U132@DOM1	C2040	C2327	?		
10511	5010073	C25126$@DOM1	C25126$@DOM1	C529	C529	?	Networ	
10512	5010560	C22501$@DOM1	C22501$@DOM1	C22502	C1065	Kerberos	Networ	
10513	5011052	C743$@DOM1	C743$@DOM1	C586	C586	?	Networ	

[10514 rows x 9 columns]

In [5]:
```
df[8].value_counts()
```

Out[5]:
```
Success    10400
Fail         114
Name: 8, dtype: int64
```

In [6]:
```
114./10400
```

Out[6]:
```
0.010961538461538462
```

1 Conclusion

In this dataset, Fails take roughly 1%. If it is a representative sample of the real data, then running machine learning on the whole set will just not make sense. Any classifier that just predict "Success" for every line will attain 99% accuracy.

This means that I need to collect data for "Fail" cases and randomly sample data for "Success" in roughly equal amounts and then look at machine learning (classifier) for such sets to see the true accuracy of the algorithm.

2 Next Steps

I want to examine this dataset to see if there are any obvious correlations and to understand the data I have in my columns.

3 authentication type

```
In [9]: df[5].unique()
```

```
Out[9]: array(['?', 'Kerberos', 'NTLM', 'MICROSOFT_AUTHENTICATION_PACKAGE_V1_0',
               'Negotiate'], dtype=object)
```

4 logon type

```
In [10]: df[6].unique()
```

```
Out[10]: array(['?', 'Network', 'Service', 'Unlock', 'Interactive', 'Batch',
                'NewCredentials', 'RemoteInteractive', 'NetworkCleartext',
                'CachedInteractive'], dtype=object)
```

5 authentication orientation

```
In [11]: df[7].unique()
```

```
Out[11]: array(['TGS', 'LogOn', 'TGT', 'LogOff', 'AuthMap', 'ScreenLock'], dtype=object)
```

```
In [14]: df.groupby([5,8]).count()
```

Out[14]:

5	8	0	1	2	3	4	6	7
?	Fail	77	77	77	77	77	77	77
	Success	5704	5704	5704	5704	5704	5704	5704
Kerberos	Fail	9	9	9	9	9	9	9
	Success	3708	3708	3708	3708	3708	3708	3708
MICROSOFT_AUTHENTICATION_PACKAGE_V1_0	Fail	8	8	8	8	8	8	8
	Success	1	1	1	1	1	1	1
NTLM	Fail	15	15	15	15	15	15	15
	Success	439	439	439	439	439	439	439
Negotiate	Fail	5	5	5	5	5	5	5
	Success	548	548	548	548	548	548	548

```
In [15]: df.groupby([6,8]).count()
```

Out[15]:

6	8	0	1	2	3	4	5	7
?	Fail	77	77	77	77	77	77	77
	Success	1361	1361	1361	1361	1361	1361	1361
Batch	Fail	1	1	1	1	1	1	1
	Success	12	12	12	12	12	12	12

CachedInteractive	Success	1	1	1	1	1	1	1
Interactive	Fail	3	3	3	3	3	3	3
	Success	19	19	19	19	19	19	19
Network	Fail	32	32	32	32	32	32	32
	Success	8463	8463	8463	8463	8463	8463	8463
NetworkCleartext	Success	1	1	1	1	1	1	1
NewCredentials	Success	14	14	14	14	14	14	14
RemoteInteractive	Success	5	5	5	5	5	5	5
Service	Success	492	492	492	492	492	492	492
Unlock	Fail	1	1	1	1	1	1	1
	Success	32	32	32	32	32	32	32

```
In [13]: df.groupby([7,8]).count()
```

Out[13]:

7	8	0	1	2	3	4	5	6
AuthMap	Success	103	103	103	103	103	103	103
LogOff	Success	4343	4343	4343	4343	4343	4343	4343
LogOn	Fail	37	37	37	37	37	37	37
	Success	4696	4696	4696	4696	4696	4696	4696
ScreenLock	Success	2	2	2	2	2	2	2
TGS	Fail	4	4	4	4	4	4	4
	Success	973	973	973	973	973	973	973
TGT	Fail	73	73	73	73	73	73	73
	Success	283	283	283	283	283	283	283

```
In [16]: df.groupby([6,7]).count()
```

Out[16]:

6	7	0	1	2	3	4	5	8
?	AuthMap	103	103	103	103	103	103	103
	ScreenLock	2	2	2	2	2	2	2
	TGS	977	977	977	977	977	977	977
	TGT	356	356	356	356	356	356	356
Batch	LogOff	7	7	7	7	7	7	7
	LogOn	6	6	6	6	6	6	6
CachedInteractive	LogOn	1	1	1	1	1	1	1
Interactive	LogOff	8	8	8	8	8	8	8
	LogOn	14	14	14	14	14	14	14
Network	LogOff	4310	4310	4310	4310	4310	4310	4310
	LogOn	4185	4185	4185	4185	4185	4185	4185
NetworkCleartext	LogOn	1	1	1	1	1	1	1
NewCredentials	LogOff	2	2	2	2	2	2	2
	LogOn	12	12	12	12	12	12	12
RemoteInteractive	LogOff	2	2	2	2	2	2	2
	LogOn	3	3	3	3	3	3	3
Service	LogOff	5	5	5	5	5	5	5
	LogOn	487	487	487	487	487	487	487
Unlock	LogOff	9	9	9	9	9	9	9
	LogOn	24	24	24	24	24	24	24

This is a simple way to see if there are any labels in columns 5-7 that predict the outcome. (answer: not really as the count for most events that can be interpreted this way is too low). Also I am try to see if there are any interesting correlations between labels.

```
In [18]: print len(df[3].unique()), len(df[4].unique())
```

```
3904 1385
```

Potentially too many variables to be used in analysis

```
In [19]: df["source_user"], df["source_domain"] = zip(*df[1].str.split('@').tolist())

In [20]: df["source_user"]=df["source_user"].str.rstrip('$')

In [21]: df["destination_user"], df["destination_domain"] = zip(*df[2].str.split('@').tolist())
         df["destination_user"]=df["destination_user"].str.rstrip('$')

In [22]: df['same_user']=(df['destination_user']==df['source_user'])
         df['same_domain']=(df['destination_domain']==df['source_domain'])

In [23]: df['same_user'].value_counts()

Out[23]: True     10348
         False      166
         Name: same_user, dtype: int64

In [24]: df['same_domain'].value_counts()

Out[24]: True     10432
         False       82
         Name: same_domain, dtype: int64

In [25]: df['source_domain'].unique()

Out[25]: array(['DOM1', 'C586', 'C457', '?', 'C15108', 'C46', 'C3758', 'C13281',
                'C4576', 'C13406', 'C467', 'C3653', 'C612', 'C4379', 'C1065',
                'C793', 'C625', 'C1672', 'C1731', 'C4227', 'C529', 'C3432', 'C2769',
                'C15089', 'DOM9', 'C21690', 'C17851', 'C2743', 'C832', 'C4695',
                'C2117', 'C13204', 'DOM3', 'DOM5', 'C1871', 'C12802', 'C5306',
                'C13183', 'C5894', 'C13052', 'C2606', 'C2106', 'C17222', 'C4883',
                'C5371', 'C5404', 'C15244', 'C4438', 'C11499', 'C16598', 'C1747',
                'C1909', 'C14332', 'C423', 'C20557', 'C21598', 'C8260', 'C8814',
                'C19497', 'C2925', 'C2866', 'C22616', 'C528', 'C9610', 'C4835',
                'C4803', 'C2121', 'C8683', 'C2459', 'C11145', 'C1786', 'C9873',
                'C10865', 'C2464', 'DOM55', 'C5334', 'C7533', 'C10747', 'C3967',
                'C20567', 'C22918', 'C17437', 'C13578', 'C15888', 'C4223', 'C12502',
                'C2198', 'C12203', 'C19485', 'C5347', 'C3025', 'C8113', 'C3898',
                'C17448', 'C25360', 'C23220', 'C14660', 'C4511', 'C21486', 'C9589',
                'C17577', 'C9782', 'C10730', 'C3258', 'C17094', 'C4189', 'C18240',
                'C14366', 'C12423', 'C17905', 'C4347', 'C5157', 'C19918', 'C12856',
                'C2370', 'C6629', 'C2004', 'C15611', 'C12029', 'C3778', 'C11400',
                'C14036', 'C19869', 'C7164', 'C5279', 'C12768', 'C15615', 'C8466',
                'C20102', 'C11090', 'C8670', 'C23237', 'C13913', 'C1854', 'C9150',
                'C8367', 'C2060', 'C6027', 'C20610', 'C3254', 'C11688', 'C23850',
                'C25486', 'C3824', 'C9762', 'C4929', 'C4481', 'C16473', 'C1760',
                'C10512', 'C15374', 'C3350', 'C11627', 'C11594', 'C13542', 'C18574',
                'C4421', 'C4640', 'C10944', 'C15613', 'C6037', 'C19202', 'C9067',
                'C16175', 'C16535', 'C11829', 'C12659', 'C3044', 'C11136', 'C3221',
                'C2424', 'C4971', 'C20997', 'C6639', 'C17954', 'C15937', 'C2608',
                'C6951', 'C6221', 'C18510', 'C6302', 'C4400', 'C11386', 'C23674',
                'C17135', 'C18430', 'C12777', 'C17608', 'C9822', 'C14438', 'C20682',
                'C20732', 'C3344', 'C9170', 'C14218', 'C18591', 'C20795', 'C7858',
```

```
       'C4716', 'C14395', 'C11525', 'C15624', 'C3980', 'C10554', 'C13416',
       'C8548', 'C8599', 'C25401', 'C20005', 'C17400', 'C24706', 'C6537',
       'C13933', 'C13218', 'C11508', 'C20931', 'C6183', 'C15535', 'C1761',
       'C5498', 'C6532', 'C13602', 'C22124', 'C2920', 'C20518', 'C20177',
       'C14891', 'C5422', 'C2472', 'C631', 'C14292', 'C7229', 'C5039',
       'C13292', 'C18207', 'C4603', 'C11798', 'C18359', 'C13929', 'C5410',
       'C4578', 'C4429', 'C6675', 'C2307', 'C5321', 'C12228', 'C19472',
       'C4319', 'C2895', 'C2261', 'C13779', 'C11564', 'C15988', 'C11138',
       'C2708', 'C11695', 'C19284', 'C15978', 'C21347', 'C4731', 'C2018',
       'C9080', 'C4593', 'C15491', 'C5353', 'C5402', 'C24397', 'C14962',
       'C23058', 'C11990', 'C4563', 'C1950', 'C10690', 'C21955', 'C11307',
       'C15119', 'C7089', 'C18708', 'C8685', 'C13868', 'C8579', 'C24311',
       'C3530', 'C4847', 'C2823', 'C9030', 'C11015', 'C3995', 'C9866',
       'C3897', 'C12919', 'C341', 'C23955', 'C21220', 'C18458', 'C15030',
       'C5433', 'C10267', 'C10860', 'C5145', 'C21093', 'C18580', 'C17078',
       'C3982', 'C10563', 'C4442', 'C6655', 'C8743', 'C19877', 'C8190',
       'C3837', 'C14114', 'C17652', 'C21539', 'C15462', 'C17295', 'C4767',
       'C5538', 'C3334', 'C4533', 'C10801', 'C4859', 'C9373', 'C13408',
       'C11541', 'C20625', 'C10732', 'C5123', 'C15436', 'C4059', 'C4962',
       'C9715', 'C18302', 'C16963', 'C5141', 'C6371', 'C3806', 'C2589',
       'C14905', 'C3730', 'C3336', 'C4007', 'C12425', 'C18180', 'C21405',
       'C19438', 'C3924', 'C14240', 'C1979', 'C4856', 'C25512', 'C9756',
       'C2672', 'C10041', 'C6462', 'C21877', 'C6005', 'C3089', 'C3945',
       'C23517', 'C1695', 'C3363', 'C4931', 'C15176', 'C10067', 'C12102',
       'C15145', 'C15626', 'C2981', 'C22008', 'C3103', 'C19679', 'C4112',
       'C18456', 'C2205', 'C12279', 'C4335', 'C8793', 'C15087', 'C16727',
       'C5472', 'C19719', 'C10770', 'C5288', 'C8976', 'C5220', 'C11731',
       'C8806', 'C21063', 'C24735', 'C2817', 'C6425', 'C15570', 'C8679',
       'C7616', 'C14271', 'C16277', 'C2690', 'C3583', 'C20613', 'C2629',
       'C3914', 'C8738', 'C4917', 'C16905', 'C24020', 'C13446', 'C20658',
       'C6443', 'C13439', 'C3454', 'C5780', 'C19361', 'C27033', 'C9048',
       'C7568', 'C5159', 'C19460', 'C4123', 'C11142', 'C19774', 'C3094',
       'C2562', 'C18921', 'C11547', 'C4623', 'C3149', 'C4864', 'C5375',
       'C17113', 'C14402', 'C4757', 'C14081', 'C16645', 'C4501'], dtype=object)
```

In [26]:
```
df['destination_domain'].unique()
```

Out[26]:
```
array(['DOM1', 'C586', 'C457', '?', 'C15108', 'C46', 'C3758', 'C13281',
       'C4576', 'C13406', 'C12913', 'C467', 'C3653', 'C612', 'C4379',
       'C1065', 'C15378', 'C625', 'C1672', 'C1731', 'C19776', 'C13352',
       'C4227', 'C529', 'C3432', 'C2769', 'DOM5', 'C10819', 'C15089',
       'DOM9', 'C21690', 'C17851', 'C2743', 'C4695', 'C2625', 'C2117',
       'C13204', 'DOM3', 'C832', 'C1871', 'C12802', 'C5306', 'C3495',
       'C13183', 'C5894', 'C13052', 'C2606', 'C2106', 'C17222', 'C4883',
       'C5371', 'C5404', 'C15314', 'C22758', 'C446', 'C15244', 'C4438',
       'C11499', 'C19004', 'C16598', 'C1747', 'C1909', 'C14332', 'C698',
       'C423', 'C5580', 'C7061', 'C9125', 'C20557', 'C21598', 'C8260',
       'C8814', 'C19497', 'C2925', 'C12652', 'C2866', 'C22616', 'C528',
       'C802', 'C9610', 'C4835', 'C4803', 'C2121', 'C561', 'C8683',
       'C2459', 'C11145', 'C1786', 'C9873', 'C10865', 'C2464', 'C21873',
       'C6558', 'C24871', 'C5334', 'C7533', 'C10747', 'C3967', 'C20567',
       'C22918', 'C17437', 'C13578', 'C12141', 'C15888', 'C4223', 'C12502',
       'C20889', 'C2198', 'C12203', 'C19485', 'C5347', 'C24733', 'C3025',
       'C12590', 'C8113', 'C3898', 'C17448', 'C25360', 'C23220', 'C14660',
       'C4511', 'C21486', 'C9589', 'C17577', 'C9506', 'C12283', 'C9782',
```

```
'C10730', 'C3258', 'C18401', 'C17094', 'C4189', 'C18240', 'C14366',
'C12423', 'C17905', 'C4347', 'C5157', 'C19918', 'C12856', 'C8800',
'C2370', 'C6629', 'C2004', 'C15611', 'C12029', 'C3778', 'C11400',
'C14036', 'C19869', 'C7164', 'C5279', 'C12768', 'C15615', 'C8466',
'C20102', 'C11090', 'C8670', 'C23237', 'C13913', 'C1854', 'C9150',
'C8367', 'C2060', 'C6027', 'C20610', 'C3254', 'C11688', 'C23850',
'C25486', 'C3824', 'C9762', 'C4929', 'C4481', 'C16473', 'C1760',
'C10512', 'C15374', 'C3350', 'C11627', 'C11594', 'C13542', 'C18574',
'C4421', 'C4640', 'C10944', 'C15613', 'C6037', 'C19202', 'C9067',
'C16175', 'C16535', 'C11829', 'C13670', 'C12659', 'C3044', 'C11136',
'C3221', 'C2424', 'C4971', 'C20997', 'C6639', 'C17954', 'C15937',
'C2608', 'C6951', 'C6221', 'C18510', 'C6302', 'C4400', 'C13160',
'C11386', 'C23674', 'C17135', 'C18430', 'C12777', 'C17608', 'C9822',
'C14438', 'C20682', 'C26', 'C172', 'C20732', 'C3344', 'C9170',
'C14218', 'C18591', 'C20795', 'C7858', 'C4716', 'C14395', 'C11525',
'C15624', 'C3980', 'C10554', 'C13416', 'C8548', 'C8599', 'C25401',
'C20005', 'C17400', 'C24706', 'C6537', 'C13933', 'C13218', 'C11508',
'C20931', 'C6183', 'C15535', 'C1761', 'C5498', 'C6532', 'C13602',
'C22124', 'C9623', 'C2920', 'C20518', 'C20177', 'C14891', 'C5422',
'C2472', 'C631', 'C14292', 'C7229', 'C5039', 'C13292', 'C18207',
'C4603', 'C11798', 'C18359', 'C13929', 'C5410', 'C4578', 'C23215',
'C4429', 'C6675', 'C339', 'C19402', 'C2307', 'C5321', 'C12228',
'C19472', 'C4319', 'C2895', 'C2261', 'C13779', 'C11564', 'C15988',
'C11138', 'C2708', 'C11695', 'C19284', 'C15978', 'C21347', 'C4731',
'C13', 'C2018', 'C9080', 'C4593', 'C15491', 'C5353', 'C5402',
'C24397', 'C14962', 'C23058', 'C11990', 'C4563', 'C1950', 'C10690',
'C21955', 'C11307', 'C15119', 'C7089', 'C18708', 'C8685', 'C13868',
'C8579', 'C24311', 'C3530', 'C4847', 'C2823', 'C9030', 'C11015',
'C3995', 'C9866', 'C20085', 'C3897', 'C12919', 'C22010', 'C341',
'C23955', 'C21220', 'C18458', 'C15030', 'C2957', 'C5433', 'C10267',
'C10860', 'C5145', 'C21093', 'C18580', 'C17078', 'C3982', 'C1184',
'C10563', 'C4442', 'C6655', 'C8743', 'C19877', 'C8190', 'C3837',
'C14114', 'C22315', 'C9274', 'C17652', 'C21539', 'C15462', 'C17295',
'C4767', 'C5538', 'C3334', 'C4533', 'C10801', 'C4859', 'C9373',
'C13408', 'C8571', 'C11541', 'C20625', 'C10732', 'C5123', 'C15436',
'C4059', 'C4962', 'C9715', 'C18302', 'C16963', 'C25151', 'C24944',
'C5141', 'C10591', 'C673', 'C6371', 'C3806', 'C2589', 'C14905',
'C3730', 'C3336', 'C4007', 'C12425', 'C18180', 'C21405', 'C19438',
'C1291', 'C3924', 'C14240', 'C1979', 'C4856', 'C25512', 'C9756',
'C2672', 'C10041', 'C6462', 'C7174', 'C21877', 'C6005', 'C3089',
'C1266', 'C3945', 'C23517', 'C1695', 'C3363', 'C20679', 'C4931',
'C15176', 'C10067', 'C1371', 'C12102', 'C15145', 'C15626', 'C2981',
'C22008', 'C3103', 'C19679', 'C4112', 'C18456', 'C2205', 'C12279',
'C10455', 'C4335', 'C8793', 'C15087', 'C6594', 'C16727', 'C5472',
'C19719', 'C10770', 'C1373', 'C5288', 'C8976', 'C5220', 'C11731',
'C8806', 'C21063', 'C24735', 'C2817', 'C6425', 'C15570', 'C8679',
'C7616', 'C14271', 'C16277', 'C2690', 'C3583', 'C20613', 'C2629',
'C3914', 'C8738', 'C4917', 'C20406', 'C16905', 'C24020', 'C12116',
'C13446', 'C20658', 'C102', 'C6443', 'C13439', 'C3454', 'C9216',
'C5780', 'C19361', 'C27033', 'C9048', 'C7568', 'C1495', 'C5159',
'C19460', 'C4123', 'C11142', 'C19774', 'C3094', 'C2562', 'C18921',
'C11547', 'C26929', 'C3290', 'C1788', 'C4623', 'C2654', 'C3149',
'C4864', 'C5375', 'C17113', 'C14402', 'C4757', 'C14081', 'C16645',
'C4501'], dtype=object)
```

```
In [27]: df['source_user'].unique()

Out[27]: array(['C2374', 'C1652', 'C538', ..., 'U3145', 'C18896', 'C25126'], dtype=object)

In [28]: df['destination_user'].unique()

Out[28]: array(['C2374', 'C1652', 'C538', ..., 'U3145', 'C18896', 'C25126'], dtype=object)
```

Potentially too many variable. I now want to explore what users I have in addition to C-numbers and U-numbers. (C=computer and U=user?)

```
In [29]: good=df[~df.source_user.str.startswith("U")]
         good=good.source_user[~good.source_user.str.startswith('C')]
         good.unique()

Out[29]: array(['ANONYMOUS LOGON', 'LOCAL SERVICE', 'NETWORK SERVICE', 'SYSTEM'], dtype=object)

In [30]: good=df[~np.logical_or(df.destination_user.str.startswith("U"), df.destination_user.str.startsw
         #good=good.destination_user[~good.destination_user.str.contains('C')]
         good.destination_user.unique()

Out[30]: array(['ANONYMOUS LOGON', 'LOCAL SERVICE', 'SYSTEM', 'NETWORK SERVICE'], dtype=object)
```

Idea: one can expand this column into into 6 categories: C-users, U-users, 'ANONYMOUS LOGON', 'LOCAL_SERVICE', 'SYSTEM', 'NETWORK SERVICE'

```
In [31]: dd=df['destination_domain'].str.startswith('C')
         print min(df['destination_domain'][dd].str.slice(1).astype(int)), max(df['destination_domain']
         dd=df[~df.destination_domain.str.startswith('C')]
         print dd.destination_domain.unique()

13 27033
['DOM1' '?' 'DOM5' 'DOM9' 'DOM3']

In [33]: sd=df['source_domain'].str.startswith('C')
         print min(df['source_domain'][sd].str.slice(1).astype(int)), max(df['source_domain'][sd].str.sl
         sd=df[~df.source_domain.str.startswith('C')]
         print sd.source_domain.unique()

46 27033
['DOM1' '?' 'DOM9' 'DOM3' 'DOM5' 'DOM55']
```

6 Conclusion

This dataset contains columns of categorical data (aside from time). To work with this data, each label should be converted to its own column with values 1 (True) if the label applies and 0 (False) otherwise. Some columns (5-7) contain ~10 labels, where as other columns contain ten thousands of labels. I will ignore the 2nd class of labels on the first pass. Instead I will consider when these labels coincide. This way I will prevent my set of features from exploding. Also the 2nd class of labels most likely comes from some ordering of computers and users in the lab. Considering if one wants to authenticate to the same computer or to a different computer should matter more for authentication success than specific computer label.

```
In [34]: df['source_user_comp_same']=(df[3]==df['source_user'])
         df['destination_user_comp_same']=(df['destination_user']==df[4])
         df['same_comp']=(df[3]==df[4])
         df['source_domain_comp_same']=(df[3]==df['source_domain'])
         df['destination_domain_comp_same']=(df['destination_domain']==df[4])
```

```
In [35]: df['source_user_comp_same'].value_counts()

Out[35]: False    7481
         True     3033
         Name: source_user_comp_same, dtype: int64

In [36]: df['destination_user_comp_same'].value_counts()

Out[36]: False    9677
         True      837
         Name: destination_user_comp_same, dtype: int64

In [37]: df['same_comp'].value_counts()

Out[37]: True     5955
         False    4559
         Name: same_comp, dtype: int64

In [38]: df['source_domain_comp_same'].value_counts()

Out[38]: False    9936
         True      578
         Name: source_domain_comp_same, dtype: int64

In [39]: df['destination_domain_comp_same'].value_counts()

Out[39]: False    9665
         True      849
         Name: destination_domain_comp_same, dtype: int64

In [ ]:
```