EDA

EDA AVProductsInstalled

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
```

In [2]:
```
%time train = pd.read_csv("train.csv")
```

```
<string>:2: DtypeWarning: Columns (28) have mixed types. Specify dty
pe option on import or set low_memory=False.

CPU times: user 1min 38s, sys: 34.2 s, total: 2min 12s
Wall time: 2min 1s
```

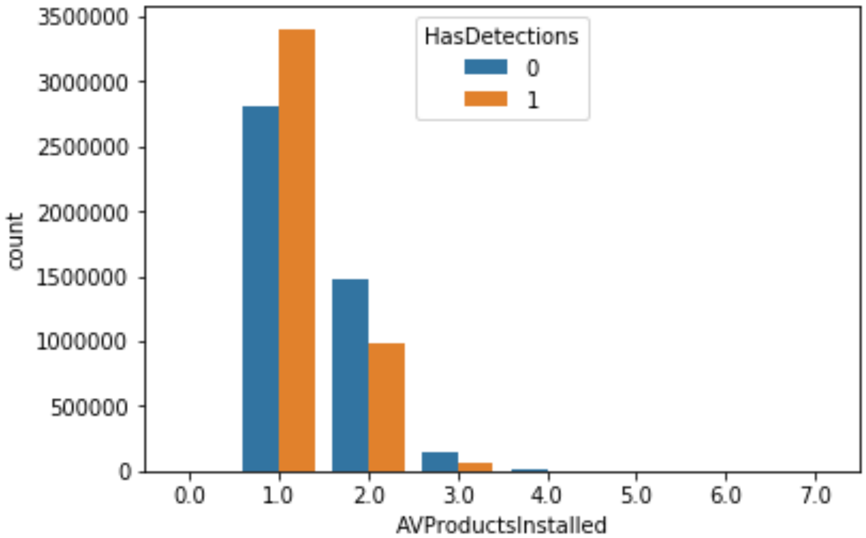
In [3]:
```
train.head()
```

Out[3]:

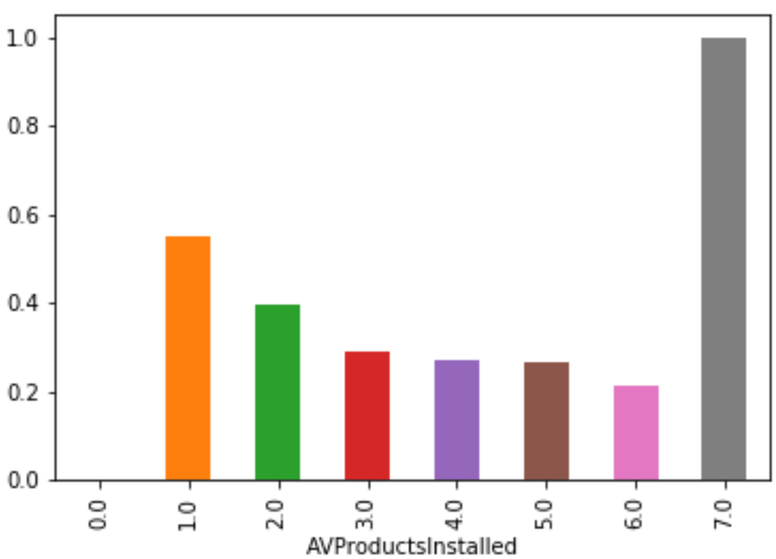
	MachineIdentifier	ProductName	EngineVersion	AppVersion	AvSigVer
0	0000028988387b115f69f31a3bf04f09	win8defender	1.1.15100.1	4.18.1807.18075	1.273.17
1	000007535c3f730efa9ea0b7ef1bd645	win8defender	1.1.14600.4	4.13.17134.1	1.263.
2	000007905a28d863f6d0d597892cd692	win8defender	1.1.15100.1	4.18.1807.18075	1.273.13
3	00000b11598a75ea8ba1beea8459149f	win8defender	1.1.15100.1	4.18.1807.18075	1.273.15
4	000014a5f00daa18e76b81417eeb99fc	win8defender	1.1.15100.1	4.18.1807.18075	1.273.13

5 rows × 83 columns

In [23]:
```
ax = plt.axes()
sns.countplot(x='AVProductsInstalled', hue = 'HasDetections', data = t
rain, ax=ax);
```



In [29]:
```
## plot the ratio of HasDetections grouped by # of AV products install
ed
ax = plt.axes()
ratio_hasdetection = train.groupby(['AVProductsInstalled']).HasDetecti
ons.apply(lambda x: sum(x)/len(x))
ratio_hasdetection.plot(kind = 'bar', ax = ax);
```



In [16]:
```
num = train.groupby(['AVProductsInstalled']).HasDetections.count()
```

In [17]:
```
## How to normalize the ratio so that the frequency don't take into ef
fect?
```

In [18]:
```
df = pd.DataFrame(columns = ['counts', 'ratio'])
```

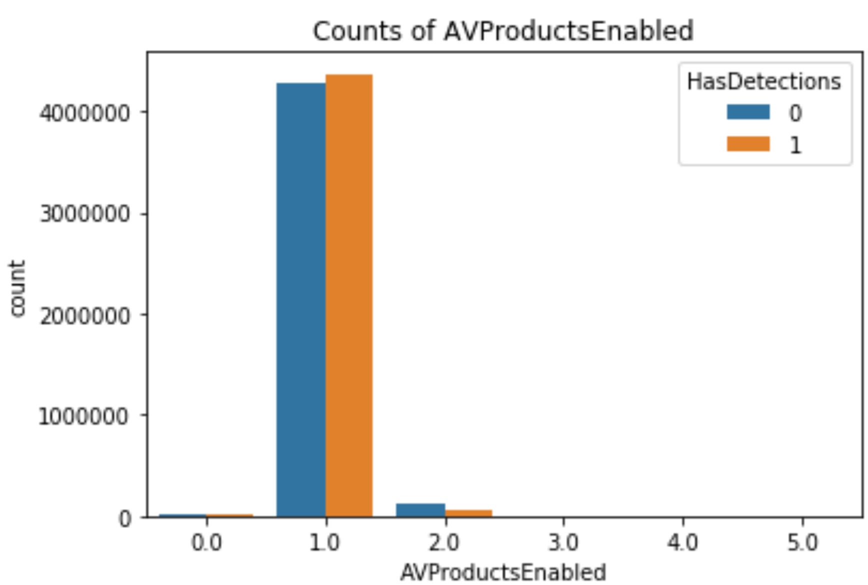
In [26]:
```
df['counts'] = num.astype('int')
df['ratio'] = ratio_hasdetection
```

In [27]:
```
df.T
```

Out[27]:

AVProductsInstalled	0.0	1.0	2.0	3.0	4.0	5.0
counts	1.0	6.208893e+06	2.459008e+06	208103.000000	8757.000000	471.000000
ratio	0.0	5.485806e-01	3.969064e-01	0.291596	0.270755	0.265393

In [21]:
```
ax = plt.axes()
sns.countplot(x='AVProductsEnabled', hue = 'HasDetections', data = tra
in);
ax.set_title('Counts of AVProductsEnabled');
```



In [19]:
```
train['AVProductsInstalled'].unique()
```

Out[19]:
```
array([ 1.,  2.,  3.,  5., nan,  4.,  6.,  7.,  0.])
```

In [40]:
```
train[['AVProductsInstalled', 'AVProductsEnabled', 'HasDetections']].c
orr(method = 'spearman')
```

Out[40]:

	AVProductsInstalled	AVProductsEnabled	HasDetections
AVProductsInstalled	1.000000	0.238208	-0.149501
AVProductsEnabled	0.238208	1.000000	-0.042343
HasDetections	-0.149501	-0.042343	1.000000

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scipy.stats

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline
```

In [2]:
```
train = pd.read_csv('train.csv', usecols = ['Census_IsTouchEnabled', 'Platf
train.head()
```

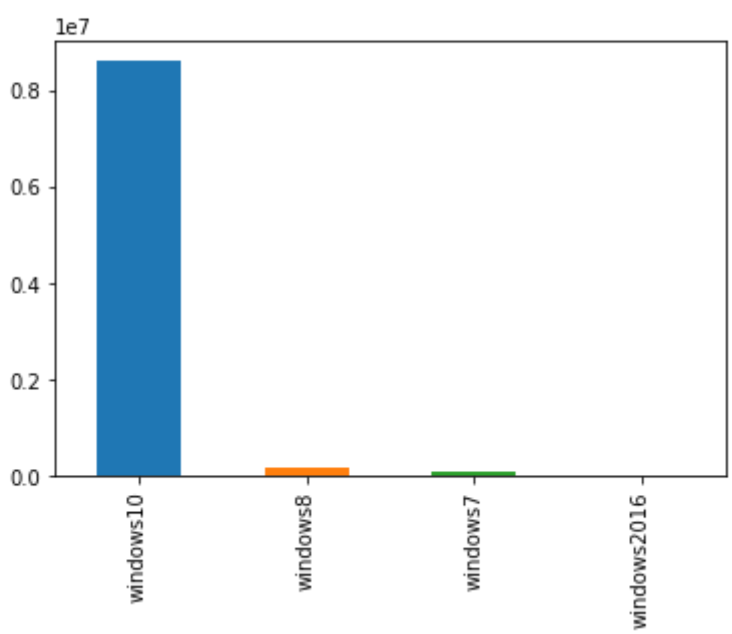
Out[2]:

	Platform	SkuEdition	Census_IsTouchEnabled	HasDetections
0	windows10	Pro	0	0
1	windows10	Pro	0	0
2	windows10	Home	0	0
3	windows10	Pro	0	1
4	windows10	Home	0	1

In [5]:
```
val_counts  = train['Platform'].value_counts()
```

In [6]:
```
val_counts.plot(kind = 'bar')
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1a243e0898>



In [9]:
```
pct = train.groupby('Platform', as_index = False)['HasDetections'].mean()
pct
```

Out[9]:

	Platform	HasDetections
0	windows10	0.500032
1	windows2016	0.349593
2	windows7	0.486511
3	windows8	0.506720

In [10]:
```
counts = train['Platform'].value_counts().reset_index(drop = False)
counts.columns = ['Platform', 'Count']
counts
```

Out[10]:

	Platform	Count
0	windows10	8618715
1	windows8	194508
2	windows7	93889
3	windows2016	14371

In [11]:
```
combined = counts.merge(pct, on = 'Platform', how = 'outer')
combined
```

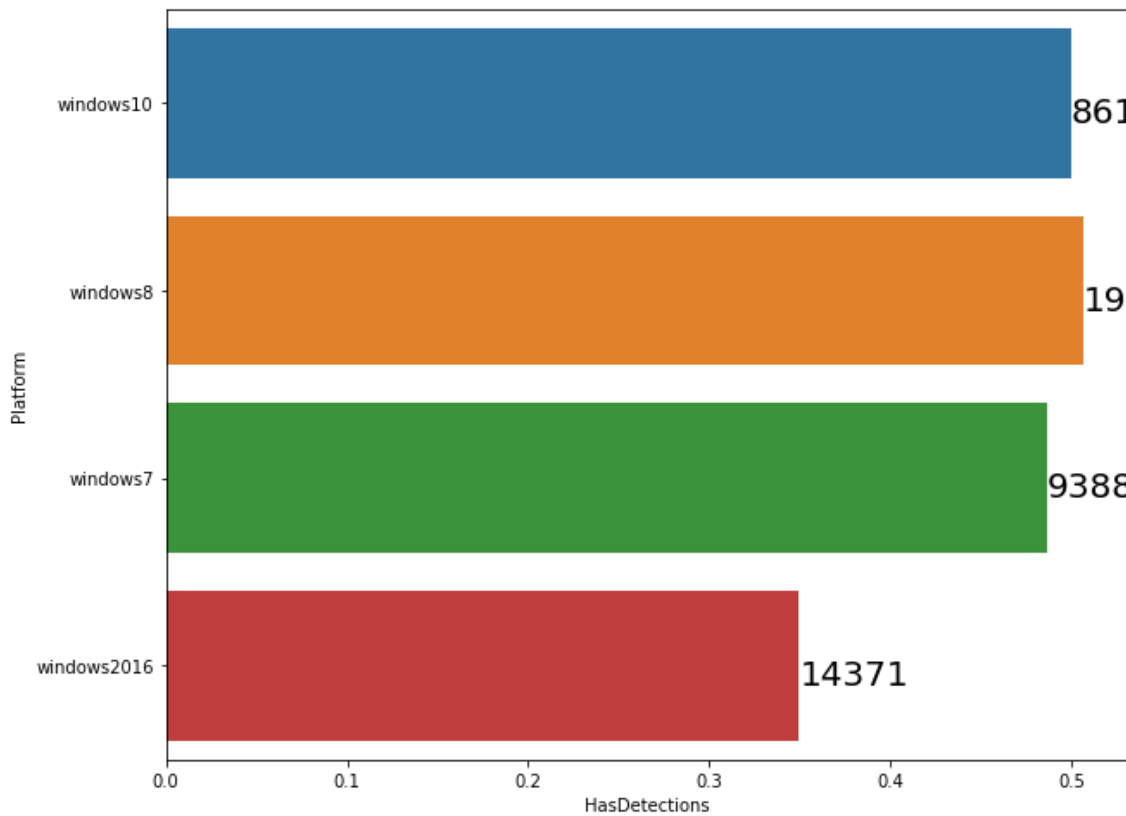
Out[11]:

	Platform	Count	HasDetections
0	windows10	8618715	0.500032
1	windows8	194508	0.506720
2	windows7	93889	0.486511
3	windows2016	14371	0.349593

In [11]:
```
train['Platform'].value_counts()
```

Out[11]:
```
windows10      8618715
windows8        194508
windows7         93889
windows2016      14371
Name: Platform, dtype: int64
```

In [38]:
```
counts = train['Platform'].value_counts().reset_index(drop = False)
counts.columns = ['Platform', 'Count']
pct = train.groupby('Platform', as_index = False)['HasDetections'].mean()

combined = counts.merge(pct, on = 'Platform', how = 'outer')

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
combined['Platform'] = combined['Platform'].astype('category')
# sns.barplot(ax=ax, x='HasDetections', y=col, data=cat_percent, order=cat_
sns.barplot(ax=ax, x='HasDetections', y='Platform', data=combined, order=cc

for i, p in enumerate(ax.patches):
        ax.annotate('{}'.format(combined['Count'].values[i]), (p.get_width(
```



In [15]:
```
train['HasDetections'].mean()
```

Out[15]: 0.49979269141688665

In [41]:
```
mean_rate = train['HasDetections'].mean()
exp_counts = combined['Count'] * mean_rate
obs_counts = combined['Count'] * combined['HasDetections']
scipy.stats.chisquare(f_obs = obs_counts, f_exp = exp_counts, ddof = 1)
```

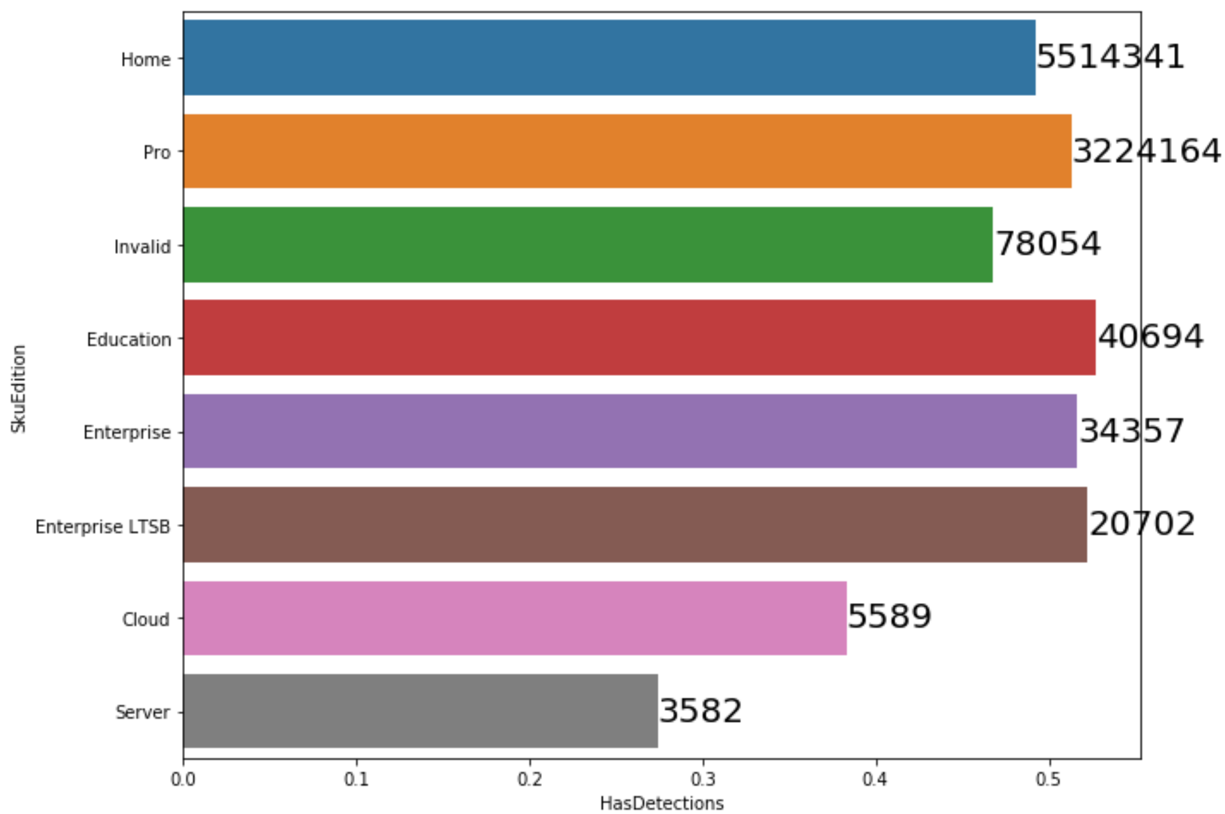
Out[41]: Power_divergenceResult(statistic=701.48418163299, pvalue=4.72765076258460
3e-153)

In [42]:
```
pd.DataFrame(data = {'expected': exp_counts.apply(lambda x: str(np.round(x,
```

Out[42]:

Platform	windows10	windows8	windows7	windows2016
expected	4307570.77	97213.68	46925.04	7182.52
observed	4309629	98561	45678	5024

In [22]:
```
counts = train['SkuEdition'].value_counts().reset_index(drop = False)
counts.columns = ['SkuEdition', 'Count']
pct = train.groupby('SkuEdition', as_index = False)['HasDetections'].mean()

combined = counts.merge(pct, on = 'SkuEdition', how = 'outer')

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
combined['SkuEdition'] = combined['SkuEdition'].astype('category')
# sns.barplot(ax=ax, x='HasDetections', y=col, data=cat_percent, order=cat_
sns.barplot(ax=ax, x='HasDetections', y='SkuEdition', data=combined, order=

for i, p in enumerate(ax.patches):
        ax.annotate('{}'.format(combined['Count'].values[i]), (p.get_width(
```



In [23]:
```
combined
```

Out[23]:

	SkuEdition	Count	HasDetections
0	Home	5514341	0.492266
1	Pro	3224164	0.513226
2	Invalid	78054	0.467625
3	Education	40694	0.526982
4	Enterprise	34357	0.516721
5	Enterprise LTSB	20702	0.522655
6	Cloud	5589	0.383432
7	Server	3582	0.274149

In [25]:
```
mean_rate = train['HasDetections'].mean()
exp_counts = combined['Count']*mean_rate
obs_counts = combined['Count'] * combined['HasDetections']
scipy.stats.chisquare(f_obs = obs_counts, f_exp = exp_counts, ddof = 0)
```

Out[25]:
```
Power_divergenceResult(statistic=2568.6692072153674, pvalue=0.0)
```

In [37]:
```
pd.DataFrame(data = {'expected': exp_counts.apply(lambda x: str(np.round(x,
```

Out[37]:

SkuEdition	Home	Pro	Invalid	Education	Enterprise	Enterprise LTSB	Cloud	Server
expected	2756027.33	1611413.6	39010.82	20338.56	17171.38	10346.71	2793.34	1790.26
observed	2714523	1654726	36500	21445	17753	10820	2143	982

In [51]:

In [17]:
```
cols = ['ProductName',
 'EngineVersion',
 'AvSigVersion',
 'Platform',
 'Processor',
 'OsVer',
 'OsPlatformSubRelease',
 'OsBuildLab',
 'SkuEdition',
 'Census_MDC2FormFactor',
 'Census_ChassisTypeName',
 'Census_OSVersion',
 'Census_OSBranch',
 'Census_OSEdition',
 'Census_OSSkuName',
 'AVProductStatesIdentifier',
 'AVProductsInstalled',
 'HasTpm',
 'OsSuite',
 'IsProtected',
 'IeVerIdentifier',
 'Census_ProcessorCoreCount',
 'Census_ProcessorManufacturerIdentifier',
 'Census_ProcessorModelIdentifier',
 'Census_PrimaryDiskTotalCapacity',
 'Census_SystemVolumeTotalCapacity',
 'Census_HasOpticalDiskDrive',
 'Census_InternalPrimaryDisplayResolutionHorizontal',
 'Census_InternalPrimaryDisplayResolutionVertical',
 'Census_OSBuildRevision',
 'Census_IsSecureBootEnabled',
 'Census_IsTouchEnabled',
 'Census_IsAlwaysOnAlwaysConnectedCapable',
 'Wdft_IsGamer']
cleaned = pd.read_csv('cleaned_sampled_data.csv')
cleaned.head().T
```

Out[17]:

	0	
Unnamed: 0	1355933	
ProductName	win8defender	
EngineVersion	1.1.15200.1	
AppVersion	4.18.1807.18075	
AvSigVersion	1.275.1582.0	
Platform	windows10	
Processor	x64	
OsVer	10.0.0.0	
OsPlatformSubRelease	rs3	
OsBuildLab	16299.15.amd64fre.rs3_release.170928-1534	16299.431.an

	0
SkuEdition	Home
Census_MDC2FormFactor	Convertible
Census_PrimaryDiskTypeName	HDD
Census_ChassisTypeName	Notebook
Census_PowerPlatformRoleName	Mobile
Census_OSVersion	10.0.16299.192
Census_OSArchitecture	amd64
Census_OSBranch	rs3_release
Census_OSEdition	Core
Census_OSSkuName	CORE
Census_OSInstallTypeName	Upgrade
Census_OSWUAutoUpdateOptionsName	Notify
Census_GenuineStateName	IS_GENUINE
Census_ActivationChannel	Retail
Census_FlightRing	Retail
RtpStateBitfield	7
IsSxsPassiveMode	0
AVProductStatesIdentifier	53447
AVProductsInstalled	1
AVProductsEnabled	1
...	...
LocaleEnglishNameIdentifier	71
OsBuild	16299
OsSuite	768
IsProtected	1
IeVerIdentifier	111
Firewall	1
Census_OEMNameIdentifier	2206
Census_OEMModelIdentifier	244535
Census_ProcessorCoreCount	4
Census_ProcessorManufacturerIdentifier	5
Census_ProcessorModelIdentifier	3392
Census_PrimaryDiskTotalCapacity	476940
Census_SystemVolumeTotalCapacity	452837
Census_HasOpticalDiskDrive	0

	0
Census_TotalPhysicalRAM	8192
Census_InternalPrimaryDiagonalDisplaySizeInInches	11.6
Census_InternalPrimaryDisplayResolutionHorizontal	1366
Census_InternalPrimaryDisplayResolutionVertical	768
Census_InternalBatteryNumberOfCharges	0
Census_OSBuildRevision	192
Census_OSUILocaleIdentifier	31
Census_FirmwareManufacturerIdentifier	554
Census_FirmwareVersionIdentifier	33120
Census_IsSecureBootEnabled	1
Census_IsTouchEnabled	1
Census_IsPenCapable	0
Census_IsAlwaysOnAlwaysConnectedCapable	0
Wdft_IsGamer	0
Wdft_RegionIdentifier	1
HasDetections	NaN

64 rows × 5 columns

In [52]:
```
cleaned.dtypes.index[:15].tolist()
```

Out[52]:
```
['ProductName',
 'EngineVersion',
 'AvSigVersion',
 'Platform',
 'Processor',
 'OsVer',
 'OsPlatformSubRelease',
 'OsBuildLab',
 'SkuEdition',
 'Census_MDC2FormFactor',
 'Census_ChassisTypeName',
 'Census_OSVersion',
 'Census_OSBranch',
 'Census_OSEdition',
 'Census_OSSkuName']
```

In [50]:
```
{col:cleaned[col].unique() for col in cleaned.dtypes.index[:15]}
```

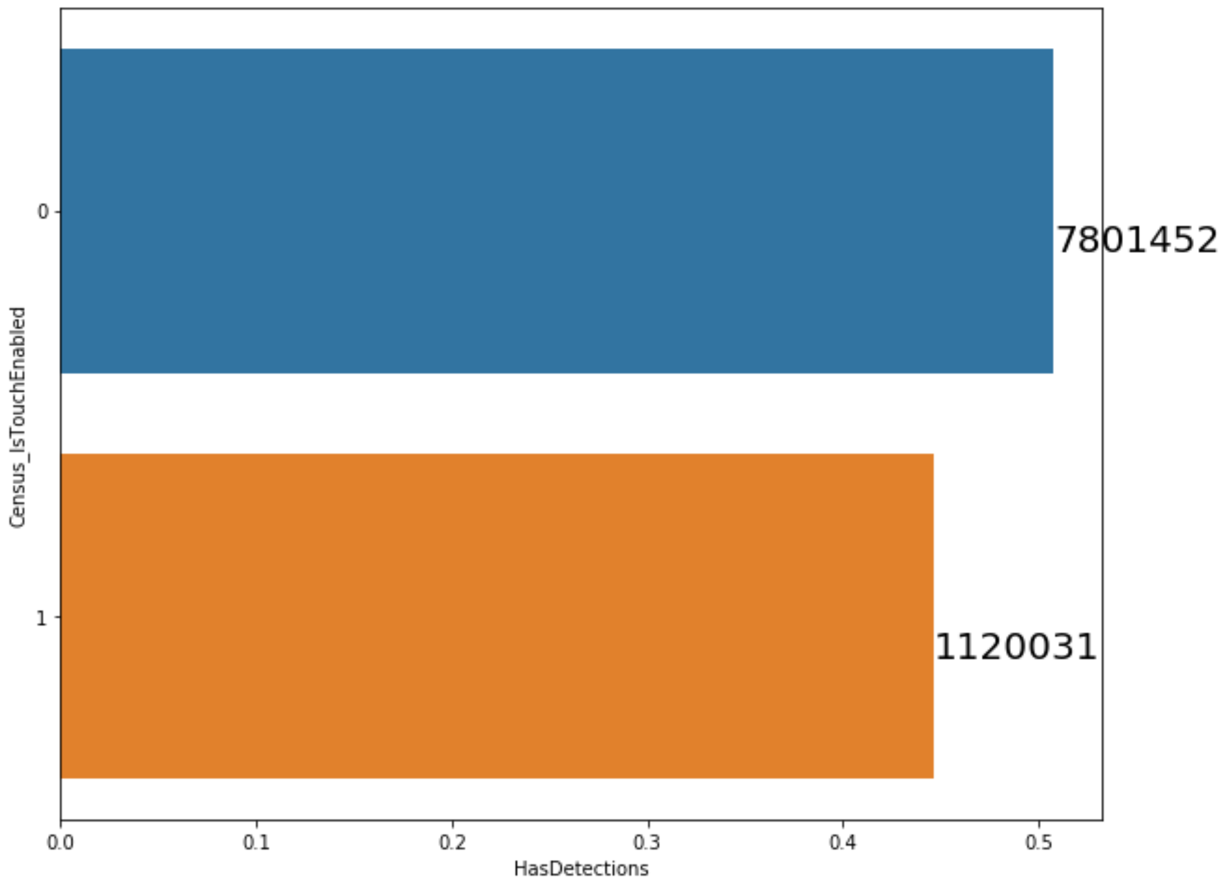
Out[50]:
```
{'ProductName': array(['win8defender', 'mse'], dtype=object),
 'EngineVersion': array(['1.1.15200.1', '1.1.15300.6', '1.1.15000.2', '1.
1.15100.1',
        '1.1.14500.5', '1.1.13903.0', '1.1.13202.0', '1.1.14800.3',
        '1.1.14600.4', '1.1.15300.5', '1.1.14700.5', '1.1.14901.4',
        '1.1.14104.0', '1.1.13303.0', '1.1.13804.0', '1.1.14405.2',
        '1.1.13504.0', '1.1.13701.0', '1.1.13407.0', '1.1.13704.0',
        '1.1.14202.0', '1.1.14700.4', '1.1.13000.0', '1.1.14305.0',
        '1.1.12902.0', '1.1.14306.0', '1.1.14003.0', '1.1.14500.2',
        '1.1.13103.0', '1.1.15000.1', '1.1.14901.3', '1.1.13601.0',
        '1.1.14103.0', '1.1.12101.0', '1.1.14800.1', '1.1.12805.0'],
       dtype=object),
 'AvSigVersion': array(['1.275.1582.0', '1.275.166.0', '1.275.278.0',
..., '1.261.1079.0',
        '1.271.55.0', '1.265.862.0'], dtype=object),
 'Platform': array(['windows10', 'windows8', 'windows7', 'windows2016'],
dtype=object),
 'Processor': array(['x64', 'x86'], dtype=object),
 'OsVer': array(['10.0.0.0', '6.3.0.0', '6.1.1.0', '6.1.0.0'], dtype=obje
```

In [ ]:
```
onehot = OneHotEncoder()
cat_features = cleaned.dtypes.index[:15].tolist()

num_features = cleaned.dtypes.index[15:].tolist()
```

HasDetections all NaN? Use whole dataset or subsample? Think about how to preprocess different columns!!!

In [ ]:
```
```

In [3]:
```
counts = train['Census_IsTouchEnabled'].value_counts().reset_index(drop = F
counts.columns = ['Census_IsTouchEnabled', 'Count']
pct = train.groupby('Census_IsTouchEnabled', as_index = False)['HasDetectic

combined = counts.merge(pct, on = 'Census_IsTouchEnabled', how = 'outer')

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
combined['Census_IsTouchEnabled'] = combined['Census_IsTouchEnabled'].astyp
# sns.barplot(ax=ax, x='HasDetections', y=col, data=cat_percent, order=cat_
sns.barplot(ax=ax, x='HasDetections', y='Census_IsTouchEnabled', data=combi

for i, p in enumerate(ax.patches):
        ax.annotate('{}'.format(combined['Count'].values[i]), (p.get_width(
```



In [4]:
```
combined
```

Out[4]:

	Census_IsTouchEnabled	Count	HasDetections
0	0	7801452	0.507448
1	1	1120031	0.446467

In [10]:
```
p1 = combined.loc[0]['HasDetections']
n1 = combined.loc[0]['Count']
SE1 = np.sqrt(p1 * (1 - p1) / n1)
interval1 = [p1 - 1.96 * SE1, p1 + 1.96 * SE1]
print('95% confidence interval for not virtual device ' + str(interval1))
```

```
95% confidence interval for not virtual device [0.5070976605321477, 0.507
7993099292485]
```

In [11]:
```
p2 = combined.loc[1]['HasDetections']
n2 = combined.loc[1]['Count']
SE2 = np.sqrt(p2 * (1 - p2) / n2)
interval2 = [p2 - 1.96 * SE2, p2 + 1.96 * SE2]
print('95% confidence interval for virtual device ' + str(interval2))
```

```
95% confidence interval for virtual device [0.44554642927594257, 0.447387
7841520786]
```

In [12]:
```
def two_prop_test(p1, p2, n1, n2):
    pooled_p = (p1 * n1 + p2 * n2) / (n1 + n2)
    SE = np.sqrt(pooled_p* ( 1 - pooled_p ) * ((1/n1) + (1/n2)))
    z = (p1 - p2) / SE
    pval = 2 * min(scipy.stats.norm.cdf(z), 1-scipy.stats.norm.cdf(z))
    return pval, z
```

In [8]:
```
scipy.stats.norm.cdf(1)
```

Out[8]:
```
0.8413447460685429
```

In [13]:
```
two_prop_test(p1, p2, n1, n2)
```

Out[13]:
```
(0.0, 120.70117181293827)
```

In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats
```

In [2]:
```
datapath = './train.csv'
df = pd.read_csv(datapath)
df.dropna(subset=['SMode', 'HasDetections'],inplace=True)
df.head()
```

```
C:\Users\14481\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2785: DtypeWarning: Columns (28) have mixed types. Specify dtype option on import or set low_memor
y=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	MachineIdentifier	ProductName	EngineVersion	AppVersion	AvSigVersion	IsBeta	RtpStateBitfield	IsSxsPassiveMode	DefaultBrowsersIdentifier	AVProductStatesIdentifier	...	Census_FirmwareVer
0	0000028988387b115f69f31a3bf04f09	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1735.0	0	7.0	0	NaN	53447.0	...	
1	000007535c3f730efa9ea0b7ef1bd645	win8defender	1.1.14600.4	4.13.17134.1	1.263.48.0	0	7.0	0	NaN	53447.0	...	
2	000007905a28d863f6d0d597892cd692	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1341.0	0	7.0	0	NaN	53447.0	...	
3	00000b11598a75ea8ba1beea8459149f	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1527.0	0	7.0	0	NaN	53447.0	...	
4	000014a5f00daa18e76b81417eeb99fc	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1379.0	0	7.0	0	NaN	53447.0	...	

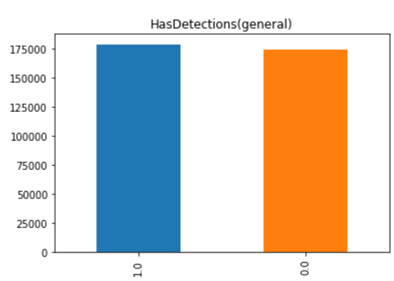
5 rows × 83 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['MachineIdentifier', 'ProductName', 'EngineVersion', 'AppVersion',
       'AvSigVersion', 'IsBeta', 'RtpStateBitfield', 'IsSxsPassiveMode',
       'DefaultBrowsersIdentifier', 'AVProductStatesIdentifier',
       'AVProductsInstalled', 'AVProductsEnabled', 'HasTpm',
       'CountryIdentifier', 'CityIdentifier', 'OrganizationIdentifier',
       'GeoNameIdentifier', 'LocaleEnglishNameIdentifier', 'Platform',
       'Processor', 'OsVer', 'OsBuild', 'OsSuite', 'OsPlatformSubRelease',
       'OsBuildLab', 'SkuEdition', 'IsProtected', 'AutoSampleOptIn', 'PuaMode',
       'SMode', 'IeVerIdentifier', 'SmartScreen', 'Firewall', 'UacLuaenable',
       'Census_MDC2FormFactor', 'Census_DeviceFamily',
       'Census_OEMNameIdentifier', 'Census_OEMModelIdentifier',
       'Census_ProcessorCoreCount', 'Census_ProcessorManufacturerIdentifier',
       'Census_ProcessorModelIdentifier', 'Census_ProcessorClass',
       'Census_PrimaryDiskTotalCapacity', 'Census_PrimaryDiskTypeName',
       'Census_SystemVolumeTotalCapacity', 'Census_HasOpticalDiskDrive',
       'Census_TotalPhysicalRAM', 'Census_ChassisTypeName',
       'Census_InternalPrimaryDiagonalDisplaySizeInInches',
       'Census_InternalPrimaryDisplayResolutionHorizontal',
       'Census_InternalPrimaryDisplayResolutionVertical',
       'Census_PowerPlatformRoleName', 'Census_InternalBatteryType',
       'Census_InternalBatteryNumberOfCharges', 'Census_OSVersion',
       'Census_OSArchitecture', 'Census_OSBranch', 'Census_OSBuildNumber',
       'Census_OSBuildRevision', 'Census_OSEdition', 'Census_OSSkuName',
       'Census_OSInstallTypeName', 'Census_OSInstallLanguageIdentifier',
       'Census_OSUILocaleIdentifier', 'Census_OSWUAutoUpdateOptionsName',
       'Census_IsPortableOperatingSystem', 'Census_GenuineStateName',
       'Census_ActivationChannel', 'Census_IsFlightingInternal',
       'Census_IsFlightsDisabled', 'Census_FlightRing',
       'Census_ThresholdOptIn', 'Census_FirmwareManufacturerIdentifier',
       'Census_FirmwareVersionIdentifier', 'Census_IsSecureBootEnabled',
       'Census_IsWIMBootEnabled', 'Census_IsVirtualDevice',
       'Census_IsTouchEnabled', 'Census_IsPenCapable',
       'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer',
       'Wdft_RegionIdentifier', 'HasDetections'],
      dtype='object')
```

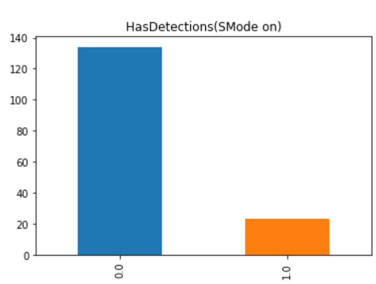
In [4]:
```
df['HasDetections'].value_counts().plot.bar()
plt.title('HasDetections(general)')
```

Out[4]:
```
Text(0.5,1,'HasDetections(general)')
```



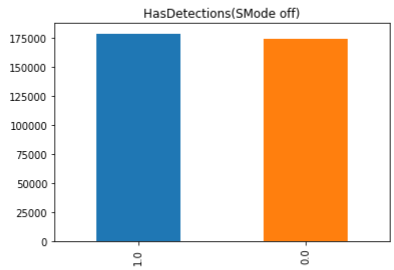
In [5]:
```
df[df['SMode']== 1]['HasDetections'].value_counts().plot.bar()
plt.title('HasDetections(SMode on)')
count1 = df[df['SMode']== 1]['HasDetections'].value_counts()
print("The infection ratio with SMode on is ", count1[1]/count1.sum())
```

```
The infection ratio with SMode on is  0.1464968152866242
```

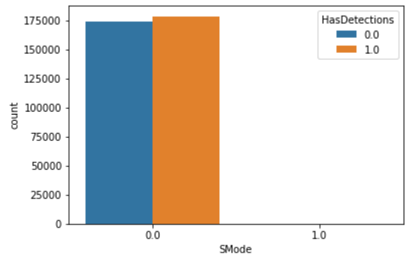


In [6]:
```
df[df['SMode']== 0]['HasDetections'].value_counts().plot.bar()
plt.title('HasDetections(SMode off)')
count2 = df[df['SMode']== 0]['HasDetections'].value_counts()
print("The infection ratio with SMode off is ", count2[1]/count2.sum())
```

```
The infection ratio with SMode off is  0.5063729364802806
```



In [7]:
```
sns.countplot(x='SMode', hue='HasDetections', data=df)
plt.show()
```



In [8]:
```
corr = df[['SMode','HasDetections']].corr()['HasDetections']
```

In [9]:
```
abs(corr).sort_values(ascending=False)
```

Out[9]:
```
HasDetections    1.000000
SMode            0.015174
Name: HasDetections, dtype: float64
```

In [10]:
```
SMode_observed = df['SMode'].value_counts().fillna(0)
SMode_observed.to_frame()
```

Out[10]:

	SMode
0.0	352977
1.0	157

In [11]:
```
SMode_expected = pd.Series(data = [0.9995,0.0005], index = ['0.0', '1.0']) * len(df)
SMode_expected.to_frame()
```

Out[11]:

	0
0.0	352957.433
1.0	176.567

In [12]:
```
scipy.stats.chisquare(SMode_observed, SMode_expected, ddof=0)
```

Out[12]:
```
Power_divergenceResult(statistic=2.1694825109406315, pvalue=0.1407735985998993)
```

In [13]:
```
df[['SMode', 'HasDetections']].corr(method = 'spearman')
```

Out[13]:

	SMode	HasDetections
SMode	1.000000	-0.015174
HasDetections	-0.015174	1.000000

In [ ]:

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
tbl = pd.read_csv('train.csv',usecols=['Census_InternalPrimaryDisplayResolutionHorizontal','Census_InternalPrimaryDisplayResolutionVertical','HasDetections'])
```

In [6]:

```
resolution = tbl
resolution = resolution.rename({'Census_InternalPrimaryDisplayResolutionHorizontal':'hresolution','Census_InternalPrimaryDisplayResolutionVertical':'vresolution','HasDetections':'affected'},axis=1)
resolution = resolution[resolution.hresolution > 0]
```

In [8]:

```
resolution.groupby('affected').hresolution.describe()
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
affected								
0	4439474.0	1536.002683	369.248635	144.0	1366.0	1366.0	1920.0	12288.0
1	4434867.0	1559.496495	367.007207	200.0	1366.0	1366.0	1920.0	12288.0

In [9]:

```
resolution.hresolution.nunique()
```

Out[9]:

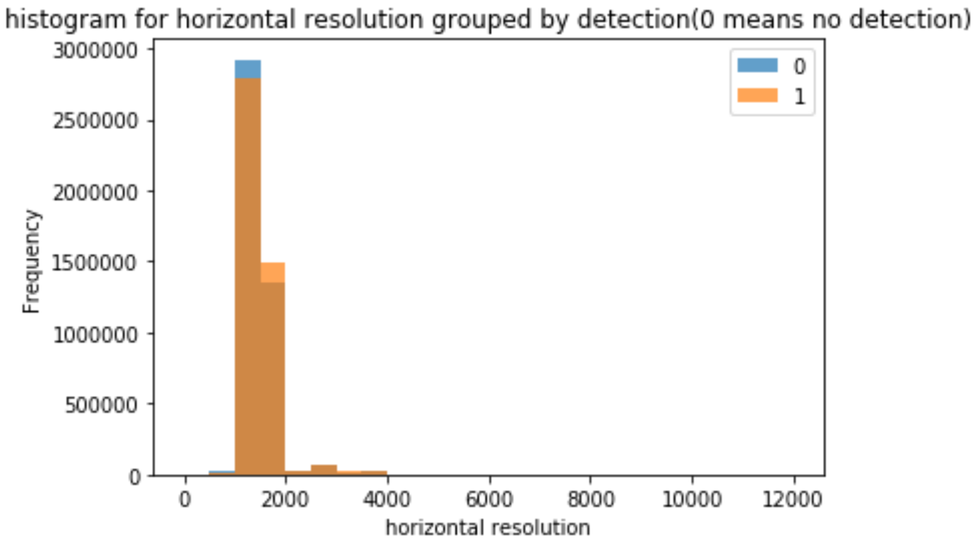
2179

In [13]:

```
resolution.groupby('affected')['hresolution'].plot(kind = 'hist',alpha=0.7,bins=np.arange(0,max(
resolution.hresolution),500))
plt.legend()
plt.title('histogram for horizontal resolution grouped by detection(0 means no detection)')
plt.xlabel('horizontal resolution')
```

Out[13]:

```
Text(0.5, 0, 'horizontal resolution')
```

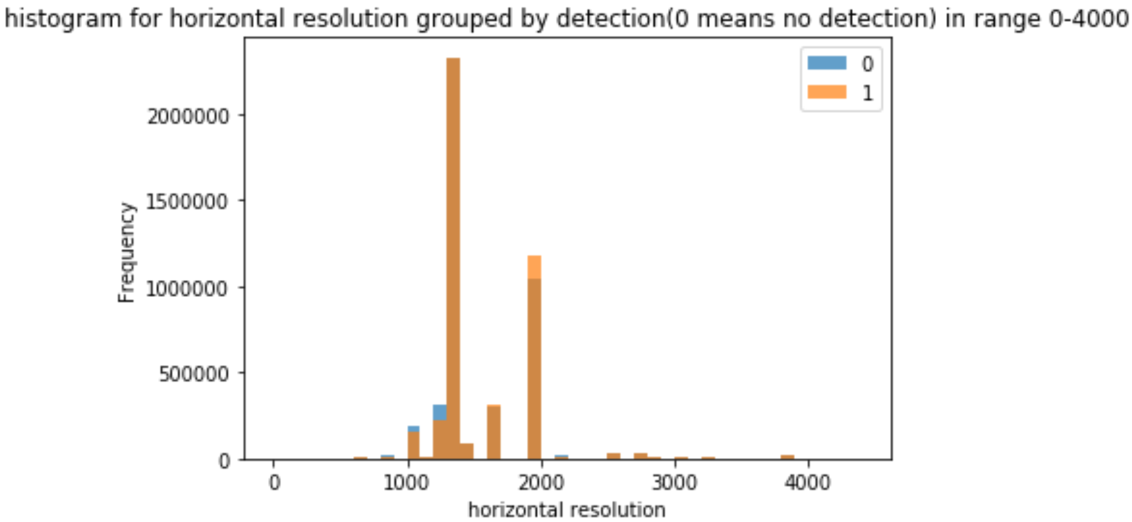


In [16]:

```
resolution.groupby('affected')['hresolution'].plot(kind = 'hist',alpha=0.7,bins=np.arange(0,4500
,100))
plt.legend()
plt.title('histogram for horizontal resolution grouped by detection(0 means no detection) in ran
ge 0-4000')
plt.xlabel('horizontal resolution')
```

Out[16]:

```
Text(0.5, 0, 'horizontal resolution')
```

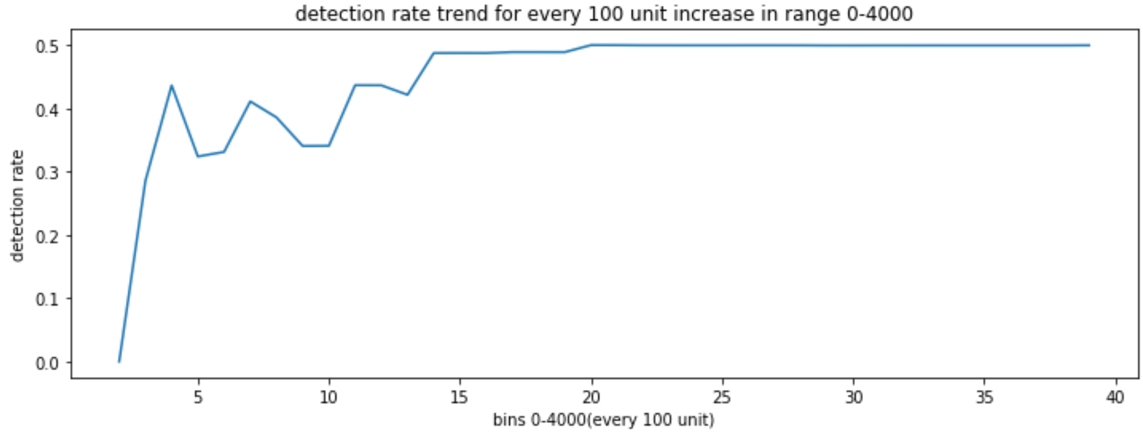


In [15]:

```
h_means = []
for i in range(0,4000,100):
    h_means.append(resolution[resolution.hresolution < i].affected.mean())
pd.Series(h_means).plot(figsize=(12,4))
plt.xlabel('bins 0-4000(every 100 unit)')
plt.ylabel('detection rate')
plt.title('detection rate trend for every 100 unit increase in range 0-4000')
```

Out[15]:

```
Text(0.5, 1.0, 'detection rate trend for every 100 unit increase in range 0-4000')
```

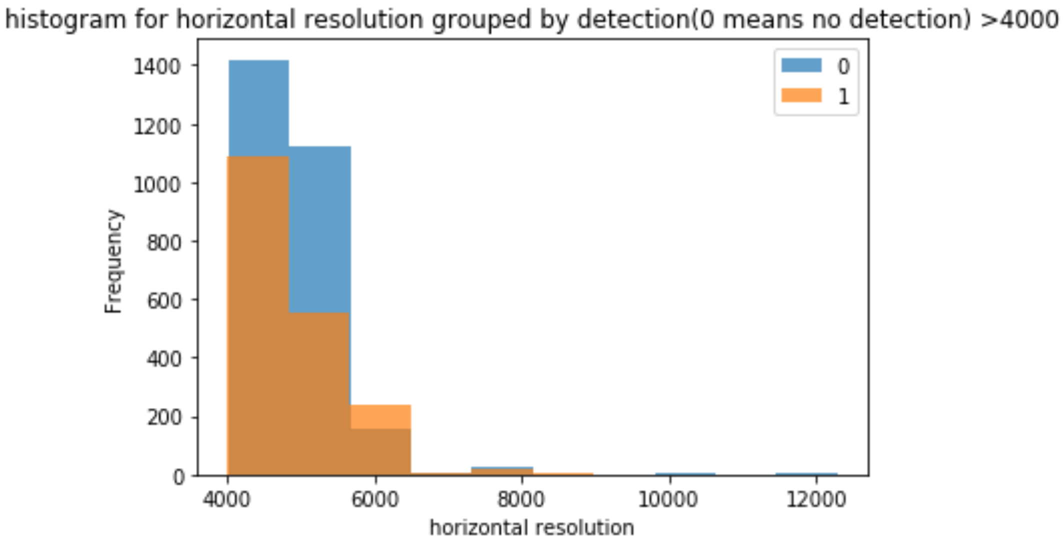


In [17]:

```
high_res = resolution[resolution.hresolution > 4000]
high_res.groupby('affected').hresolution.plot(kind='hist',alpha=0.7)
plt.legend()
plt.title('histogram for horizontal resolution grouped by detection(0 means no detection) >4000'
)
plt.xlabel('horizontal resolution')
```

Out[17]:

```
Text(0.5, 0, 'horizontal resolution')
```

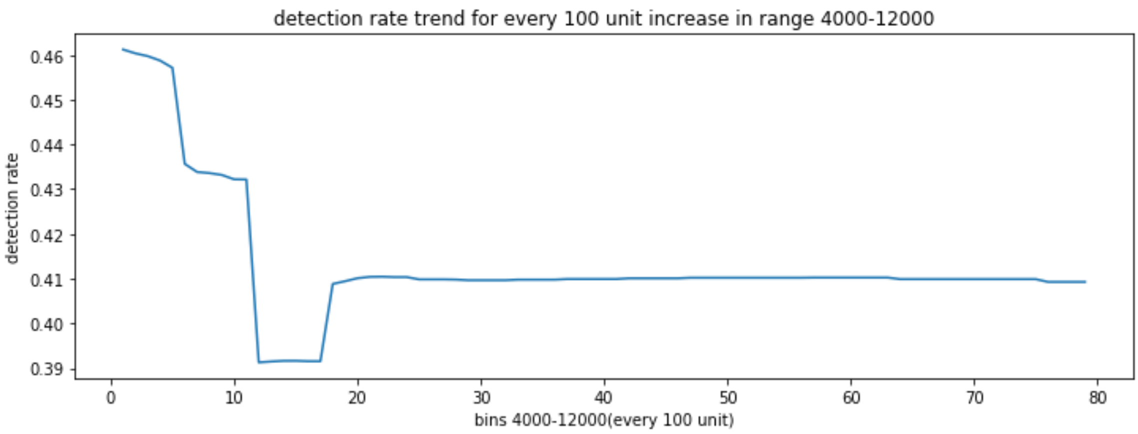


In [18]:

```
highres_means = []
for i in range(4000,12000,100):
    highres_means.append(high_res[high_res.hresolution < i].affected.mean())
pd.Series(highres_means).plot(figsize=(12,4))
plt.xlabel('bins 4000-12000(every 100 unit)')
plt.ylabel('detection rate')
plt.title('detection rate trend for every 100 unit increase in range 4000-12000')
```

Out[18]:

```
Text(0.5, 1.0, 'detection rate trend for every 100 unit increase in range 4000-120
00')
```

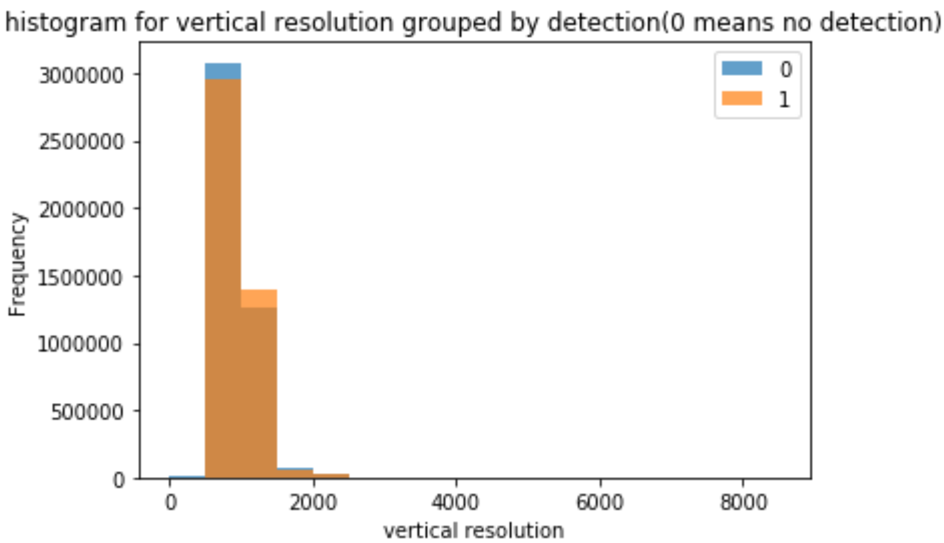


In [19]:

```
resolution.groupby('affected')['vresolution'].plot(kind = 'hist',alpha=0.7,bins=np.arange(0,max(
resolution.vresolution),500))
plt.legend()
plt.title('histogram for vertical resolution grouped by detection(0 means no detection)')
plt.xlabel('vertical resolution')
```

Out[19]:

```
Text(0.5, 0, 'vertical resolution')
```

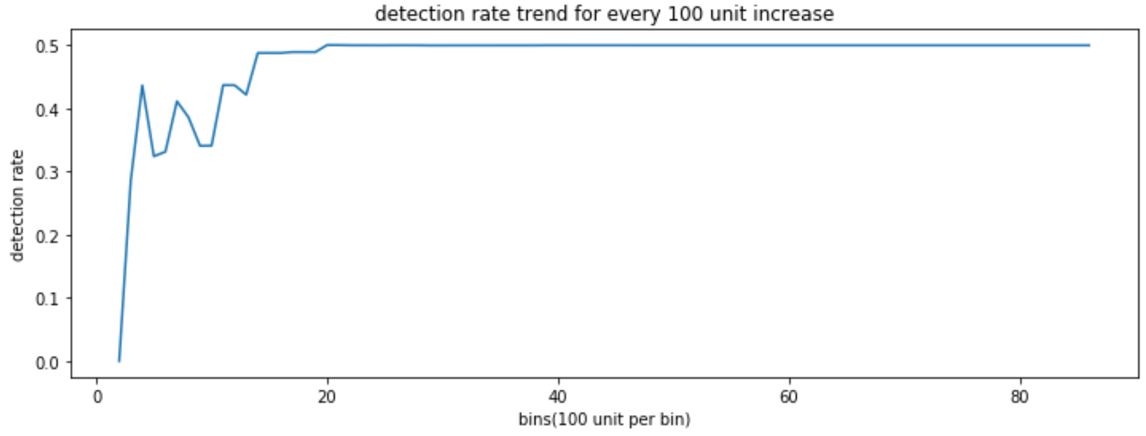


In [21]:

```
v_means = []
for i in range(0,int(max(resolution.vresolution)),100):
    v_means.append(resolution[resolution.hresolution < i].affected.mean())
pd.Series(v_means).plot(figsize=(12,4))
plt.xlabel('bins(100 unit per bin)')
plt.ylabel('detection rate')
plt.title('detection rate trend for every 100 unit increase')
```

Out[21]:

```
Text(0.5, 1.0, 'detection rate trend for every 100 unit increase')
```



In [ ]:

```

```

Scenario 3 Hardware (disk,RAM,touch)

March 22, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

In [3]: COLS = [
            'HasDetections', 'Census_TotalPhysicalRAM', 'Census_IsVirtualDevice',
            'AVProductsInstalled', 'AVProductsEnabled', 'SMode',
            'Census_IsAlwaysOnAlwaysConnectedCapable', 'Census_PrimaryDiskTotalCapacity',
            'Census_IsTouchEnabled'
        ]

In [4]: df_train = pd.read_csv("data/train.csv", sep=',', engine='c', usecols=COLS)

In [5]: df_train.head().T

Out[5]:                                                0         1         2  \
        AVProductsInstalled                          1.0       1.0       1.0
        AVProductsEnabled                            1.0       1.0       1.0
        SMode                                        0.0       0.0       0.0
        Census_PrimaryDiskTotalCapacity         476940.0  476940.0  114473.0
        Census_TotalPhysicalRAM                   4096.0    4096.0    4096.0
        Census_IsVirtualDevice                       0.0       0.0       0.0
        Census_IsTouchEnabled                        0.0       0.0       0.0
        Census_IsAlwaysOnAlwaysConnectedCapable      0.0       0.0       0.0
        HasDetections                                0.0       0.0       0.0

                                                       3         4
        AVProductsInstalled                          1.0       1.0
        AVProductsEnabled                            1.0       1.0
        SMode                                        0.0       0.0
        Census_PrimaryDiskTotalCapacity         238475.0  476940.0
        Census_TotalPhysicalRAM                   4096.0    6144.0
        Census_IsVirtualDevice                       0.0       0.0
        Census_IsTouchEnabled                        0.0       0.0
        Census_IsAlwaysOnAlwaysConnectedCapable      0.0       0.0
        HasDetections                                1.0       1.0
```

In [6]:
```
df_train[['Census_TotalPhysicalRAM', 'Census_PrimaryDiskTotalCapacity',
    'Census_IsTouchEnabled'
]].describe().T
```

Out[6]:
```
                                     count          mean           std    min  \
Census_TotalPhysicalRAM          8840950.0  6.115261e+03  5.115821e+03  255.0
Census_PrimaryDiskTotalCapacity  8868467.0  3.089053e+06  4.451634e+09    0.0
Census_IsTouchEnabled            8921483.0  1.255431e-01  3.313338e-01    0.0

                                      25%       50%       75%           max
Census_TotalPhysicalRAM            4096.0    4096.0    8192.0  1.572864e+06
Census_PrimaryDiskTotalCapacity  239372.0  476940.0  953869.0  8.160437e+12
Census_IsTouchEnabled                 0.0       0.0       0.0  1.000000e+00
```

In [7]:
```
df_train[['Census_TotalPhysicalRAM'
]].describe().T
```

Out[7]:
```
                             count         mean          std    min     25%  \
Census_TotalPhysicalRAM  8840950.0  6115.260794  5115.820685  255.0  4096.0

                            50%     75%        max
Census_TotalPhysicalRAM  4096.0  8192.0  1572864.0
```

In [8]:
```
df_train[['Census_PrimaryDiskTotalCapacity'
]].describe().T
```

Out[8]:
```
                                     count          mean           std  min  \
Census_PrimaryDiskTotalCapacity  8868467.0  3.089053e+06  4.451634e+09  0.0

                                      25%       50%       75%           max
Census_PrimaryDiskTotalCapacity  239372.0  476940.0  953869.0  8.160437e+12
```

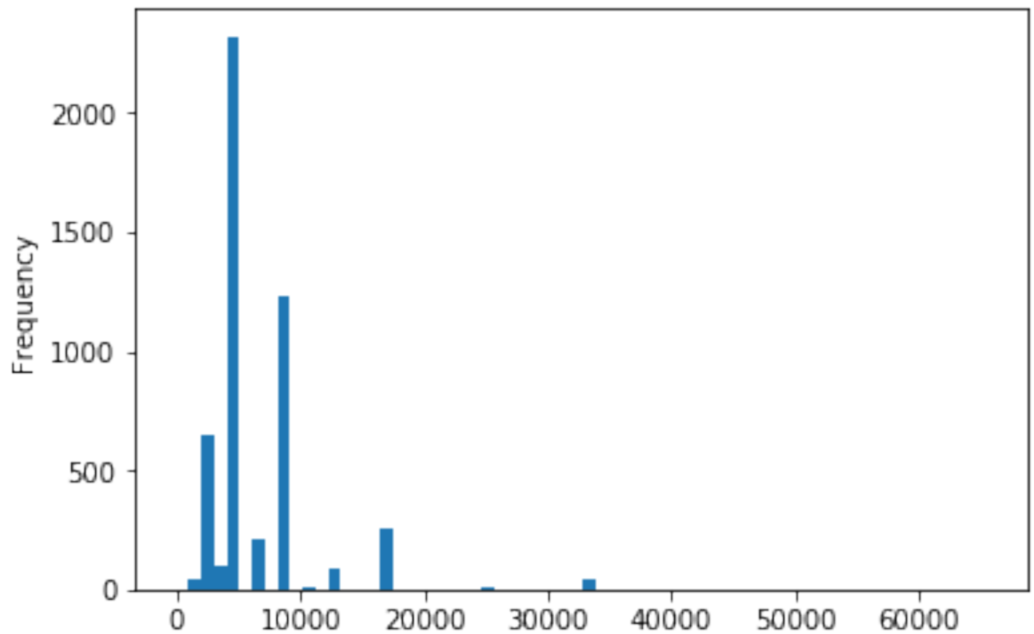
In [9]:
```
df_train[['Census_IsTouchEnabled'
]].describe().T
```

Out[9]:
```
                           count      mean       std  min  25%  50%  75%  max
Census_IsTouchEnabled  8921483.0  0.125543  0.331334  0.0  0.0  0.0  0.0  1.0
```

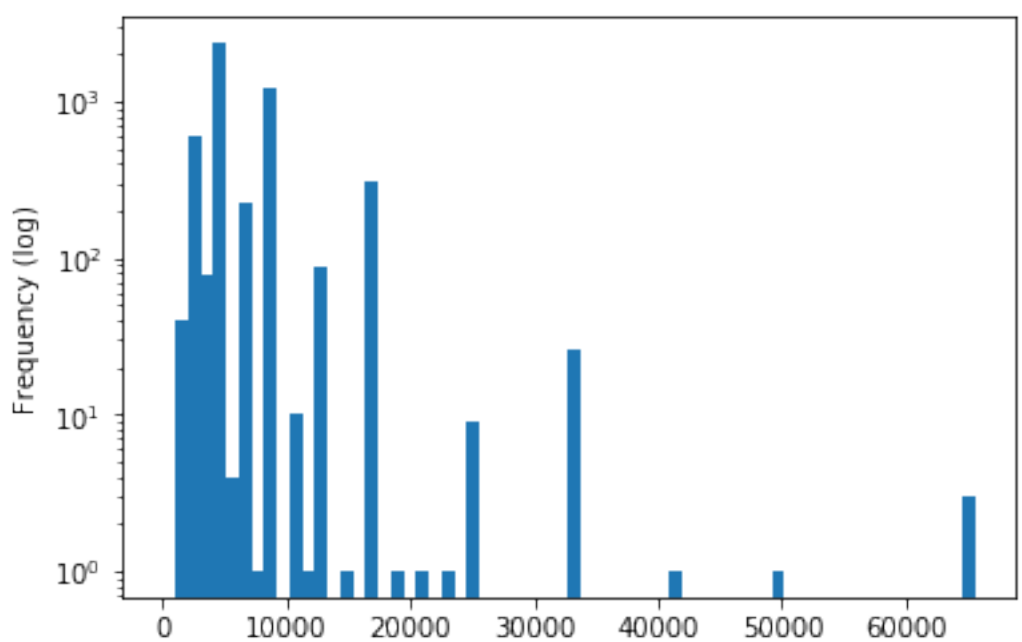
In [10]:
```
df_train[['Census_TotalPhysicalRAM', 'Census_PrimaryDiskTotalCapacity',
    'Census_IsTouchEnabled'
]].isna().mean()
```

Out[10]:
```
Census_TotalPhysicalRAM            0.009027
Census_PrimaryDiskTotalCapacity    0.005943
Census_IsTouchEnabled              0.000000
dtype: float64
```

In [11]:
```
df_train.Census_TotalPhysicalRAM.sample(5000).plot.hist(bins=np.arange(0, 65537, 1024)
```




```
In [12]: df_train.Census_TotalPhysicalRAM.sample(5000).plot.hist(bins=np.arange(0, 65537, 1024)
         plt.yscale('log')
         plt.ylabel('Frequency (log)')
         plt.show()
```

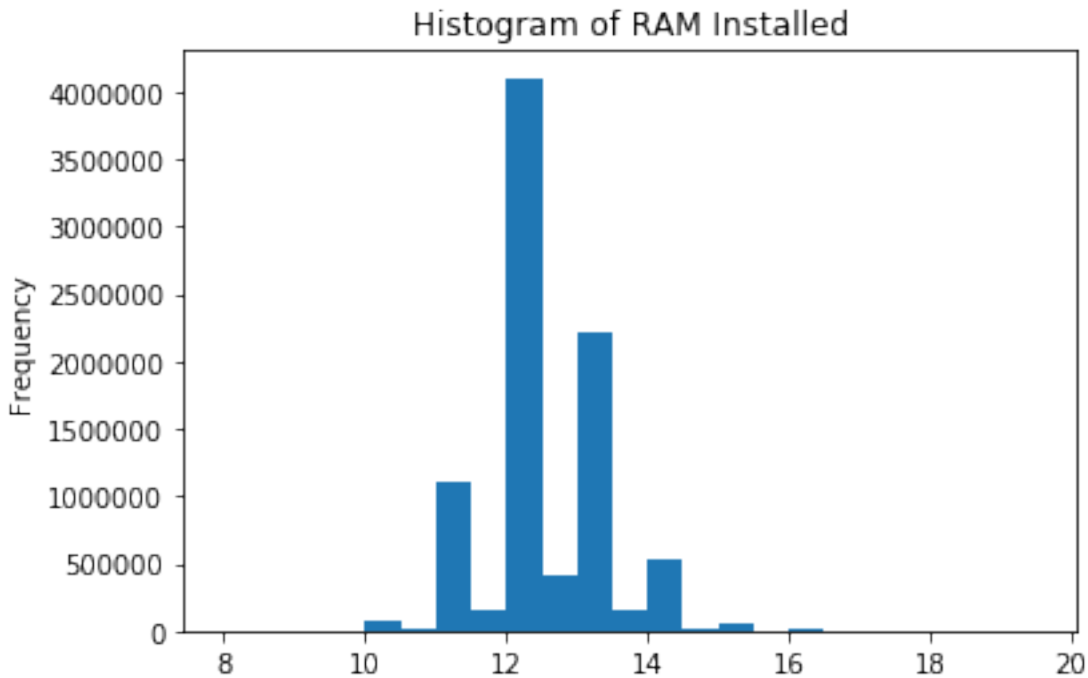


In [13]:
```
df_train.Census_TotalPhysicalRAM.sample(5000).value_counts()
```

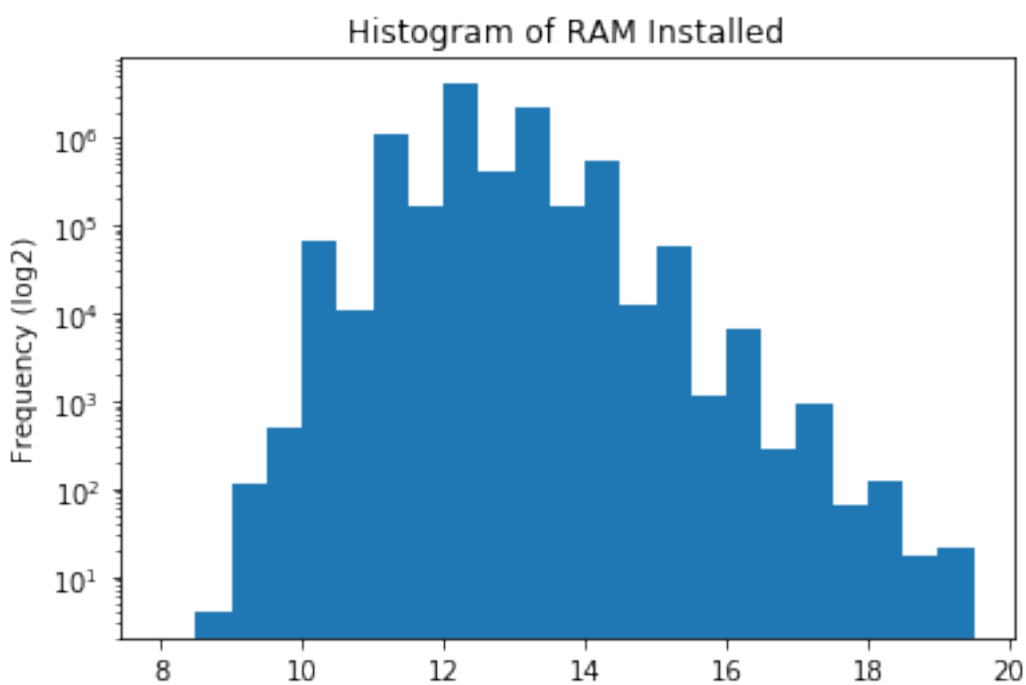
Out[13]:
```
4096.0      2261
8192.0      1270
2048.0       643
16384.0      287
6144.0       187
12288.0      103
3072.0        82
32768.0       39
1024.0        28
10240.0        8
24576.0        6
20480.0        4
2560.0         3
1536.0         3
5120.0         3
4095.0         2
14336.0        1
1399.0         1
2047.0         1
3485.0         1
1023.0         1
16367.0        1
3579.0         1
1802.0         1
8096.0         1
3582.0         1
1280.0         1
16303.0        1
131072.0       1
18468.0        1
8208.0         1
65536.0        1
7168.0         1
3007.0         1
16127.0        1
Name: Census_TotalPhysicalRAM, dtype: int64
```

In [14]:
```
s_RAM = df_train.Census_TotalPhysicalRAM.values
log_s_RAM = np.log2(s_RAM)
```

In [15]:
```
plt.hist(log_s_RAM, bins=np.arange(8, 20, 0.5))
plt.title('Histogram of RAM Installed')
plt.ylabel('Frequency')
plt.show()
```



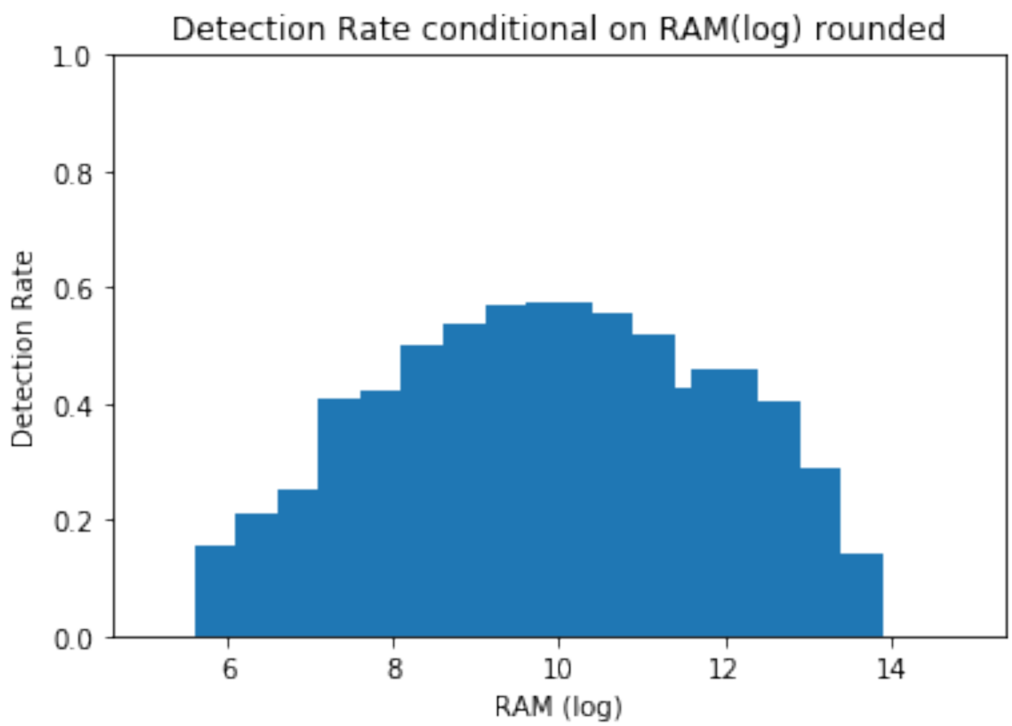

```
In [16]: plt.hist(log_s_RAM, bins=np.arange(8, 20, 0.5))
         plt.title('Histogram of RAM Installed')
         plt.yscale('log')
         plt.ylabel('Frequency (log2)')
         plt.show()
```



```
In [17]: df_train['RAM_rounded_log'] = np.round(np.log(df_train.Census_TotalPhysicalRAM.values

         rates_by_RAM = df_train.groupby('RAM_rounded_log')['HasDetections'].agg('mean')
         rates_by_RAM
```

```
Out[17]: RAM_rounded_log
         5.5     0.000000
         6.0     0.153153
         6.5     0.212766
         7.0     0.250654
         7.5     0.406198
         8.0     0.423870
         8.5     0.499268
         9.0     0.538428
         9.5     0.568571
         10.0    0.573308
         10.5    0.555264
         11.0    0.520483
         11.5    0.425000
         12.0    0.456947
         12.5    0.406015
         13.0    0.289474
         13.5    0.142857
         14.0    0.000000
         14.5    0.000000
         Name: HasDetections, dtype: float64
```

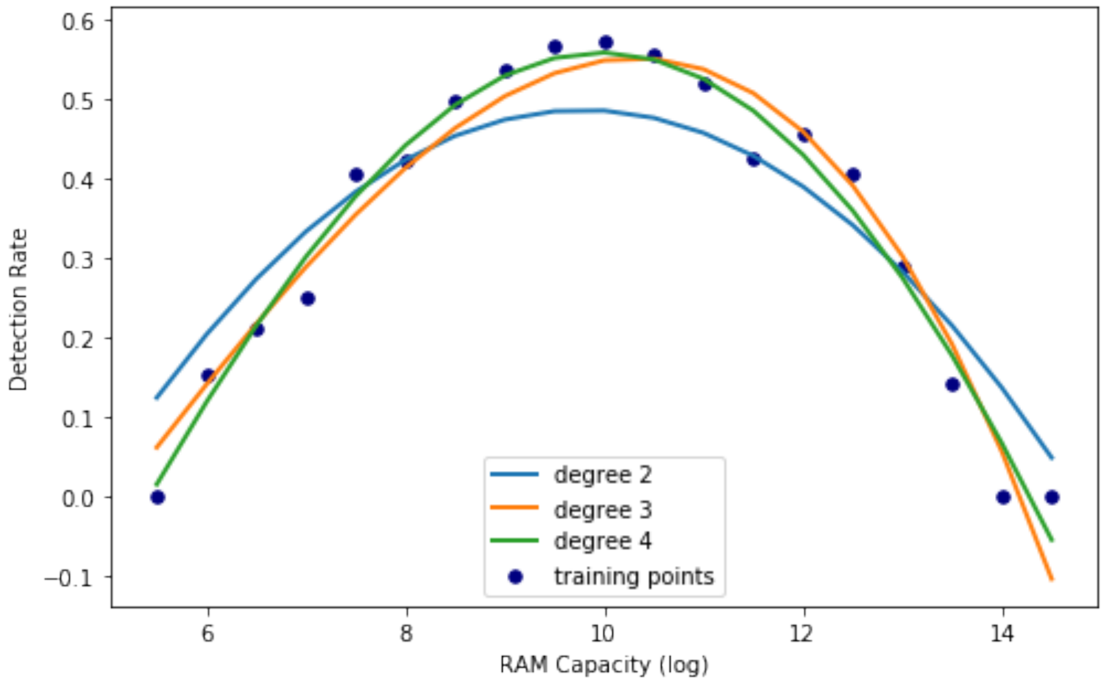
```
In [18]: plt.bar(rates_by_RAM.index, rates_by_RAM.values)
         plt.title('Detection Rate conditional on RAM(log) rounded')
         plt.ylabel('Detection Rate')
         plt.xlabel('RAM (log)')
         plt.ylim((0,1))
         plt.show()
```



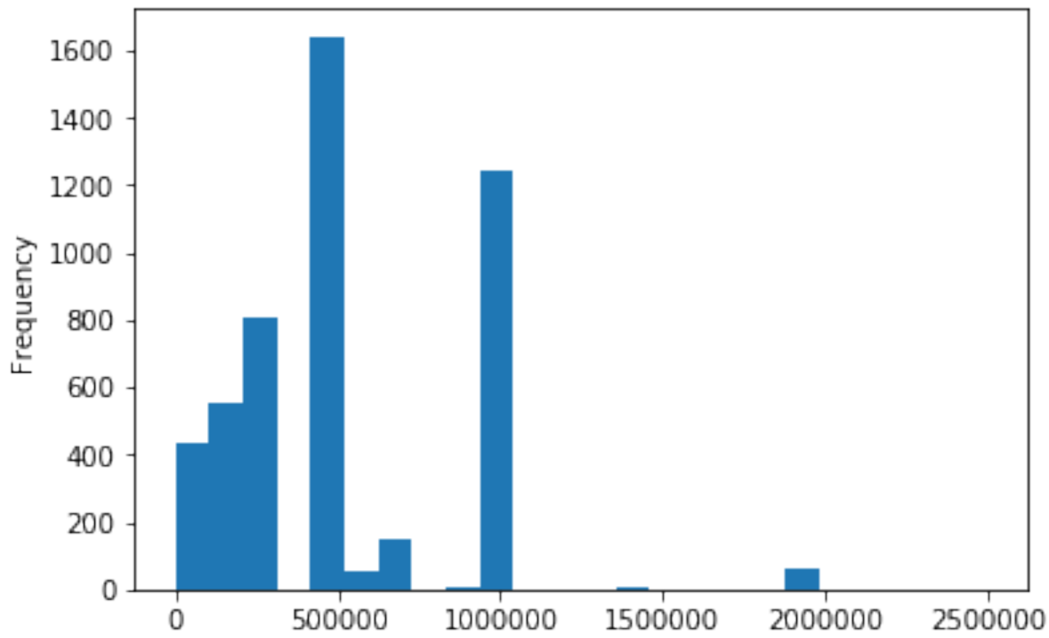

```
In [19]: from sklearn.linear_model import Ridge
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.pipeline import make_pipeline

In [20]: x = rates_by_RAM.index
         x_plot = rates_by_RAM.index
         X = x[:, np.newaxis]
         X_plot = x_plot[:, np.newaxis]
         y = rates_by_RAM.values

         lw = 2
         fig, axes = plt.subplots(figsize=(8, 5))
         plt.scatter(x, y, color='navy', s=30, marker='o', label="training points")

         for count, degree in enumerate([2, 3, 4]):
             model = make_pipeline(PolynomialFeatures(degree, include_bias=False), Ridge())
             model.fit(X, y)
             print(model.named_steps['ridge'].coef_, model.named_steps['ridge'].intercept_)
             y_plot = model.predict(X_plot)
             plt.plot(x_plot, y_plot, linewidth=lw,
                       label="degree %d" % degree)

         plt.legend(loc='lower center')
         plt.ylabel('Detection Rate')
```

```
        plt.xlabel('RAM Capacity (log)')
        plt.show()
```

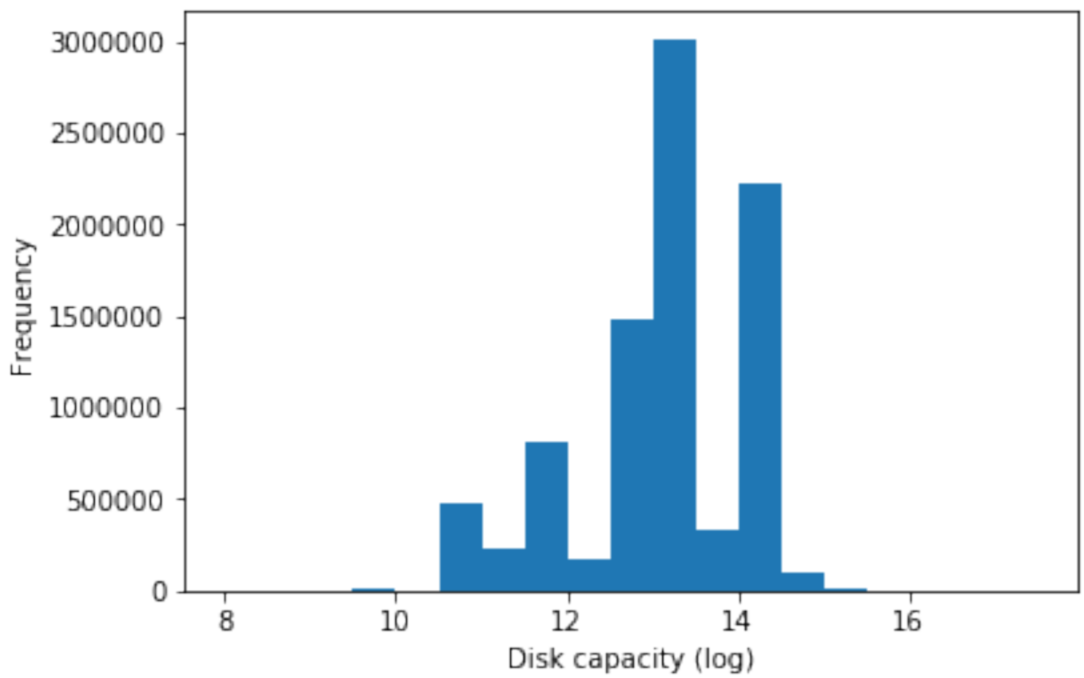
```
[ 0.38631895 -0.01973611] -1.4035445346918967
[ 0.0306556   0.02699832 -0.0018393 ] -0.6177166509035579
[ 0.00794238  0.05663592 -0.005564    0.00013169] -0.9367193282532358
```



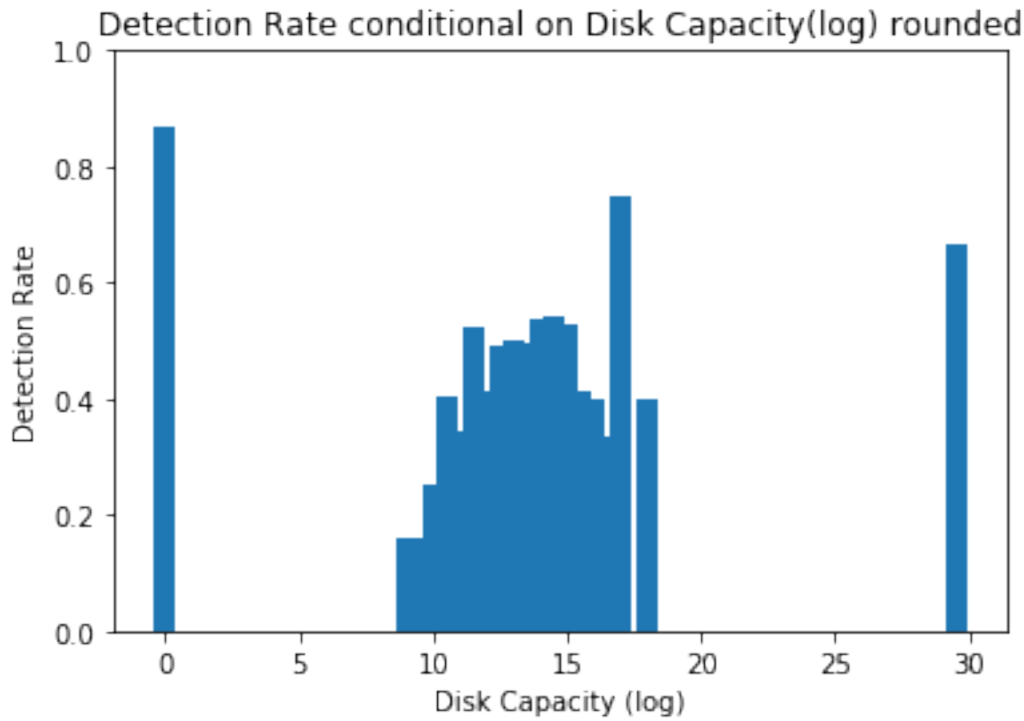

```
In [21]: np.corrcoef((x-np.mean(x))**2, y)

Out[21]: array([[ 1.        , -0.98112967],
                [-0.98112967,  1.        ]])

In [22]: df_train.Census_PrimaryDiskTotalCapacity.sample(5000).plot.hist(bins=np.linspace(0, 25

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbdbb5fb0b8>
```




```
In [23]: plt.hist(np.round(np.log(df_train.Census_PrimaryDiskTotalCapacity.dropna().values + 1)
                  bins=np.arange(8, 18, 0.5));
         plt.ylabel('Frequency')
         plt.xlabel('Disk capacity (log)')

Out[23]: Text(0.5,0,'Disk capacity (log)')
```

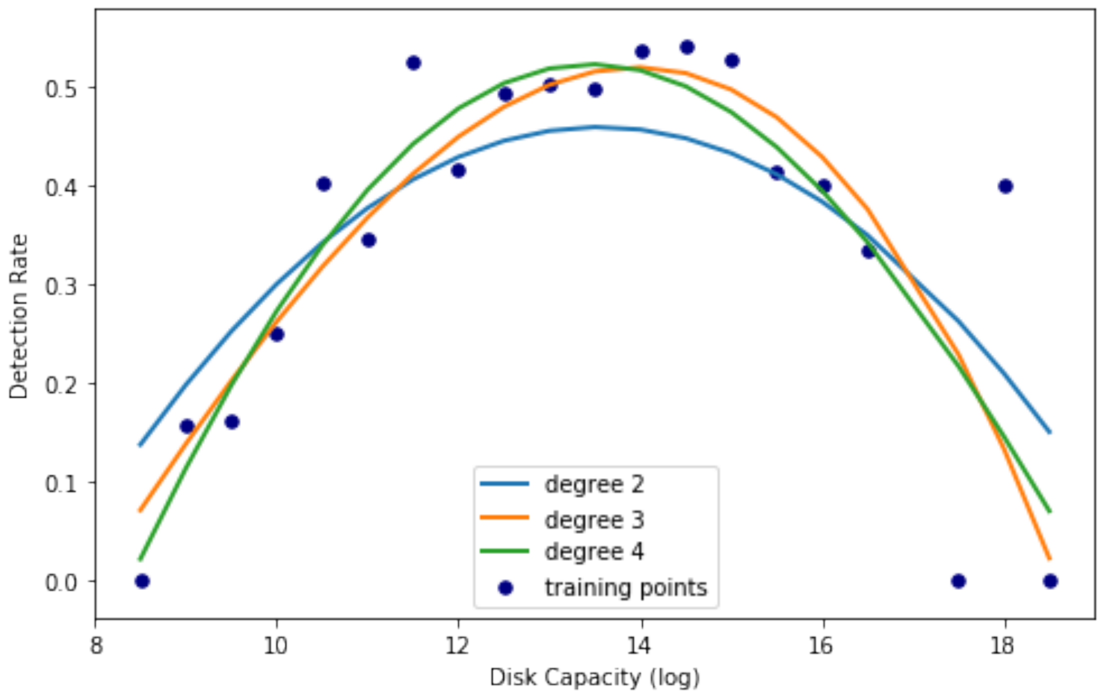


```
In [24]: df_train['disk_rounded_log'] = np.round(np.log(df_train.Census_PrimaryDiskTotalCapaci

         rates_by_disk = df_train.groupby('disk_rounded_log')['HasDetections'].agg('mean')
         rates_by_disk

Out[24]: disk_rounded_log
         0.0     0.866667
         8.5     0.000000
         9.0     0.157895
         9.5     0.161569
         10.0    0.250589
         10.5    0.403275
         11.0    0.344633
         11.5    0.524520
         12.0    0.415220
         12.5    0.493409
         13.0    0.501470
         13.5    0.497369
         14.0    0.536157
         14.5    0.541539
         15.0    0.527977
         15.5    0.414343
         16.0    0.400000
         16.5    0.333333
         17.0    0.750000
         17.5    0.000000
         18.0    0.400000
         18.5    0.000000
         29.5    0.666667
         Name: HasDetections, dtype: float64

In [25]: plt.bar(rates_by_disk.index, rates_by_disk.values)
         plt.title('Detection Rate conditional on Disk Capacity(log) rounded')
         plt.ylabel('Detection Rate')
         plt.xlabel('Disk Capacity (log)')
         plt.ylim((0,1))
         plt.show()
```

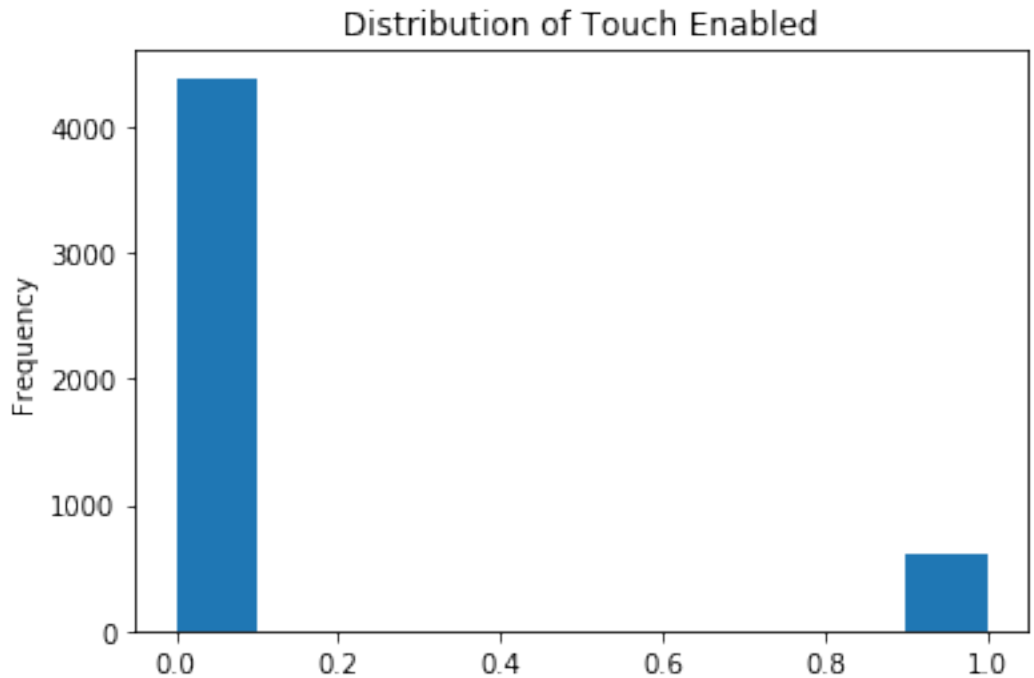



```
In [26]: # Dropped outliers rates
         rates_by_disk_dropped = rates_by_disk[lambda x: x < 0.6]
         x = rates_by_disk_dropped.index
         x_plot = rates_by_disk_dropped.index
         X = x[:, np.newaxis]
         X_plot = x_plot[:, np.newaxis]
         y = rates_by_disk_dropped.values

         lw = 2
         fig, axes = plt.subplots(figsize=(8, 5))
         plt.scatter(x, y, color='navy', s=30, marker='o', label="training points")

         for count, degree in enumerate([2, 3, 4]):
             model = make_pipeline(PolynomialFeatures(degree, include_bias=False), Ridge())
             model.fit(X, y)
             print(model.named_steps['ridge'].coef_, model.named_steps['ridge'].intercept_)
             y_plot = model.predict(X_plot)
             plt.plot(x_plot, y_plot, linewidth=lw,
                      label="degree %d" % degree)

         plt.legend(loc='lower center')
         plt.ylabel('Detection Rate')
         plt.xlabel('Disk Capacity (log)')
         plt.show()
```

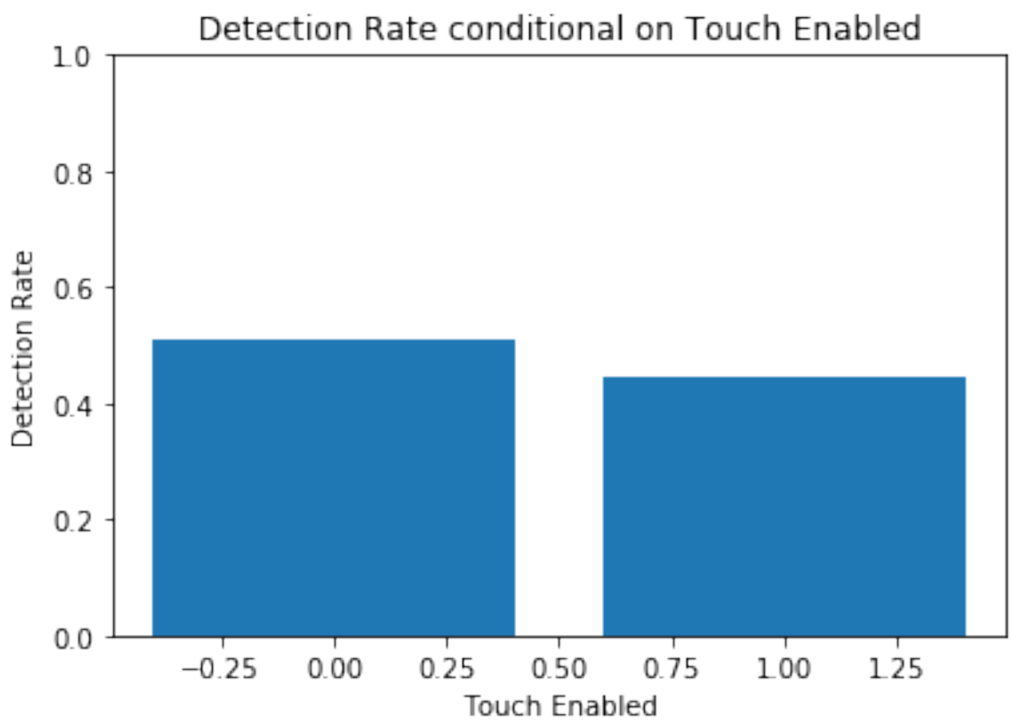
```
[ 0.34175864 -0.01260981] -1.856566740104062
[ 0.02392179  0.01831371 -0.0009152 ] -0.8935253358371735
[ 5.07182598e-03  4.64093088e-02 -3.59126546e-03  7.14871533e-05] -1.542470336851569
```




```
In [27]: np.corrcoef((x-np.mean(x))**2, y)

Out[27]: array([[ 1.        , -0.86876658],
                [-0.86876658,  1.        ]])

In [28]: np.corrcoef((x-np.mean(x))**2, y)

Out[28]: array([[ 1.        , -0.86876658],
                [-0.86876658,  1.        ]])

In [29]: df_train.Census_IsTouchEnabled.values.mean()

Out[29]: 0.12554314120197282

In [30]: df_train.Census_IsTouchEnabled.sample(5000).plot.hist()
         plt.title('Distribution of Touch Enabled')

Out[30]: Text(0.5,1,'Distribution of Touch Enabled')
```




```
In [31]: rates_by_touch = df_train.groupby('Census_IsTouchEnabled')['HasDetections'].mean()

In [32]: plt.bar(rates_by_touch.index, rates_by_touch.values)
         plt.title('Detection Rate conditional on Touch Enabled')
         plt.ylabel('Detection Rate')
         plt.xlabel('Touch Enabled')
         plt.ylim((0,1))
         plt.show()
```



In [33]:
```
df_train.columns
```

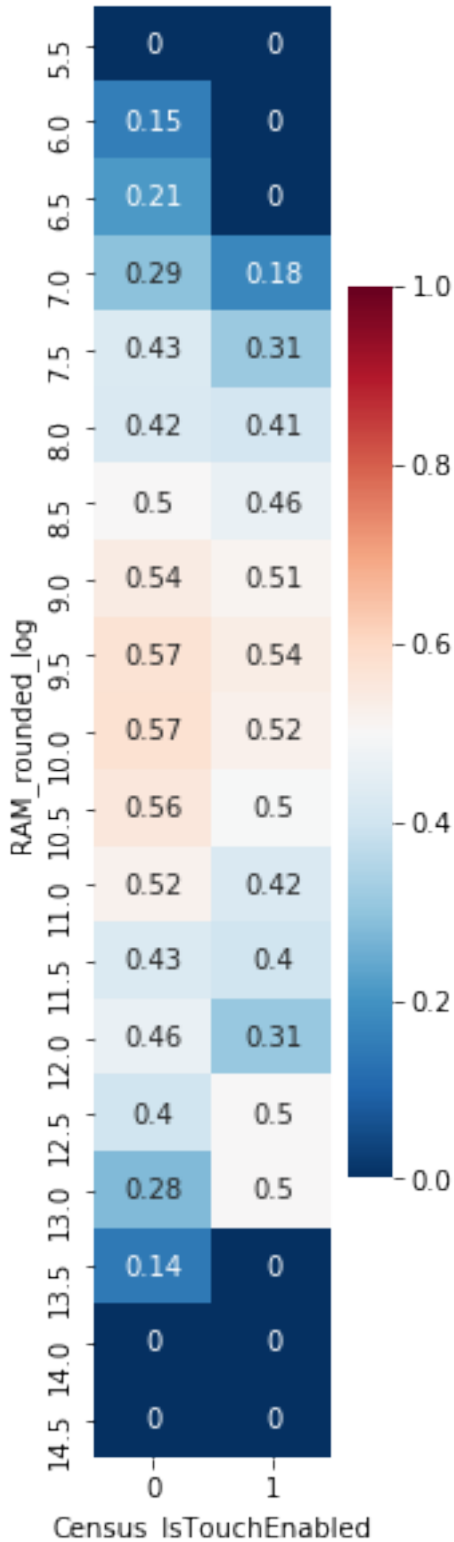
Out[33]:
```
Index(['AVProductsInstalled', 'AVProductsEnabled', 'SMode',
       'Census_PrimaryDiskTotalCapacity', 'Census_TotalPhysicalRAM',
       'Census_IsVirtualDevice', 'Census_IsTouchEnabled',
       'Census_IsAlwaysOnAlwaysConnectedCapable', 'HasDetections',
       'RAM_rounded_log', 'disk_rounded_log'],
      dtype='object')
```

In [34]:
```
pivot_ram_touch = pd.pivot_table(df_train, values='HasDetections',
                                 index='RAM_rounded_log', columns='Census_IsTouchEnabl
pivot_ram_touch
```

Out[34]:
```
Census_IsTouchEnabled         0         1
RAM_rounded_log
5.5                    0.000000       NaN
6.0                    0.154545  0.000000
6.5                    0.212766       NaN
7.0                    0.293748  0.175191
7.5                    0.428302  0.314758
8.0                    0.424104  0.406583
8.5                    0.503717  0.462441
9.0                    0.542506  0.512505
9.5                    0.572849  0.536269
```

```
10.0                    0.574710  0.524017
10.5                    0.558014  0.497965
11.0                    0.522884  0.424084
11.5                    0.426667  0.400000
12.0                    0.461229  0.310345
12.5                    0.401575  0.500000
13.0                    0.277778  0.500000
13.5                    0.142857       NaN
14.0                    0.000000       NaN
14.5                    0.000000       NaN
```

In [35]:
```
fig, ax = plt.subplots(figsize=(2,10))
ax = sns.heatmap(pivot_ram_touch.fillna(0), vmin=0, vmax=1, annot=True, cmap="RdBu_r")
```



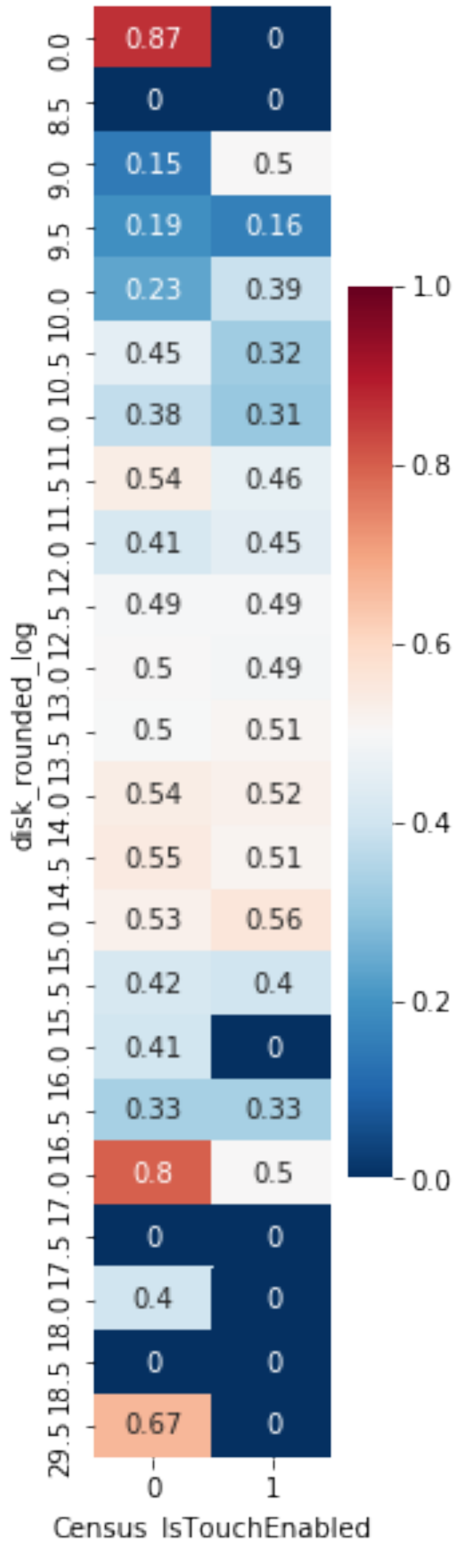

```
In [36]: pivot_disk_touch = pd.pivot_table(df_train, values='HasDetections',
```

```
                                    index='disk_rounded_log', columns='Census_IsTouchEnal
pivot_disk_touch
```

Out[36]:

Census_IsTouchEnabled	0	1
disk_rounded_log		
0.0	0.866667	NaN
8.5	NaN	0.000000
9.0	0.145455	0.500000
9.5	0.193220	0.156540
10.0	0.234762	0.389744
10.5	0.450299	0.321638
11.0	0.376927	0.310612
11.5	0.538833	0.457954
12.0	0.414297	0.446674
12.5	0.493295	0.494415
13.0	0.502442	0.491125
13.5	0.496512	0.508062
14.0	0.537565	0.523504
14.5	0.545435	0.512731
15.0	0.526324	0.558824
15.5	0.415254	0.400000
16.0	0.405063	0.000000
16.5	0.333333	0.333333
17.0	0.800000	0.500000
17.5	0.000000	0.000000
18.0	0.400000	NaN
18.5	0.000000	NaN
29.5	0.666667	NaN

In [37]:
```
fig, ax = plt.subplots(figsize=(2,10))
ax = sns.heatmap(pivot_disk_touch.fillna(0), vmin=0, vmax=1, annot=True, cmap="RdBu_r'
```



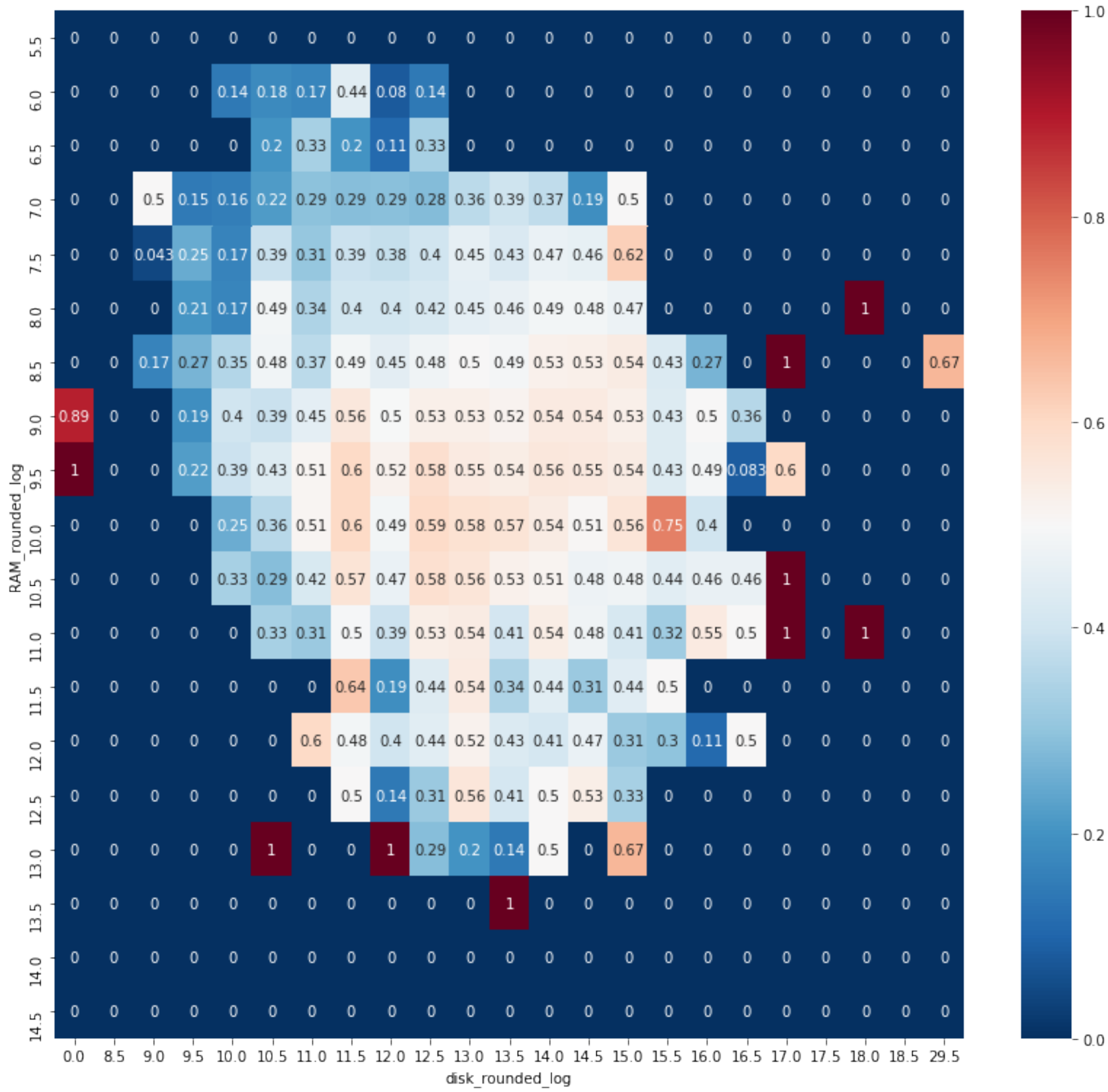

```
In [38]: import seaborn as sns
```

```
In [39]: pivot_ram_disk = pd.pivot_table(df_train, values='HasDetections',
                                         index='RAM_rounded_log', columns='disk_rounded_log', a
         pivot_ram_disk
```

```
Out[39]: disk_rounded_log       0.0  8.5       9.0       9.5      10.0      10.5  \
         RAM_rounded_log
         5.5                    NaN  NaN       NaN       NaN       NaN       NaN
         6.0                    NaN  NaN       NaN       NaN  0.142857  0.181818
         6.5                    NaN  NaN       NaN  0.000000  0.000000  0.200000
         7.0                    NaN  NaN  0.500000  0.148038  0.155280  0.215172
         7.5               0.000000  NaN  0.043478  0.248120  0.166863  0.390555
         8.0                    NaN  NaN  0.000000  0.205882  0.170984  0.487533
         8.5                    NaN  0.0  0.166667  0.267380  0.351796  0.482835
         9.0               0.888889  NaN  0.000000  0.188679  0.401254  0.388341
         9.5               1.000000  NaN  0.000000  0.222222  0.393443  0.431280
         10.0                   NaN  NaN       NaN       NaN  0.250000  0.360000
         10.5                   NaN  NaN       NaN  0.000000  0.333333  0.285714
         11.0                   NaN  NaN       NaN       NaN  0.000000  0.333333
         11.5                   NaN  NaN       NaN       NaN       NaN  0.000000
         12.0                   NaN  NaN       NaN       NaN       NaN       NaN
         12.5                   NaN  NaN       NaN       NaN       NaN       NaN
         13.0                   NaN  NaN       NaN       NaN       NaN  1.000000
         13.5                   NaN  NaN       NaN       NaN       NaN       NaN
         14.0                   NaN  NaN       NaN       NaN       NaN       NaN
         14.5                   NaN  NaN       NaN       NaN       NaN       NaN

         disk_rounded_log      11.0      11.5      12.0      12.5    ...         14.5  \
         RAM_rounded_log                                             ...
         5.5               0.000000       NaN       NaN       NaN    ...          NaN
         6.0               0.171429  0.444444  0.080000  0.142857    ...          NaN
         6.5               0.333333  0.200000  0.111111  0.333333    ...          NaN
         7.0               0.294357  0.287066  0.291961  0.281456    ...     0.190476
         7.5               0.306592  0.393878  0.379253  0.398558    ...     0.459843
         8.0               0.343858  0.403815  0.404435  0.415507    ...     0.484733
         8.5               0.365886  0.485106  0.446543  0.477978    ...     0.529923
         9.0               0.452887  0.558314  0.498605  0.528351    ...     0.544667
         9.5               0.509681  0.600989  0.519713  0.583631    ...     0.552811
         10.0              0.509202  0.598493  0.487455  0.593942    ...     0.506524
         10.5              0.420455  0.568656  0.467192  0.578797    ...     0.475509
         11.0              0.314286  0.495385  0.393443  0.532625    ...     0.479915
         11.5              0.000000  0.636364  0.187500  0.436170    ...     0.314286
         12.0              0.600000  0.484848  0.400000  0.442396    ...     0.465347
         12.5              0.000000  0.500000  0.142857  0.312500    ...     0.533333
         13.0              0.000000       NaN  1.000000  0.285714    ...     0.000000
         13.5                   NaN  0.000000       NaN       NaN    ...     0.000000
         14.0                   NaN  0.000000       NaN       NaN    ...     0.000000
         14.5                   NaN       NaN       NaN       NaN    ...     0.000000
```

```
disk_rounded_log      15.0      15.5      16.0      16.5  17.0  17.5  18.0  \
RAM_rounded_log
5.5                    NaN       NaN       NaN       NaN   NaN   NaN   NaN
6.0                    NaN       NaN       NaN       NaN   NaN   NaN   NaN
6.5                    NaN       NaN       NaN       NaN   NaN   NaN   NaN
7.0               0.500000       NaN  0.000000       NaN   NaN   NaN   NaN
7.5               0.620000  0.000000  0.000000       NaN   NaN   NaN   0.0
8.0               0.470588  0.000000       NaN       NaN   NaN   NaN   1.0
8.5               0.539409  0.428571  0.266667       NaN   1.0   NaN   0.0
9.0               0.529165  0.431818  0.500000  0.357143   0.0   0.0   NaN
9.5               0.540519  0.431818  0.489362  0.083333   0.6   0.0   0.0
10.0              0.562500  0.750000  0.400000       NaN   NaN   NaN   NaN
10.5              0.479495  0.441860  0.464286  0.461538   1.0   0.0   NaN
11.0              0.414634  0.321429  0.545455  0.500000   1.0   NaN   1.0
11.5              0.437500  0.500000  0.000000  0.000000   NaN   NaN   NaN
12.0              0.312500  0.300000  0.111111  0.500000   NaN   NaN   NaN
12.5              0.333333  0.000000  0.000000  0.000000   NaN   NaN   NaN
13.0              0.666667  0.000000       NaN       NaN   NaN   NaN   NaN
13.5              0.000000       NaN       NaN       NaN   NaN   NaN   NaN
14.0                   NaN       NaN       NaN       NaN   NaN   NaN   NaN
14.5                   NaN       NaN       NaN       NaN   NaN   NaN   NaN

disk_rounded_log  18.5      29.5
RAM_rounded_log
5.5                NaN       NaN
6.0                NaN       NaN
6.5                NaN       NaN
7.0                NaN       NaN
7.5                NaN       NaN
8.0                NaN       NaN
8.5                NaN  0.666667
9.0                NaN       NaN
9.5                NaN       NaN
10.0               NaN       NaN
10.5               0.0       NaN
11.0               NaN       NaN
11.5               NaN       NaN
12.0               NaN       NaN
12.5               NaN       NaN
13.0               NaN       NaN
13.5               NaN       NaN
14.0               NaN       NaN
14.5               NaN       NaN

[19 rows x 23 columns]
```

In [40]:
```
fig, ax = plt.subplots(figsize=(14,13))
ax = sns.heatmap(pivot_ram_disk.fillna(0), vmin=0, vmax=1, annot=True, cmap="RdBu_r")
```


RAM_rounded_log
disk_rounded_log

Feature Selection

In [0]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
```

In [0]:

```
%time train = pd.read_csv("train.csv")
```

```
<string>:2: DtypeWarning: Columns (28) have mixed types. Specify dtype
option on import or set low_memory=False.

CPU times: user 1min 31s, sys: 36.3 s, total: 2min 7s
Wall time: 2min 4s
```

In [0]:

```
train.head()
```

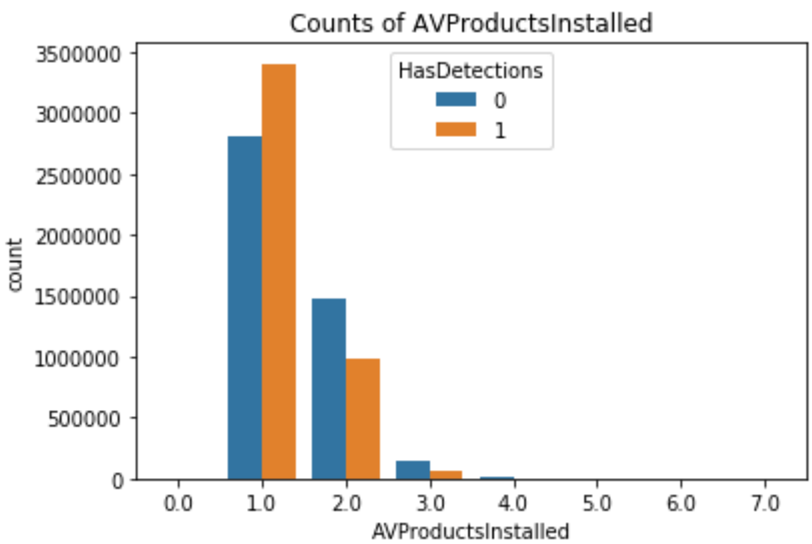
Out[352]:

	MachineIdentifier	ProductName	EngineVersion	AppVersion	AvSigVersic
0	0000028988387b115f69f31a3bf04f09	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1735
1	000007535c3f730efa9ea0b7ef1bd645	win8defender	1.1.14600.4	4.13.17134.1	1.263.48
2	000007905a28d863f6d0d597892cd692	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1341
3	00000b11598a75ea8ba1beea8459149f	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1527
4	000014a5f00daa18e76b81417eeb99fc	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1379

5 rows × 83 columns

In [0]:

```
ax = plt.axes()
sns.countplot(x='AVProductsInstalled', hue = 'HasDetections', data = train, ax=ax);
ax.set_title('Counts of AVProductsInstalled');
```

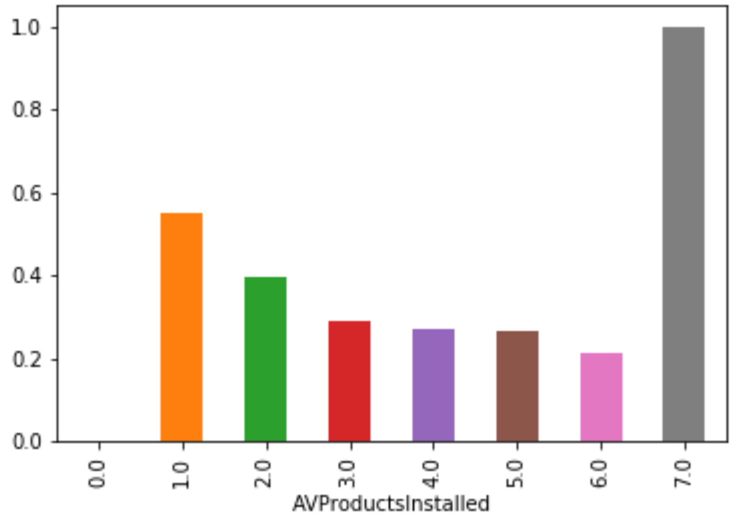


In [0]:

```
## plot the ratio of HasDetections grouped by # of AV products installed
ratio_hasdetection = train.groupby(['AVProductsInstalled']).HasDetections.apply(laml
ratio_hasdetection.plot(kind = 'bar')
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a18155a90>
```



In [0]:

```
num = train.groupby(['AVProductsInstalled']).HasDetections.count()
num
```

Out[38]:

```
AVProductsInstalled
0.0          1
1.0    6208893
2.0    2459008
3.0     208103
4.0       8757
5.0        471
6.0         28
7.0          1
Name: HasDetections, dtype: int64
```

In [0]:

```
## How to normalize the ratio so that the frequency don't take into effect?
```

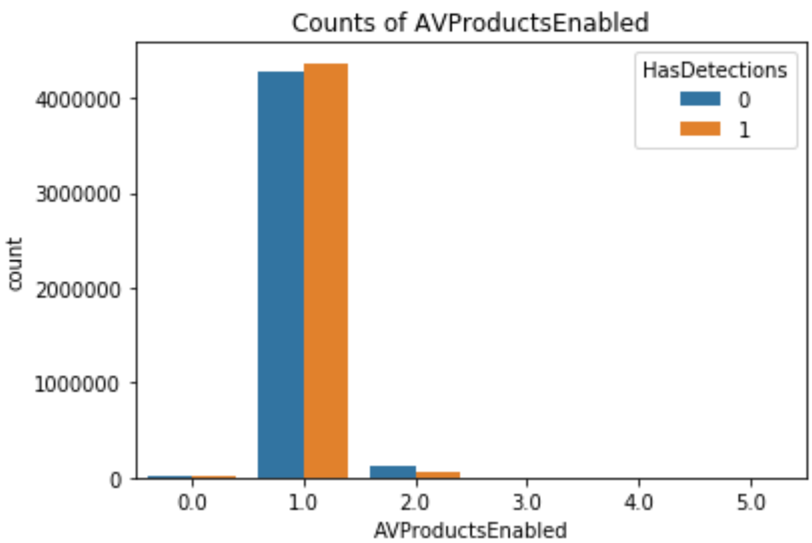
In [0]:

```
print(ratio_hasdetection)
```

```
AVProductsInstalled
0.0    0.000000
1.0    0.548581
2.0    0.396906
3.0    0.291596
4.0    0.270755
5.0    0.265393
6.0    0.214286
7.0    1.000000
Name: HasDetections, dtype: float64
```

In [0]:

```
ax = plt.axes()
sns.countplot(x='AVProductsEnabled', hue = 'HasDetections', data = train);
ax.set_title('Counts of AVProductsEnabled');
```



In [0]:

```
train['AVProductsInstalled'].unique()
```

Out[19]:

```
array([ 1.,  2.,  3.,  5., nan,  4.,  6.,  7.,  0.])
```

In [0]:

```
train[['AVProductsInstalled', 'AVProductsEnabled', 'HasDetections']].corr(method =
```

Out[40]:

	AVProductsInstalled	AVProductsEnabled	HasDetections
AVProductsInstalled	1.000000	0.238208	-0.149501
AVProductsEnabled	0.238208	1.000000	-0.042343
HasDetections	-0.149501	-0.042343	1.000000

NEW START

In [0]:

```
sampled_train = train.sample(frac = 0.001, replace=False, random_state=0)
```

In [0]:

```
categorical_features = [
        'ProductName',
        'EngineVersion',
        'AppVersion',
        'AvSigVersion',
        'Platform',
        'Processor',
        'OsVer',
        'OsPlatformSubRelease',
        'OsBuildLab',
        'SkuEdition',
        'SmartScreen',
        'Census_MDC2FormFactor',
        'Census_DeviceFamily',
        'Census_PrimaryDiskTypeName',
        'Census_ChassisTypeName',
        'Census_PowerPlatformRoleName',
        'Census_OSVersion',
        'Census_OSArchitecture',
        'Census_OSBranch',
        'Census_OSEdition',
        'Census_OSSkuName',
        'Census_OSInstallTypeName',
        'Census_OSWUAutoUpdateOptionsName',
        'Census_GenuineStateName',
        'Census_ActivationChannel',
        'Census_FlightRing',
]
```

In [0]:

```
numeric_features = [
        'IsBeta',
        'RtpStateBitfield',
        'IsSxsPassiveMode',
        'DefaultBrowsersIdentifier',
        'AVProductStatesIdentifier',
        'AVProductsInstalled',
        'AVProductsEnabled',
        'HasTpm',
        'CountryIdentifier',
        'CityIdentifier',
        'OrganizationIdentifier',
        'GeoNameIdentifier',
        'LocaleEnglishNameIdentifier',
        'OsBuild',
        'OsSuite',
        'IsProtected',
        'AutoSampleOptIn',
        'SMode',
        'IeVerIdentifier',
        'Firewall',
        'UacLuaenable',
        'Census_OEMNameIdentifier',
        'Census_OEMModelIdentifier',
        'Census_ProcessorCoreCount',
        'Census_ProcessorManufacturerIdentifier',
        'Census_ProcessorModelIdentifier',
        'Census_PrimaryDiskTotalCapacity',
        'Census_SystemVolumeTotalCapacity',
        'Census_HasOpticalDiskDrive',
        'Census_TotalPhysicalRAM',
        'Census_InternalPrimaryDiagonalDisplaySizeInInches',
        'Census_InternalPrimaryDisplayResolutionHorizontal',
        'Census_InternalPrimaryDisplayResolutionVertical',
        'Census_InternalBatteryNumberOfCharges',
        'Census_OSBuildNumber',
        'Census_OSBuildRevision',
        'Census_OSInstallLanguageIdentifier',
        'Census_OSUILocaleIdentifier',
        'Census_IsPortableOperatingSystem',
        'Census_IsFlightsDisabled',
        'Census_ThresholdOptIn',
        'Census_FirmwareManufacturerIdentifier',
        'Census_FirmwareVersionIdentifier',
        'Census_IsSecureBootEnabled',
        'Census_IsWIMBootEnabled',
        'Census_IsVirtualDevice',
        'Census_IsTouchEnabled',
        'Census_IsPenCapable',
        'Census_IsAlwaysOnAlwaysConnectedCapable',
        'Wdft_IsGamer',
        'Wdft_RegionIdentifier',
]
```

In [0]:

```
X_sampled_train = sampled_train.drop(['HasDetections'], axis = 1)
Y_sampled_train = sampled_train['HasDetections']
```

MachineIdentifier column is dropped because it is used to identify the machine which is useless.

In [0]:

```
X_sampled_train.drop(['MachineIdentifier'], axis = 1, inplace=True)
```

Drop columns with more than 30% null values

In [0]:

```
prop_nan = X_sampled_train.apply(lambda x: np.sum(x.isna())/len(x) ,axis = 0)
largely_missing_cols = prop_nan[prop_nan > 0.3].index
```

In [0]:

```
X_sampled_train.drop(largely_missing_cols, axis = 1, inplace=True)
```

Drop categorcial columns with too skewed data (one category has more than 99% appearances)

In [0]:

```
single_category_percent = X_sampled_train.apply(lambda x: np.max(x.value_counts(norm
too_skewed_cols = single_category_percent[single_category_percent > 0.99].index
```

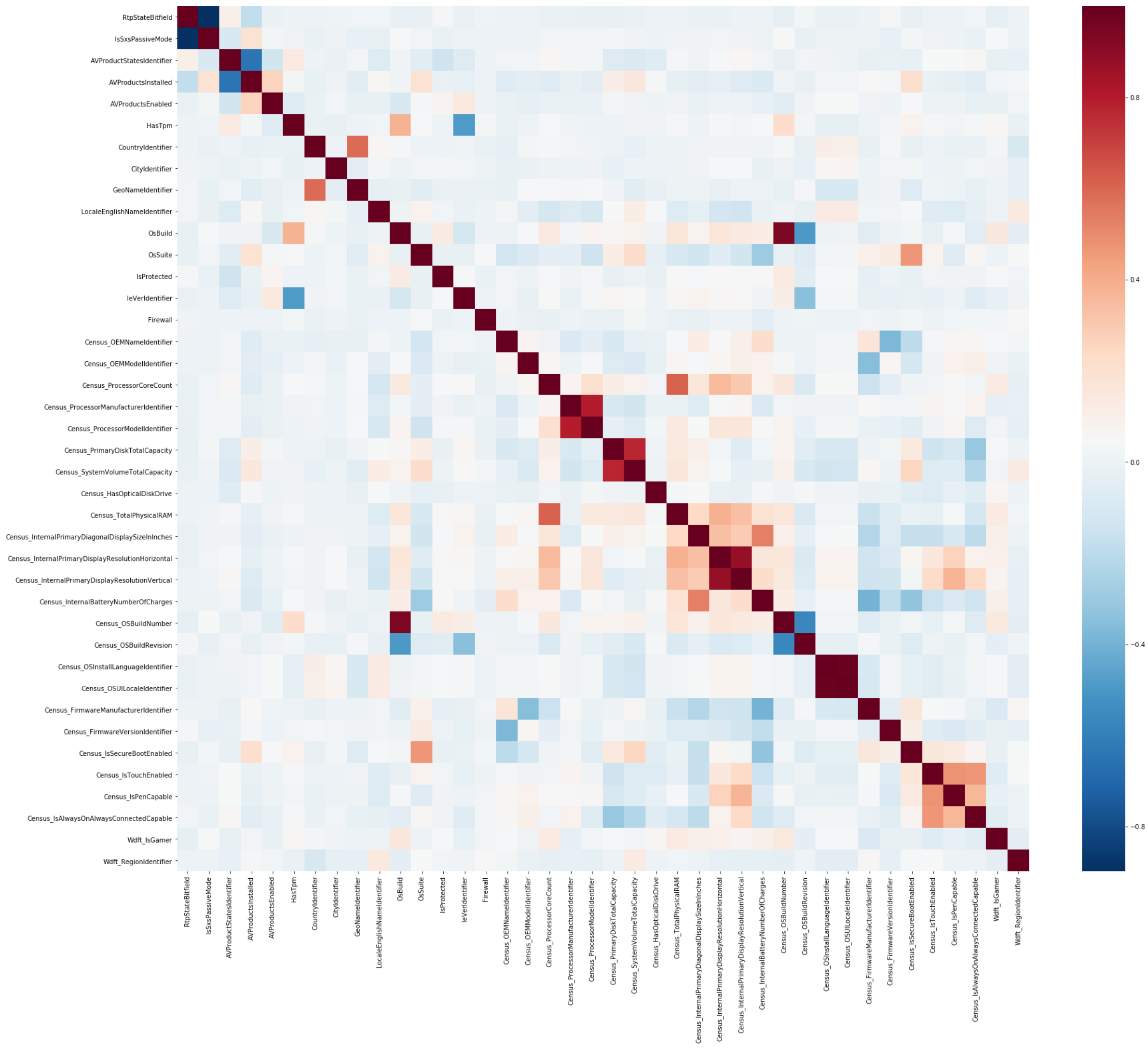
In [0]:

```
X_sampled_train.drop(too_skewed_cols, axis = 1, inplace=True)
```

Drop columns with very high linear correlations (>0.9). (Pearson) If correlation is very high, it guarantees that two columns are strongly linearly correlated.

In [0]:

```
corrs = X_sampled_train.corr(method = 'pearson')
plt.figure(figsize=(30,25))
sns.heatmap(data = corrs, cmap = 'RdBu_r');
```



Since each column has more than 90% of non-null values. Thus each column is large enough to conclude that the result from pearson corelation is statistically significant.

In [0]:

```
cols_high_corr = (corrs > 0.9) | (corrs < -0.9)
```

In [0]:

```
high_corr_cols_pair = []
```

In [0]:

```
for col1 in cols_high_corr.columns:
    for col2 in cols_high_corr.index:
        if cols_high_corr.loc[col1, col2] == True and col1 != col2:
            high_corr_cols_pair.append([col1, col2])
```

In [0]:

```
high_corr_cols_pair
```

Out[366]:

```
[['OsBuild', 'Census_OSBuildNumber'],
 ['Census_OSBuildNumber', 'OsBuild'],
 ['Census_OSInstallLanguageIdentifier', 'Census_OSUILocaleIdentifie
r'],
 ['Census_OSUILocaleIdentifier', 'Census_OSInstallLanguageIdentifie
r']]
```

Drop the one with more null values

In [0]:

```
prop_null_OsBuild = np.sum(X_sampled_train['OsBuild'].isna())/len(X_sampled_train)
prop_null_Census_OSBuildNumber = np.sum(X_sampled_train['Census_OSBuildNumber'].isna
prop_null_Census_OSInstallLanguageIdentifier = np.sum(X_sampled_train['Census_OSInst
prop_null_Census_OSUILocaleIdentifier = np.sum(X_sampled_train['Census_OSUILocaleIde
```

In [0]:

```
print('\n',prop_null_OsBuild, '\n', prop_null_Census_OSBuildNumber, '\n', prop_null_
     '\n', prop_null_Census_OSUILocaleIdentifier)
```

```
 0.0
 0.0
 0.006837798453088219
 0.0
```

In [0]:

```
strongly_dependent_cols = ['Census_OSBuildNumber', 'Census_OSInstallLanguageIdentifi
X_sampled_train.drop(strongly_dependent_cols, axis = 1, inplace=True)
```

In [0]:

```
X_sampled_train.shape
```

Out[370]:

```
(8921, 62)
```

Replace null in categorical features by making it another category 'unknown'

In [0]:

```
curr_categorical_feat = []
curr_numerical_feat = []
for col in X_sampled_train.columns:
    if col in categorical_features:
        curr_categorical_feat.append(col)
    else:
        curr_numerical_feat.append(col)
```

In [0]:

```
categorical_data = X_sampled_train[curr_categorical_feat]
numerical_data = X_sampled_train[curr_numerical_feat]
categorical_data = categorical_data.fillna('unknown')
```

Replace null in numerical features by randomly sampling from its distribution

In [0]:

```
def fill_by_dist(col):
    nonnulls = col.dropna().values
    return col.apply(lambda x: np.random.choice(a = nonnulls,size = 1)[0] if pd.isnu
numerical_data = numerical_data.apply(fill_by_dist,axis = 0)
```

In [0]:

```
X_sampled_train = categorical_data.merge(numerical_data, left_index = True, right_in
```

Run Recursive Feature Election (Backward Elimination for Logistic Regression)

In [0]:

```
X_sampled_train.join(Y_sampled_train)
```

Out[375]:

	ProductName	EngineVersion	AppVersion	AvSigVersion	Platform	Processor	(
184619	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1688.0	windows10	x64	1(
3830331	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1129.0	windows10	x64	1(
1581610	win8defender	1.1.14901.4	4.16.17656.18052	1.269.1925.0	windows10	x64	1(
3750525	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1571.0	windows10	x64	1(
3305038	win8defender	1.1.15200.1	4.18.1807.18075	1.275.938.0	windows10	x86	1(
6319876	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1360.0	windows10	x64	1(
5834703	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1086.0	windows10	x86	1(
5263516	win8defender	1.1.15200.1	4.18.1807.18075	1.275.571.0	windows10	x64	1(
7327446	win8defender	1.1.15100.1	4.18.1807.18075	1.273.894.0	windows10	x64	1(
8130082	win8defender	1.1.15200.1	4.18.1807.18075	1.275.511.0	windows10	x64	1(
275144	win8defender	1.1.15100.1	4.11.15063.1155	1.273.1073.0	windows10	x64	1(
7078611	win8defender	1.1.14104.0	4.12.16299.15	1.251.42.0	windows10	x64	1(
5691269	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1712.0	windows10	x64	1(
112482	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1176.0	windows10	x64	1(
8570481	win8defender	1.1.15200.1	4.18.1807.18075	1.273.1826.0	windows10	x64	1(
8297587	win8defender	1.1.15100.1	4.11.15063.447	1.273.1482.0	windows10	x64	1(
1421017	win8defender	1.1.15200.1	4.18.1807.18075	1.275.376.0	windows10	x64	1(
4252755	win8defender	1.1.15000.2	4.14.17639.18041	1.271.388.0	windows10	x64	1(
1392784	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1667.0	windows10	x64	1(
1641300	win8defender	1.1.14600.4	4.13.17134.228	1.263.48.0	windows10	x64	1(
2090966	win8defender	1.1.15100.1	4.18.1807.18075	1.273.810.0	windows10	x64	1(
8486326	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1482.0	windows10	x64	1(
423370	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1140.0	windows10	x64	1(
7743831	win8defender	1.1.15100.1	4.14.17639.18041	1.273.595.0	windows10	x64	1(
4928951	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1784.0	windows10	x64	1(
1433814	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1080.0	windows10	x86	1(
5355159	win8defender	1.1.14003.0	4.9.10586.0	1.249.1361.0	windows10	x64	1(
561259	win8defender	1.1.15200.1	4.18.1807.18075	1.275.500.0	windows10	x64	1(
2985726	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1519.0	windows10	x64	1(
5168305	win8defender	1.1.14600.4	4.13.17134.1	1.263.48.0	windows10	x64	1(
...	...	...	...	...	...	...	
5206064	win8defender	1.1.15100.1	4.18.1807.18075	1.273.605.0	windows10	x64	1(
6533531	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1520.0	windows10	x64	1(
8135709	win8defender	1.1.15200.1	4.18.1807.18075	1.273.1826.0	windows10	x64	1(

	ProductName	EngineVersion	AppVersion	AvSigVersion	Platform	Processor	(
4663641	win8defender	1.1.15100.1	4.18.1807.18075	1.273.841.0	windows10	x64	1(
3654596	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1783.0	windows10	x64	1(
4437908	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1826.0	windows10	x64	1(
2730730	win8defender	1.1.14800.3	4.14.17639.18041	1.267.1675.0	windows10	x64	1(
1851463	win8defender	1.1.15200.1	4.18.1807.18075	1.275.327.0	windows10	x64	1(
5654857	win8defender	1.1.15200.1	4.13.17134.228	1.275.1527.0	windows10	x64	1(
5482182	win8defender	1.1.15200.1	4.18.1807.18075	1.275.828.0	windows10	x64	1(
7321820	win8defender	1.1.15200.1	4.18.1807.18075	1.275.852.0	windows10	x64	1(
6596154	win8defender	1.1.15100.1	4.18.1807.18075	1.273.587.0	windows10	x64	1(
7174555	win8defender	1.1.15200.1	4.13.17134.1	1.275.485.0	windows10	x64	1(
7373797	win8defender	1.1.14901.4	4.16.17656.18052	1.269.1000.0	windows10	x64	1(
1747389	win8defender	1.1.15200.1	4.13.17134.228	1.275.1685.0	windows10	x64	1(
8102409	win8defender	1.1.15200.1	4.10.14393.0	1.275.1628.0	windows10	x64	1(
6887500	win8defender	1.1.15200.1	4.12.16299.15	1.275.1209.0	windows10	x64	1(
1921969	win8defender	1.1.15200.1	4.18.1807.18075	1.275.327.0	windows10	x86	1(
982765	win8defender	1.1.15200.1	4.18.1807.18075	1.275.819.0	windows10	x64	1(
4433234	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1261.0	windows10	x64	1(
4619928	win8defender	1.1.14500.5	4.9.10586.589	1.261.232.0	windows10	x86	1(
8303135	win8defender	1.1.15100.1	4.18.1807.18075	1.273.1440.0	windows10	x64	1(
6622845	win8defender	1.1.15100.1	4.12.17007.18022	1.273.1165.0	windows10	x64	1(
4726558	win8defender	1.1.13407.0	4.10.14393.1794	1.235.2629.0	windows10	x64	1(
7204438	win8defender	1.1.15000.2	4.18.1806.18062	1.271.1003.0	windows10	x64	1(
7873021	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1090.0	windows10	x64	1(
6623457	win8defender	1.1.15200.1	4.18.1807.18075	1.275.1598.0	windows10	x64	1(
1216281	win8defender	1.1.15200.1	4.18.1807.18075	1.275.613.0	windows10	x64	1(
3463454	win8defender	1.1.15200.1	4.18.1807.18075	1.275.472.0	windows10	x64	1(
2064355	win8defender	1.1.15100.1	4.18.1807.18075	1.273.571.0	windows10	x64	1(

8921 rows × 63 columns

In [0]:

```
(X_sampled_train.join(Y_sampled_train)).to_csv('cleaned_sampled_data.csv')
```

In [0]:

```
obj_col = X_sampled_train.dtypes[(X_sampled_train.dtypes == np.object).values].inde
```

In [0]:

```
num_col = X_sampled_train.dtypes[(X_sampled_train.dtypes != np.object).values].inde
```

In [0]:

```

```

In [0]:

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import RFECV
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

## ?????? OneHot or OrdinalEncoder ?????? if oneHot, would the number of columns be
ct = ColumnTransformer([
    ('OrdinalEncoder', OrdinalEncoder(), obj_col)],
    remainder = 'passthrough'
)

ct2 = ColumnTransformer([
    ('OrdinalEncoder', StandardScaler(), X_sampled_train.columns.tolist())],
    remainder = 'passthrough'
)

data = ct.fit_transform(X_sampled_train)
ordinalEncoded_X = pd.DataFrame(data = data, columns = obj_col.append(num_col))

## Standizing the data
data2 = ct2.fit_transform(ordinalEncoded_X)
ordinalEncoded_X = pd.DataFrame(data = data2, columns = ordinalEncoded_X.columns)

Y_sampled_train = Y_sampled_train.reset_index(drop=True)

estimator = LogisticRegression(solver = 'liblinear')
selector = RFE(estimator, 15, step = 1)
selector = selector.fit(ordinalEncoded_X, Y_sampled_train)

# FRECV to select columns using cross validation but unable to decide number of colu
selector_cv = RFECV(estimator, step = 1, cv = 5)
selector_cv = selector_cv.fit(ordinalEncoded_X, Y_sampled_train)
```

In [0]:

```
selector.estimator_.coef_
```

Out[343]:

```
array([[-0.12665584, -0.55069278, -0.14243675,  0.6434935 , -0.1279717
4,
         0.11634091,  0.9245736 , -0.90557817, -0.36001417,  0.1617354
7,
         0.12162561,  0.07058609,  0.1498997 , -0.1143375 , -0.1014903
2]])
```

In [0]:

```
selected_cols = pd.Series(index=ordinalEncoded_X.columns).loc[selector.support_].in
```

In [0]:

```
selected_cv_cols = pd.Series(index=ordinalEncoded_X.columns).loc[selector_cv.support
```

In [0]:

```
## 15 columns selected by recursive feature elimination.
selected_cols
```

Out[346]:

```
['ProductName',
 'Platform',
 'Processor',
 'OsVer',
 'OsBuildLab',
 'Census_OSVersion',
 'Census_OSEdition',
 'Census_OSSkuName',
 'AVProductsInstalled',
 'HasTpm',
 'IsProtected',
 'Census_PrimaryDiskTotalCapacity',
 'Census_InternalPrimaryDisplayResolutionHorizontal',
 'Census_InternalPrimaryDisplayResolutionVertical',
 'Census_IsTouchEnabled']
```

In [0]:

```
## columns selected by recursive feature elimination with cross validation (resulti
selected_cv_cols
```

Out[347]:

```
['ProductName',
 'EngineVersion',
 'AvSigVersion',
 'Platform',
 'Processor',
 'OsVer',
 'OsPlatformSubRelease',
 'OsBuildLab',
 'SkuEdition',
 'Census_MDC2FormFactor',
 'Census_ChassisTypeName',
 'Census_OSVersion',
 'Census_OSBranch',
 'Census_OSEdition',
 'Census_OSSkuName',
 'AVProductStatesIdentifier',
 'AVProductsInstalled',
 'HasTpm',
 'OsSuite',
 'IsProtected',
 'IeVerIdentifier',
 'Census_ProcessorCoreCount',
 'Census_ProcessorManufacturerIdentifier',
 'Census_ProcessorModelIdentifier',
 'Census_PrimaryDiskTotalCapacity',
 'Census_SystemVolumeTotalCapacity',
 'Census_HasOpticalDiskDrive',
 'Census_InternalPrimaryDisplayResolutionHorizontal',
 'Census_InternalPrimaryDisplayResolutionVertical',
 'Census_OSBuildRevision',
 'Census_IsSecureBootEnabled',
 'Census_IsTouchEnabled',
 'Census_IsAlwaysOnAlwaysConnectedCapable',
 'Wdft_IsGamer']
```

In [0]:

```
len(selected_cv_cols)
```

Out[348]:

```
34
```

In [0]:

```
# Create new cols
```

Data Preprocessing

Preprocessing

In [19]:

```
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier
```

In [6]:

```python
class AdditiveSmoother(BaseEstimator, ClassifierMixin):

    def __init__(self, alpha=100):
        self.alpha = alpha

    def fit(self, X, y, **kwargs):
        """
        Calculates the smoothed condition empirical
        distributions of the columns of X dependent on y.
        In this case, y is searches in the data.
        """

        self.srate = y.mean()

        smdists = {}

        # loop through the columns of X
        for c in X.columns:
            # create a smoothed empirical distribution for each column in X
            temp = pd.DataFrame({c: X[c], 'HasDetections': y})
            smoothed = ((temp.groupby(c).sum()+self.alpha * self.srate).HasDetections/ \
                        (X[c].value_counts() + self.alpha)).to_dict()
            smdists[c] = smoothed

        # smoothed empirical affected rates in smdists
        self.smdists = smdists

        return self

    def transform(self, X):
        """
        Transforms the categorical values in the columns of X to
        the smoothed affected rates of those values.
        """

        # if len(self.smdists.keys()) == 0:
        #     raise Exception
        toreturn = []
        # loop through columns of X (categorical values)
        for c in X.columns:
            # create a column of smoothed affected rates
            temp = self.smdists[c]
            lst = []
            for i in X[c]:
                if i in temp.keys():
                    lst.append(temp[i])
                else:
                    lst.append(self.srate)
            toreturn.append(lst)
    # return the array of smoothed affected rates.
        return np.array(toreturn).transpose()

    def get_params(self, deep=False):
        """
        Gets the parameters of the transformer;
        Allows Gridsearch to be used with class.
        """
        return {'alpha': self.alpha}
```

In [7]:

```
dtypes = {
        'MachineIdentifier':                                        'category',
        'ProductName':                                              'category',
        'EngineVersion':                                            'category',
        'AppVersion':                                               'category',
        'AvSigVersion':                                             'category',
        'IsBeta':                                                   'int8',
        'RtpStateBitfield':                                         'float16',
        'IsSxsPassiveMode':                                         'int8',
        'DefaultBrowsersIdentifier':                                'float16',
        'AVProductStatesIdentifier':                                'float32',
        'AVProductsInstalled':                                      'float16',
        'AVProductsEnabled':                                        'float16',
        'HasTpm':                                                   'int8',
        'CountryIdentifier':                                        'int16',
        'CityIdentifier':                                           'float32',
        'OrganizationIdentifier':                                   'float16',
        'GeoNameIdentifier':                                        'float16',
        'LocaleEnglishNameIdentifier':                              'int8',
        'Platform':                                                 'category',
        'Processor':                                                'category',
        'OsVer':                                                    'category',
        'OsBuild':                                                  'int16',
        'OsSuite':                                                  'int16',
        'OsPlatformSubRelease':                                     'category',
        'OsBuildLab':                                               'category',
        'SkuEdition':                                               'category',
        'IsProtected':                                              'float16',
        'AutoSampleOptIn':                                          'int8',
        'PuaMode':                                                  'category',
        'SMode':                                                    'float16',
        'IeVerIdentifier':                                          'float16',
        'SmartScreen':                                              'category',
        'Firewall':                                                 'float16',
        'UacLuaenable':                                             'float32',
        'Census_MDC2FormFactor':                                    'category',
        'Census_DeviceFamily':                                      'category',
        'Census_OEMNameIdentifier':                                 'float16',
        'Census_OEMModelIdentifier':                                'float32',
        'Census_ProcessorCoreCount':                                'float16',
        'Census_ProcessorManufacturerIdentifier':                   'float16',
        'Census_ProcessorModelIdentifier':                          'float16',
        'Census_ProcessorClass':                                    'category',
        'Census_PrimaryDiskTotalCapacity':                          'float32',
        'Census_PrimaryDiskTypeName':                               'category',
        'Census_SystemVolumeTotalCapacity':                         'float32',
        'Census_HasOpticalDiskDrive':                               'int8',
        'Census_TotalPhysicalRAM':                                  'float32',
        'Census_ChassisTypeName':                                   'category',
        'Census_InternalPrimaryDiagonalDisplaySizeInInches':        'float16',
        'Census_InternalPrimaryDisplayResolutionHorizontal':        'float16',
        'Census_InternalPrimaryDisplayResolutionVertical':          'float16',
        'Census_PowerPlatformRoleName':                             'category',
        'Census_InternalBatteryType':                               'category',
        'Census_InternalBatteryNumberOfCharges':                    'float32',
        'Census_OSVersion':                                         'category',
        'Census_OSArchitecture':                                    'category',
        'Census_OSBranch':                                          'category',
        'Census_OSBuildNumber':                                     'int16',
```

```
    'Census_OSBuildRevision':                        'int32',
    'Census_OSEdition':                              'category',
    'Census_OSSkuName':                              'category',
    'Census_OSInstallTypeName':                      'category',
    'Census_OSInstallLanguageIdentifier':            'float16',
    'Census_OSUILocaleIdentifier':                   'int16',
    'Census_OSWUAutoUpdateOptionsName':              'category',
    'Census_IsPortableOperatingSystem':              'int8',
    'Census_GenuineStateName':                       'category',
    'Census_ActivationChannel':                      'category',
    'Census_IsFlightingInternal':                    'float16',
    'Census_IsFlightsDisabled':                      'float16',
    'Census_FlightRing':                             'category',
    'Census_ThresholdOptIn':                         'float16',
    'Census_FirmwareManufacturerIdentifier':         'float16',
    'Census_FirmwareVersionIdentifier':              'float32',
    'Census_IsSecureBootEnabled':                    'int8',
    'Census_IsWIMBootEnabled':                       'float16',
    'Census_IsVirtualDevice':                        'float16',
    'Census_IsTouchEnabled':                         'int8',
    'Census_IsPenCapable':                           'int8',
    'Census_IsAlwaysOnAlwaysConnectedCapable':       'float16',
    'Wdft_IsGamer':                                  'float16',
    'Wdft_RegionIdentifier':                         'float16',
    'HasDetections':                                 'int8'
    }
```

In [8]:

```
tbl = pd.read_csv("train.csv",dtype=dtypes)
```

In [9]:

```
sample = tbl.sample(n=len(tbl)//1000,random_state=0)
```

In [10]:

```
cols = ['ProductName',
 'EngineVersion',
 'AvSigVersion',
 'Platform',
 'Processor',
 'OsVer',
 'OsPlatformSubRelease',
 'OsBuildLab',
 'SkuEdition',
 'Census_MDC2FormFactor',
 'Census_ChassisTypeName',
 'Census_OSVersion',
 'Census_OSBranch',
 'Census_OSEdition',
 'AVProductStatesIdentifier',
 'AVProductsInstalled',
 'HasTpm',
 'IsProtected',
 'Census_ProcessorCoreCount',
 'Census_ProcessorManufacturerIdentifier',
 'Census_PrimaryDiskTotalCapacity',
 'Census_SystemVolumeTotalCapacity',
 'Census_HasOpticalDiskDrive',
 'Census_InternalPrimaryDisplayResolutionHorizontal',
 'Census_InternalPrimaryDisplayResolutionVertical',
 'Census_OSBuildRevision',
 'Census_IsSecureBootEnabled',
 'Census_IsTouchEnabled',
 'Census_IsAlwaysOnAlwaysConnectedCapable',
 'Wdft_IsGamer',
        'HasDetections']
```

In [11]:

```
selected = sample[cols]
```

In [12]:

```
# preproc before feature engineering
def clean_MDC2(x):
    if x == 'Notebook':
        return 'Notebook'
    elif x == 'Desktop':
        return 'Desktop'
    elif x in ['Convertible','Detachable','LargeTablet','SmallTablet']:
        return 'Tablet'
    elif x == 'AllInOne':
        return 'AllInOne'
    elif x in ['SmallServer','MediumServer','LargeServer']:
        return 'Server'
    else:
        return 'Other'
#selected['Census_MDC2FormFactor'].apply(clean_MDC2).value_counts()
selected['Census_MDC2FormFactor'] = selected['Census_MDC2FormFactor'].apply(clean_MDC2)
```

```
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
  app.launch_new_instance()
```

In [13]:

```
top5 = selected['AVProductStatesIdentifier'].value_counts().index[:5].tolist()
def clean_AVID(x):
    if x in top5:
        return str(x)
    else:
        return 'other'
selected['AVProductStatesIdentifier'] = selected['AVProductStatesIdentifier'].apply(clean_AVID)
selected['AVProductStatesIdentifier'] = selected.groupby('AVProductStatesIdentifier')['HasDetect
ions'].transform(np.mean)
#selected['AVProductStatesIdentifier']
# this go to additive smoother?
```

```
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
  import sys
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
```

In [14]:

```
selected['Census_PrimaryDiskTotalCapacity'] = selected['Census_PrimaryDiskTotalCapacity'].apply(
lambda x:np.round(np.log2(x)))
selected['Census_SystemVolumeTotalCapacity'] = selected['Census_SystemVolumeTotalCapacity'].appl
y(lambda x:np.round(np.log2(x)))
```

```
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
```

In [15]:

```
mcar_impute_needed = ['Census_ProcessorCoreCount','Census_PrimaryDiskTotalCapacity','Census_Syst
emVolumeTotalCapacity',\
          'Census_InternalPrimaryDisplayResolutionHorizontal','Census_InternalPrimaryDisplayReso
lutionVertical',\
          ]
def create_imputed(col):
    num_null = col.isnull().sum()
    fill_values = col.dropna().sample(num_null, replace=True)
    fill_values.index = col.loc[col.isnull()].index
    return col.fillna(fill_values.to_dict())
for i in mcar_impute_needed:
    selected[i] = create_imputed(selected[i])
```

```
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
  # Remove the CWD from sys.path while we load stuff.
```

In [16]:

```
selected['Census_InternalPrimaryDisplayResolutionHorizontal'] = selected['Census_InternalPrimary
DisplayResolutionHorizontal'].apply(lambda x:'hard_to_be_affected' if x < 1100 or x > 4500 else
'easy_to_be_affected')
selected['Census_InternalPrimaryDisplayResolutionVertical'] = selected['Census_InternalPrimaryDi
splayResolutionVertical'].apply(lambda x:'hard_to_be_affected' if x <1100 else 'easy_to_be_affec
ted')
```

```
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
C:\Users\balalabala\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
indexing.html#indexing-view-versus-copy
```

In [20]:

```
onehot = ['SkuEdition','Census_MDC2FormFactor','HasTpm','IsProtected','Census_IsAlwaysOnAlwaysConnectedCapable','Wdft_IsGamer']
ordinal = ['AVProductsInstalled']
addsmooth = ['ProductName','EngineVersion','AvSigVersion','Platform','Processor','OsVer','OsPlatformSubRelease',\
            'OsBuildLab','Census_ChassisTypeName','Census_OSVersion','Census_OSBranch','Census_OSEdition',\
            'AVProductStatesIdentifier','Census_ProcessorManufacturerIdentifier','Census_OSBuildRevision',\
            'Census_InternalPrimaryDisplayResolutionHorizontal','Census_InternalPrimaryDisplayResolutionVertical']

onehot_pl = Pipeline([
    ('impute',SimpleImputer(strategy='constant',fill_value='unknown')),
    ('str', FunctionTransformer(lambda x: x.astype(str), validate=False)),
    ('onehot',OneHotEncoder(handle_unknown='ignore'))
])

ordinal_pl = Pipeline([
    ('impute',SimpleImputer(strategy='constant',fill_value=-1)),
    ('ordinal',OrdinalEncoder())
])

preproc = ColumnTransformer([
    #('draw_from_dist',FunctionTransformer(create_imputed,validate=False),numeric),
    ('onehot',onehot_pl,onehot),
    ('ordinal',ordinal_pl,ordinal),
    ('additive_smooth',AdditiveSmoother(100),addsmooth)  # did not handle NaN like in the homework
], remainder='passthrough')


# do modeling stuffs...

final_pl = Pipeline([
     ('preproc',preproc),
     ('model',RandomForestClassifier())
])

# final_pl.fit(selected.drop('HasDetections',axis=1),selected.HasDetections)
# final_pl.predict()
# from sklearn import metrics
# metrics.f1_score()
```

In [18]:

```
out = pd.DataFrame(preproc.fit_transform(selected.drop('HasDetections',axis=1),selected.HasDetec
tions))
out['label'] = selected.HasDetections.tolist()
out.to_csv('processed.csv')
out
```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.529669	0.499113	0.495819	12.0
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.529669	0.499113	0.459482	4.0
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.496539	0.499113	0.495819	2.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.514455	0.499113	0.495819	4.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.529669	0.499113	0.495819	4.0
5	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.450726	0.499113	0.495819	3.0
6	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.510771	0.410408	0.495819	4.0
7	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.567687	0.499113	0.495819	2.0
8	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.465428	0.499113	0.495819	2.0
9	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.514455	0.499113	0.495819	2.0
10	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.508632	0.499113	0.495819	4.0
11	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.505139	0.499113	0.495819	4.0
12	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.529669	0.499113	0.495819	4.0
13	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.444824	0.499113	0.495819	4.0
14	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.484209	0.499113	0.495819	8.0
15	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.456812	0.499113	0.495819	4.0
16	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.487415	0.499113	0.495819	4.0
17	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.488831	0.499113	0.495819	4.0
18	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.494343	0.499113	0.495819	4.0
19	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.503088	0.499113	0.495819	2.0
20	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.529669	0.499113	0.495819	4.0
21	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.502702	0.499113	0.495819	4.0
22	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.496539	0.499113	0.495819	4.0
23	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.529669	0.499113	0.495819	8.0
24	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.505139	0.410408	0.495819	4.0
25	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.488831	0.499113	0.495819	2.0
26	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.465648	0.499113	0.495819	4.0
27	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.503164	0.499113	0.495819	2.0
28	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.504513	0.499113	0.495819	4.0
29	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.450726	0.499113	0.495819	4.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8891	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.529669	0.499113	0.495819	4.0
8892	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.547336	0.499113	0.459482	4.0
8893	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.510771	0.499113	0.495819	2.0
8894	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.450726	0.410408	0.495819	2.0
8895	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.494011	0.499113	0.495819	2.0
8896	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.502092	0.499113	0.495819	2.0

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
8897	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.488831	0.499113	0.495819	4.0
8898	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.483309	0.499113	0.495819	8.0
8899	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.491388	0.499113	0.495819	2.0
8900	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.547336	0.499113	0.495819	4.0
8901	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.451031	0.499113	0.495819	4.0
8902	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.510771	0.499113	0.495819	2.0
8903	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.503088	0.499113	0.495819	4.0
8904	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.465648	0.499113	0.495819	2.0
8905	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.483309	0.499113	0.495819	2.0
8906	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.452754	0.499113	0.495819	1.0
8907	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.452754	0.499113	0.495819	8.0
8908	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.482838	0.499113	0.495819	2.0
8909	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.514529	0.499113	0.495819	8.0
8910	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.512862	0.410408	0.495819	2.0
8911	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.547336	0.499113	0.495819	4.0
8912	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.547336	0.499113	0.495819	4.0
8913	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.503164	0.499113	0.495819	4.0
8914	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.451031	0.499113	0.495819	2.0
8915	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.547336	0.499113	0.495819	4.0
8916	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.384208	0.499113	0.495819	4.0
8917	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.465428	0.499113	0.495819	2.0
8918	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.504513	0.499113	0.495819	8.0
8919	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.510771	0.499113	0.495819	4.0
8920	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.529669	0.499113	0.495819	8.0

8921 rows × 50 columns

In [ ]:

```

```

Classification

In [12]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report, co
```

In [2]:

```
# load datasets
data_path1 = './processed.csv'
df = pd.read_csv(data_path1)
df.isnull().sum()
```

Out[2]:

```
Unnamed: 0    0
0             0
1             0
2             0
3             0
4             0
5             0
6             0
7             0
8             0
9             0
10            0
11            0
12            0
13            0
14            0
15            0
16            0
17            0
18            0
19            0
20            0
21            0
22            0
23            0
24            0
25            0
26            0
27            0
28            0
29            0
30            0
31            0
32            0
33            0
34            0
35            0
36            0
37            0
38            0
39            0
40            0
41            0
42            0
43            0
44            0
45            0
46            0
47            0
48            0
label         0
dtype: int64
```

In [10]:

```
def draw_heatmap(score, lists, acc_desc, hyper_p):
    fig, ax = plt.subplots(figsize=(2,4))
    ax = sns.heatmap(score, annot=True, fmt='.3f', yticklabels=lists, xticklabels=[
    ax.collections[0].colorbar.set_label("accuracy")
    plt.title(acc_desc)
    ax.set(xlabel=hyper_p)
def model_outcomes(predictions, target):

    df = pd.DataFrame(index = range(len(target)), columns=['FP', 'FN', 'TP', 'TN'])
    for i in df.index:
        if predictions[i] == 1 and target[i] == 1:
            df.loc[i, 'TP'] = 1
        elif predictions[i] == 1 and target[i] == 0:
            df.loc[i, 'FP'] = 1
        elif predictions[i] == 0 and target[i] == 1:
            df.loc[i, 'FN'] = 1
        elif predictions[i] == 0 and target[i] == 0:
            df.loc[i, 'TN'] = 1
    df = df.fillna(0)
    return df
def metrics(predictions, target):

    df = model_outcomes(predictions, target)
    acc = (np.sum(df['TP']) + np.sum(df['TN']))/len(df)
    recall = np.sum(df['TP'])/(np.sum(df['TP']) + np.sum(df['FN']))
    specificity = np.sum(df['TN'])/(np.sum(df['TN']) + np.sum(df['FP']))
    precision = np.sum(df['TP'])/(np.sum(df['TP']) + np.sum(df['FP']))
    FNR = 1 - recall
    FPR = 1 - specificity
    FDR = np.sum(df['FP'])/(np.sum(df['FP']) + np.sum(df['TP']))
    F1 = 2*(precision*recall)/(precision + recall)
    return pd.Series(data = [acc, recall, specificity, precision, FNR, FPR, FDR, F1
```

In [4]:

```
y = df['label']
X = df.drop(['label'], axis = 1)
print(X.shape, y.shape)
```

```
(8921, 50) (8921,)
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_st
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(7136, 50) (1785, 50) (7136,) (1785,)
```

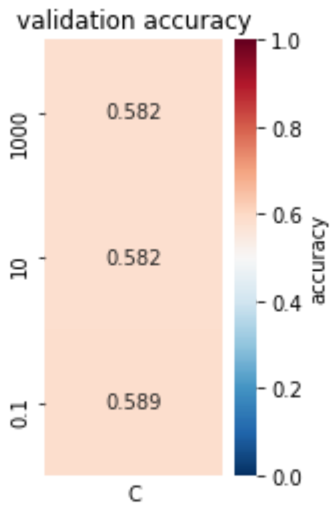
In [6]:

```
C_list = [10**(3), 10, 0.1]
# Train classifiers
logreg = LogisticRegression(solver = 'lbfgs', max_iter = 100000)
clf_log = GridSearchCV(logreg, [{'C':C_list}], cv=5, scoring='accuracy', n_jobs = -1
clf_log.fit(X_train,y_train)

# print best hyperparameters and output grid search heatmap
print('Best parameters: ' + str(clf_log.best_params_))
means = clf_log.cv_results_['mean_test_score'].reshape(3,1)
draw_heatmap(means, C_list, 'validation accuracy', 'C')
pd.Series(data = [means[0][0],means[1][0],means[2][0]], index = ['accuracy_1000','ac
```

```
Best parameters: {'C': 0.1}
```

Out[6]:

	accuracy_1000	accuracy_10	accuracy_0.1
Training accuracy	0.582259	0.582259	0.588705



Type *Markdown* and LaTeX: α^2

In [7]:

```
pd.Series(data = [means[0][0],means[1][0],means[2][0]], index = ['accuracy_1000','ac
```

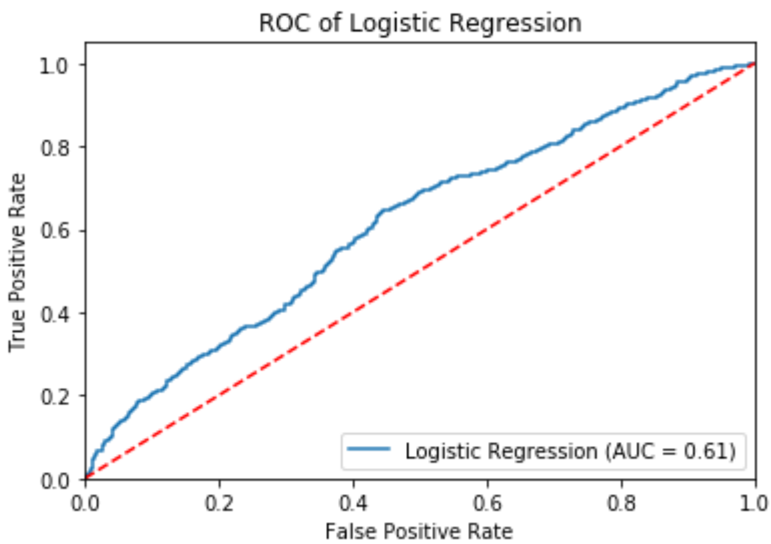
Out[7]:

	accuracy_1000	accuracy_10	accuracy_0.1
Training accuracy	0.582259	0.582259	0.588705

In [8]:

```
# Evaluate the result of classifier
y_pred = clf_log.predict(X_test)
y_pred_proba = clf_log.predict_proba(X_test)[:,1]

# print accuracy
print('The overall accuracy for logistic regression is: %.3f' %accuracy_score(y_test

# print report and ROC curve
print(classification_report(y_test, y_pred))
logit_roc_auc = roc_auc_score(y_test, y_pred_proba)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (AUC = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Logistic Regression')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

```
The overall accuracy for logistic regression is: 0.589
              precision    recall  f1-score   support

           0       0.61      0.57      0.59       915
           1       0.57      0.61      0.59       870

   micro avg       0.59      0.59      0.59      1785
   macro avg       0.59      0.59      0.59      1785
weighted avg       0.59      0.59      0.59      1785
```



In [15]:

```
metrics(y_pred, y_test.values).rename('Testing metrics').to_frame().T
```

Out[15]:

	acc	recall	specificity	precision	fnr	fpr	fdr	f1
Testing metrics	0.588796	0.611494	0.567213	0.573276	0.388506	0.432787	0.426724	0.591769

In [ ]:

```

```

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.impute import SimpleImputer
```

In [23]:

```
class AdditiveSmoother(BaseEstimator, ClassifierMixin):

    def __init__(self, alpha=100):
        self.alpha = alpha

    def fit(self, X, y, **kwargs):
        """
        Calculates the smoothed condition empirical
        distributions of the columns of X dependent on y.
        In this case, y is searches in the data.
        """

        self.srate = y.mean()

        smdists = {}

        # loop through the columns of X
        for c in X.columns:
            # create a smoothed empirical distribution for each column in X
            temp = pd.DataFrame({c: X[c], 'HasDetections': y})
            smoothed = ((temp.groupby(c).sum()+self.alpha * self.srate).HasDetectio
                        (X[c].value_counts() + self.alpha)).to_dict()
            smdists[c] = smoothed

        # smoothed empirical affected rates in smdists
        self.smdists = smdists

        return self

    def transform(self, X):
        """
        Transforms the categorical values in the columns of X to
        the smoothed affected rates of those values.
        """

        # if len(self.smdists.keys()) == 0:
        #     raise Exception
        toreturn = []
        # loop through columns of X (categorical values)
        for c in X.columns:
            # create a column of smoothed affected rates
            temp = self.smdists[c]
            lst = []
            for i in X[c]:
                if i in temp.keys():
                    lst.append(temp[i])
                else:
                    lst.append(self.srate)
            toreturn.append(lst)
    # return the array of smoothed affected rates.
        return np.array(toreturn).transpose()

    def get_params(self, deep=False):
        """
        Gets the parameters of the transformer;
        Allows Gridsearch to be used with class.
        """
        return {'alpha': self.alpha}
```

In [4]:

```
dtypes = {
        'MachineIdentifier':                                    'category',
        'ProductName':                                          'category',
        'EngineVersion':                                        'category',
        'AppVersion':                                           'category',
        'AvSigVersion':                                         'category',
        'IsBeta':                                               'int8',
        'RtpStateBitfield':                                     'float16',
        'IsSxsPassiveMode':                                     'int8',
        'DefaultBrowsersIdentifier':                            'float16',
        'AVProductStatesIdentifier':                            'float32',
        'AVProductsInstalled':                                  'float16',
        'AVProductsEnabled':                                    'float16',
        'HasTpm':                                               'int8',
        'CountryIdentifier':                                    'int16',
        'CityIdentifier':                                       'float32',
        'OrganizationIdentifier':                               'float16',
        'GeoNameIdentifier':                                    'float16',
        'LocaleEnglishNameIdentifier':                          'int8',
        'Platform':                                             'category',
        'Processor':                                            'category',
        'OsVer':                                                'category',
        'OsBuild':                                              'int16',
        'OsSuite':                                              'int16',
        'OsPlatformSubRelease':                                 'category',
        'OsBuildLab':                                           'category',
        'SkuEdition':                                           'category',
        'IsProtected':                                          'float16',
        'AutoSampleOptIn':                                      'int8',
        'PuaMode':                                              'category',
        'SMode':                                                'float16',
        'IeVerIdentifier':                                      'float16',
        'SmartScreen':                                          'category',
        'Firewall':                                             'float16',
        'UacLuaenable':                                         'float32',
        'Census_MDC2FormFactor':                                'category',
        'Census_DeviceFamily':                                  'category',
        'Census_OEMNameIdentifier':                             'float16',
        'Census_OEMModelIdentifier':                            'float32',
        'Census_ProcessorCoreCount':                            'float16',
        'Census_ProcessorManufacturerIdentifier':               'float16',
        'Census_ProcessorModelIdentifier':                      'float16',
        'Census_ProcessorClass':                                'category',
        'Census_PrimaryDiskTotalCapacity':                      'float32',
        'Census_PrimaryDiskTypeName':                           'category',
        'Census_SystemVolumeTotalCapacity':                     'float32',
        'Census_HasOpticalDiskDrive':                           'int8',
        'Census_TotalPhysicalRAM':                              'float32',
        'Census_ChassisTypeName':                               'category',
        'Census_InternalPrimaryDiagonalDisplaySizeInInches':    'float16',
        'Census_InternalPrimaryDisplayResolutionHorizontal':    'float16',
        'Census_InternalPrimaryDisplayResolutionVertical':      'float16',
        'Census_PowerPlatformRoleName':                         'category',
        'Census_InternalBatteryType':                           'category',
        'Census_InternalBatteryNumberOfCharges':                'float32',
        'Census_OSVersion':                                     'category',
        'Census_OSArchitecture':                                'category',
        'Census_OSBranch':                                      'category',
        'Census_OSBuildNumber':                                 'int16',
```

```
          'Census_OSBuildRevision':                                  'int32',
          'Census_OSEdition':                                        'category',
          'Census_OSSkuName':                                        'category',
          'Census_OSInstallTypeName':                                'category',
          'Census_OSInstallLanguageIdentifier':                      'float16',
          'Census_OSUILocaleIdentifier':                             'int16',
          'Census_OSWUAutoUpdateOptionsName':                        'category',
          'Census_IsPortableOperatingSystem':                        'int8',
          'Census_GenuineStateName':                                 'category',
          'Census_ActivationChannel':                                'category',
          'Census_IsFlightingInternal':                              'float16',
          'Census_IsFlightsDisabled':                                'float16',
          'Census_FlightRing':                                       'category',
          'Census_ThresholdOptIn':                                   'float16',
          'Census_FirmwareManufacturerIdentifier':                   'float16',
          'Census_FirmwareVersionIdentifier':                        'float32',
          'Census_IsSecureBootEnabled':                              'int8',
          'Census_IsWIMBootEnabled':                                 'float16',
          'Census_IsVirtualDevice':                                  'float16',
          'Census_IsTouchEnabled':                                   'int8',
          'Census_IsPenCapable':                                     'int8',
          'Census_IsAlwaysOnAlwaysConnectedCapable':                 'float16',
          'Wdft_IsGamer':                                            'float16',
          'Wdft_RegionIdentifier':                                   'float16',
          'HasDetections':                                           'int8'
          }
```

In [5]:

```
tbl = pd.read_csv("train.csv",dtype=dtypes)
```

In [24]:

```
sample = tbl.sample(n=len(tbl)//1000,random_state=0)
```

In [25]:

```
cols = ['ProductName',
 'EngineVersion',
 'AvSigVersion',
 'Platform',
 'Processor',
 'OsVer',
 'OsPlatformSubRelease',
 'OsBuildLab',
 'SkuEdition',
 'Census_MDC2FormFactor',
 'Census_ChassisTypeName',
 'Census_OSVersion',
 'Census_OSBranch',
 'Census_OSEdition',
 'AVProductStatesIdentifier',
 'AVProductsInstalled',
 'HasTpm',
 'IsProtected',
 'Census_ProcessorCoreCount',
 'Census_ProcessorManufacturerIdentifier',
 'Census_PrimaryDiskTotalCapacity',
 'Census_SystemVolumeTotalCapacity',
 'Census_HasOpticalDiskDrive',
 'Census_InternalPrimaryDisplayResolutionHorizontal',
 'Census_InternalPrimaryDisplayResolutionVertical',
 'Census_OSBuildRevision',
 'Census_IsSecureBootEnabled',
 'Census_IsTouchEnabled',
 'Census_IsAlwaysOnAlwaysConnectedCapable',
 'Wdft_IsGamer',
        'HasDetections']
```

In [26]:

```
selected = sample[cols]
{col: selected[col].unique() for col in cols}
```

Out[26]:

```
{'ProductName': [win8defender, mse]
 Categories (2, object): [win8defender, mse],
 'EngineVersion': [1.1.15200.1, 1.1.15100.1, 1.1.14800.3, 1.1.14901.4,
1.1.15000.2, ..., 1.1.12805.0, 1.1.14700.4, 1.1.11701.0, 1.1.14700.3,
1.1.14800.1]
 Length: 35
 Categories (35, object): [1.1.15200.1, 1.1.15100.1, 1.1.14800.3, 1.1.
14901.4, ..., 1.1.14700.4, 1.1.11701.0, 1.1.14700.3, 1.1.14800.1],
 'AvSigVersion': [1.275.1001.0, 1.275.586.0, 1.275.795.0, 1.275.11.0,
1.275.1739.0, ..., 1.263.152.0, 1.247.482.0, 1.265.419.0, 1.271.978.0,
1.275.588.0]
 Length: 1579
 Categories (1579, object): [1.275.1001.0, 1.275.586.0, 1.275.795.0,
1.275.11.0, ..., 1.247.482.0, 1.265.419.0, 1.271.978.0, 1.275.588.0],
 'Platform': [windows10, windows8, windows7, windows2016]
 Categories (4, object): [windows10, windows8, windows7, windows2016],
 'Processor': [x64, x86]
 Categories (2, object): [x64, x86],
```

In [27]:

```
# preproc before feature engineering
def clean_MDC2(x):
    if x == 'Notebook':
        return 'Notebook'
    elif x == 'Desktop':
        return 'Desktop'
    elif x in ['Convertible','Detachable','LargeTablet','SmallTablet']:
        return 'Tablet'
    elif x == 'AllInOne':
        return 'AllInOne'
    elif x in ['SmallServer','MediumServer','LargeServer']:
        return 'Server'
    else:
        return 'Other'
#selected['Census_MDC2FormFactor'].apply(clean_MDC2).value_counts()
selected['Census_MDC2FormFactor'] = selected['Census_MDC2FormFactor'].apply(clean_MI
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  app.launch_new_instance()
```

In [28]:

```
top5 = selected['AVProductStatesIdentifier'].value_counts().index[:5].tolist()
def clean_AVID(x):
    if x in top5:
        return str(x)
    else:
        return 'other'
selected['AVProductStatesIdentifier'] = selected['AVProductStatesIdentifier'].apply(
selected['AVProductStatesIdentifier'] = selected.groupby('AVProductStatesIdentifier
#selected['AVProductStatesIdentifier']
# this go to additive smoother?
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  import sys
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [29]:

```
selected['Census_PrimaryDiskTotalCapacity'] = selected['Census_PrimaryDiskTotalCapac
selected['Census_SystemVolumeTotalCapacity'] = selected['Census_SystemVolumeTotalCap
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [30]:

```
mcar_impute_needed = ['Census_ProcessorCoreCount','Census_PrimaryDiskTotalCapacity',
          'Census_InternalPrimaryDisplayResolutionHorizontal','Census_InternalPrimai
          ]
def create_imputed(col):
    num_null = col.isnull().sum()
    fill_values = col.dropna().sample(num_null, replace=True)
    fill_values.index = col.loc[col.isnull()].index
    return col.fillna(fill_values.to_dict())
for i in mcar_impute_needed:
    selected[i] = create_imputed(selected[i])
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  # Remove the CWD from sys.path while we load stuff.
```

In [31]:

```
selected['Census_InternalPrimaryDisplayResolutionHorizontal'] = selected['Census_Int
selected['Census_InternalPrimaryDisplayResolutionVertical'] = selected['Census_Inte
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [32]:

```
onehot = ['SkuEdition','Census_MDC2FormFactor','HasTpm','IsProtected','Census_IsAlwa
ordinal = ['AVProductsInstalled']
addsmooth = ['ProductName','EngineVersion','AvSigVersion','Platform','Processor','Os
             'OsBuildLab','Census_ChassisTypeName','Census_OSVersion','Census_OSBranc
             'AVProductStatesIdentifier','Census_ProcessorManufacturerIdentifier','Ce
             'Census_InternalPrimaryDisplayResolutionHorizontal','Census_InternalPrin

onehot_pl = Pipeline([
    ('impute',SimpleImputer(strategy='constant',fill_value='unknown')),
    ('str', FunctionTransformer(lambda x: x.astype(str), validate=False)),
    ('onehot',OneHotEncoder(handle_unknown='ignore'))
])

ordinal_pl = Pipeline([
    ('impute',SimpleImputer(strategy='constant',fill_value=-1)),
    ('ordinal',OrdinalEncoder())
])

preproc = ColumnTransformer([
    #('draw_from_dist',FunctionTransformer(create_imputed,validate=False),numeric),
    ('onehot',onehot_pl,onehot),
    ('ordinal',ordinal_pl,ordinal),
    ('additive_smooth',AdditiveSmoother(100),addsmooth)  # did not handle NaN like 
], remainder='passthrough')


# do modeling stuffs...

final_pl = Pipeline([
     ('preproc',preproc),
     ('model',RandomForestClassifier())
])

# final_pl.fit(selected.drop('HasDetections',axis=1),selected.HasDetections)
# final_pl.predict()
# from sklearn import metrics
# metrics.f1_score()
```

In [33]:

```
out = pd.DataFrame(preproc.fit_transform(selected.drop('HasDetections',axis=1),selec
out['label'] = selected.HasDetections.tolist()
out.to_csv('processed.csv')
out
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	4
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.529669	0.499288	0.495646	12.0	17.
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.529669	0.499288	0.462503	4.0	19.
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.496539	0.499288	0.495646	2.0	18.
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.514455	0.499288	0.495646	4.0	18.
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.529669	0.499288	0.495646	4.0	19.
5	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.450726	0.499288	0.495646	3.0	20.
6	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.510771	0.407786	0.495646	4.0	19.
7	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.567687	0.499288	0.495646	2.0	18.
8	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.465428	0.499288	0.495646	2.0	19.
9	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.514455	0.499288	0.495646	2.0	17.
10	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.508632	0.499288	0.495646	4.0	14.
11	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.505139	0.499288	0.495646	4.0	18.
12	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.529669	0.499288	0.495646	4.0	17.
13	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.444824	0.499288	0.495646	4.0	20.
14	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.484209	0.499288	0.495646	8.0	18.
15	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.456812	0.499288	0.495646	4.0	19.
16	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.487415	0.499288	0.495646	4.0	20.
17	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.488831	0.499288	0.495646	4.0	19.
18	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.494343	0.499288	0.495646	4.0	15.
19	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.503088	0.499288	0.495646	2.0	16.
20	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.529669	0.499288	0.495646	4.0	19.
21	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.502702	0.499288	0.495646	4.0	19.
22	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.496539	0.499288	0.495646	4.0	14.
23	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.529669	0.499288	0.495646	8.0	17.
24	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.505139	0.407786	0.495646	4.0	17.
25	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.488831	0.499288	0.495646	2.0	19.
26	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.465648	0.499288	0.495646	4.0	17.
27	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.503164	0.499288	0.495646	2.0	16.
28	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.504513	0.499288	0.495646	4.0	17.
29	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.450726	0.499288	0.495646	4.0	17.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
8891	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.529669	0.499288	0.495646	4.0	19.

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	4
8892	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.547336	0.499288	0.462503	4.0	17.
8893	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.510771	0.499288	0.495646	2.0	17.
8894	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.450726	0.407786	0.495646	2.0	20.
8895	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.494011	0.499288	0.495646	2.0	18.
8896	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.502092	0.499288	0.495646	2.0	19.
8897	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.488831	0.499288	0.495646	4.0	19.
8898	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.483309	0.499288	0.495646	8.0	17.
8899	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.491388	0.499288	0.495646	2.0	19.
8900	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.547336	0.499288	0.495646	4.0	20.
8901	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.451031	0.499288	0.495646	4.0	19.
8902	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.510771	0.499288	0.495646	2.0	19.
8903	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.503088	0.499288	0.495646	4.0	20.
8904	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.465648	0.499288	0.495646	2.0	19.
8905	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.483309	0.499288	0.495646	2.0	19.
8906	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.452754	0.499288	0.495646	1.0	17.
8907	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.452754	0.499288	0.495646	8.0	20.
8908	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.482838	0.499288	0.495646	2.0	20.
8909	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.514529	0.499288	0.495646	8.0	18.
8910	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.512862	0.407786	0.495646	2.0	19.
8911	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.547336	0.499288	0.495646	4.0	20.
8912	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.547336	0.499288	0.495646	4.0	20.
8913	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.503164	0.499288	0.495646	4.0	20.
8914	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.451031	0.499288	0.495646	2.0	17.
8915	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.547336	0.499288	0.495646	4.0	17.
8916	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.384208	0.499288	0.495646	4.0	17.
8917	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.465428	0.499288	0.495646	2.0	15.
8918	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.504513	0.499288	0.495646	8.0	19.
8919	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.510771	0.499288	0.495646	4.0	17.
8920	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.529669	0.499288	0.495646	8.0	18.

8921 rows × 50 columns

In [38]:

```
final_pl = Pipeline([
     #('preproc',preproc),
     ('model',RandomForestClassifier(n_estimators=50, max_depth = 9, max_features='s
                                             min_samples_leaf = 3, min_samples_split = 4))
])

X_train, X_test, y_train, y_test = train_test_split(out.drop('label',axis=1), out.la

final_pl.fit(X_train, y_train)



from sklearn import metrics
[metrics.f1_score(final_pl.predict(X_test), y_test), final_pl.score(X_test, y_test)]
```

Out[38]:

```
[0.7008179078777443, 0.6884805020170327]
```

In [39]:

```
X_test
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	...	39	40	41	42
3271	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.492607	0.483309	0.499288	0.495646
3496	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.501769	0.499288	0.495646
378	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.456812	0.499288	0.495646
3021	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.490782	0.499288	0.495646
4422	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.496539	0.499288	0.495646
1924	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.456812	0.499288	0.495646
2022	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.514529	0.499288	0.495646
4863	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.514455	0.499288	0.495646
471	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.499140	0.499288	0.495646
3936	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.498153	0.465648	0.499288	0.495646
3847	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.492607	0.489115	0.499288	0.495646
1206	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.489808	0.499288	0.495646
324	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.529669	0.499288	0.495646
182	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.444824	0.499288	0.495646
4964	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.483309	0.499288	0.495646
2143	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.498153	0.547336	0.499288	0.495646
6259	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.492607	0.465428	0.499288	0.495646
4409	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.498153	0.529669	0.499288	0.495646
5723	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.498153	0.529669	0.499288	0.495646
249	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.529669	0.499288	0.495646
2729	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.495073	0.499288	0.495646
5028	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.514529	0.499288	0.462503
3956	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.498153	0.513591	0.407786	0.495646
6616	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.456812	0.499288	0.495646
5870	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.499202	0.499288	0.495646
4180	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.498153	0.529669	0.499288	0.495646
4221	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.467065	0.499288	0.495646
4091	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.483113	0.499288	0.495646
5398	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.499018	0.499288	0.495646
7828	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.465428	0.499288	0.495646
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2097	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.503164	0.499288	0.495646
1751	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.529669	0.499288	0.495646
5747	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.465428	0.499288	0.495646

	0	1	2	3	4	5	6	7	8	9	...	39	40	41	42
8790	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.444824	0.499288	0.495646
2660	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.492607	0.514529	0.499288	0.495646
4804	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.547336	0.499288	0.495646
8551	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.510771	0.499288	0.462503
7596	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.490782	0.499288	0.495646
5211	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.514455	0.499288	0.495646
3532	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.467065	0.499288	0.495646
676	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.498153	0.529669	0.499288	0.495646
638	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.485869	0.499288	0.495646
4013	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.503815	0.499288	0.495646
5198	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.498153	0.451031	0.499288	0.495646
5941	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.498153	0.452754	0.499288	0.495646
3015	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.483309	0.499288	0.495646
73	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.510771	0.499288	0.495646
1244	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.488831	0.499288	0.495646
2123	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.487415	0.499288	0.495646
5518	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.492607	0.547336	0.499288	0.495646
4605	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.498153	0.496539	0.499288	0.495646
3141	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.492607	0.450726	0.499288	0.495646
1307	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.492607	0.456812	0.499288	0.495646
1754	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.492607	0.547336	0.499288	0.495646
6257	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.452754	0.499288	0.495646
1983	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.529669	0.499288	0.495646
3319	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.498153	0.492338	0.499288	0.495646
3883	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.529669	0.499288	0.495646
6949	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.510771	0.499288	0.495646
6262	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.492607	0.547336	0.499288	0.495646

2231 rows × 49 columns

In [ ]:

```

```

In [111]:

```
def model_outcomes(predictions, target):
    """
    :Example:
    >>> out = model_outcomes(pd.Series([1,0,1,0]), pd.Series([0,1,1,0]))
    >>> (np.diag(out) == 1).all()
    True
    >>> set(out.columns) == {'FN', 'FP', 'TN', 'TP'}
    True
    """
    df = pd.DataFrame(columns = ['FN', 'FP', 'TN', 'TP'])
    for i in range(len(predictions)):
        if predictions.iloc[i] == 1 and target.iloc[i] == 1:
            df.loc[i] = [0, 0, 0, 1]
        elif predictions.iloc[i] == 0 and target.iloc[i] == 0:
            df.loc[i] = [0, 0, 1, 0]
        elif predictions.iloc[i] == 1 and target.iloc[i] == 0:
            df.loc[i] = [0, 1, 0, 0]
        else:
            df.loc[i] = [1, 0, 0, 0]
    return df[['FP', 'FN', 'TP', 'TN']]
def metrics(predictions, target):
    """
    :Example:
    >>> out = metrics(pd.Series([1,0,1,0]), pd.Series([0,1,1,0]))
    >>> set(out.index) == {'acc', 'f1', 'fdr', 'fnr', 'fpr', 'precision', 'recall',
    True
    >>> (out == 0.5).all()
    True
    """
    outcomes = model_outcomes(predictions, target).sum()
    acc = (outcomes['TP'] + outcomes['TN']) / outcomes.sum()
    specificity = outcomes['TN'] / (outcomes['TN'] + outcomes['FP'])
    precision = outcomes['TP'] / (outcomes['TP'] + outcomes['FP'])
    recall = outcomes['TP'] / (outcomes['TP'] + outcomes['FN'])
    f1 = 2 * precision * recall / (precision + recall)
    fdr = 1 - precision
    fnr = outcomes['FN'] / (outcomes['FN'] + outcomes['TP'])
    fpr = outcomes['FP'] / (outcomes['FP'] + outcomes['TN'])
    dic = {'acc': acc, 'f1': f1, 'fdr': fdr, 'fnr': fnr, 'fpr':fpr, \
           'precision':precision, 'recall':recall, 'specificity':specificity}
    return pd.Series(dic)
metric = metrics(pd.Series(final_pl.predict(X_test)), y_test).to_frame().T
```

In [122]:

```
metric.rename({0:'score'})
```

Out[122]:

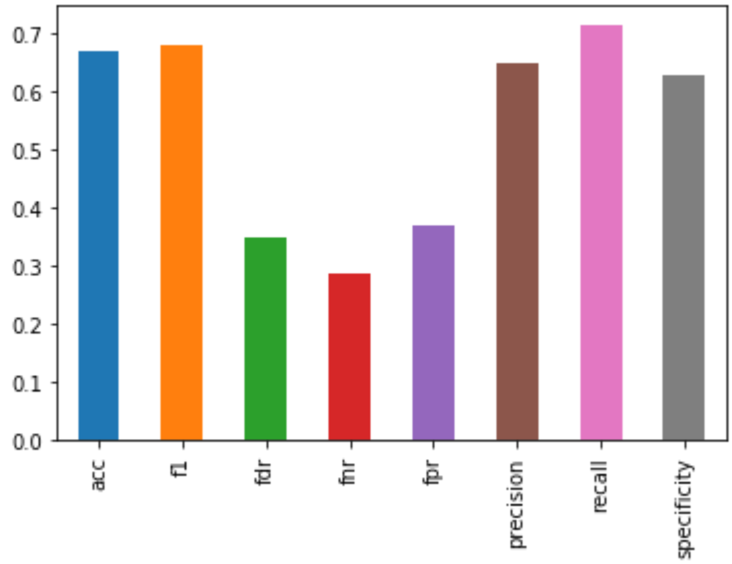
	acc	f1	fdr	fnr	fpr	precision	recall	specificity
score	0.671	0.680314	0.35079	0.285453	0.370826	0.64921	0.714547	0.629174

In [116]:

```
metric.loc[0].plot(kind = 'bar')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a27c3b5f8>
```



In [97]:

```
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier()

param_grid = {
    'n_estimators': [10, 30, 50],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [7, 9, 11],
    'min_samples_split' : [2,3,4],
    'min_samples_leaf' : [1,2,3]
}
X = out.drop('label',axis=1)
y = out.label
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X, y)
CV_rfc.best_params_
```

Out[97]:

```
{'max_depth': 7,
 'max_features': 'sqrt',
 'min_samples_leaf': 3,
 'min_samples_split': 4,
 'n_estimators': 50}
```

In [ ]:

```

```

In [1]:

```
import pandas as pd
import numpy as np
data = pd.read_csv('processed.csv')
```

In [2]:

```
print(np.sum(data['label'] == 1), np.sum(data['label'] != 1))
```

```
4406 4515
```

In [3]:

```
## Optimize the Algorithm:
## Standardization / Whitening /
```

In [4]:

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(
        data.drop(['label'], axis = 1), data['label'], test_size = 0.3, random_state
```

In [12]:

```
Cs = [0.1, 1, 10, 100, 500, 1000, 3000]
penalties = ['l2', 'l1']
param_grid = {'C': Cs, 'penalty': penalties}
svm = GridSearchCV(LinearSVC(dual=False, max_iter=50000), param_grid = param_grid, c
svm.fit(X_train, y_train)
```

Out[12]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LinearSVC(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=50000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.1, 1, 10, 100, 500, 1000, 3000], 'penalty':
['l2', 'l1']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [13]:

```
y_pred = svm.predict(X_test)
```

In [9]:

```
def model_outcomes(predictions, target):

    df = pd.DataFrame(index = range(len(target)), columns=['FP', 'FN', 'TP', 'TN'])
    for i in df.index:
        if predictions[i] == 1 and target[i] == 1:
            df.loc[i, 'TP'] = 1
        elif predictions[i] == 1 and target[i] == 0:
            df.loc[i, 'FP'] = 1
        elif predictions[i] == 0 and target[i] == 1:
            df.loc[i, 'FN'] = 1
        elif predictions[i] == 0 and target[i] == 0:
            df.loc[i, 'TN'] = 1
    df = df.fillna(0)
    return df

def metrics(predictions, target):

    df = model_outcomes(predictions, target)
    acc = (np.sum(df['TP']) + np.sum(df['TN']))/len(df)
    recall = np.sum(df['TP'])/(np.sum(df['TP']) + np.sum(df['FN']))
    specificity = np.sum(df['TN'])/(np.sum(df['TN']) + np.sum(df['FP']))
    precision = np.sum(df['TP'])/(np.sum(df['TP']) + np.sum(df['FP']))
    FNR = 1 - recall
    FPR = 1 - specificity
    FDR = np.sum(df['FP'])/(np.sum(df['FP']) + np.sum(df['TP']))
    F1 = 2*(precision*recall)/(precision + recall)
    return pd.Series(data = [acc, recall, specificity, precision, FNR, FPR, FDR, F1
```

In [15]:

```
## Predict Probability, Metric: ROC
```

In [36]:

```
metrics(y_pred, y_test.values)
```

Out[36]:

```
acc            0.659320
recall         0.651503
specificity    0.666667
precision      0.647510
fnr            0.348497
fpr            0.333333
fdr            0.352490
f1             0.649500
dtype: float64
```

In [18]:

```
svm.best_params_
```

Out[18]:

```
{'C': 100, 'penalty': 'l2'}
```

In [19]:

```
best_params = svm.best_params_
train_scores = svm.cv_results_['mean_train_score']
val_scores = svm.cv_results_['mean_test_score']
```

```
/Users/user/anaconda3/envs/dsc80/lib/python3.7/site-packages/sklearn/u
tils/deprecation.py:125: FutureWarning: You are accessing a training s
core ('mean_train_score'), which will not be available by default any
more in 0.21. If you need training scores, please set return_train_sco
re=True
  warnings.warn(*warn_args, **warn_kwargs)
```

In [35]:

```
svm.best_estimator_
```

Out[35]:

```
LinearSVC(C=100, class_weight=None, dual=False, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=50000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

In [34]:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, svm.best_estimator_.decision_function(X_test))
```

Out[34]:

```
0.7151330271641357
```

In [ ]:

```
```

In [6]:

```
best_svc = LinearSVC(C=100, class_weight=None, dual=False, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=50000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

In [8]:

```
best_svc.fit(X_train, y_train)
y_pred = best_svc.predict(X_test)
```

In [19]:

```
metrics(best_svc.predict(X_train), y_train.values).sort_index().rename('Training met
```

Out[19]:

	acc	f1	fdr	fnr	fpr	precision	recall	specificity
Training metrics	0.662076	0.660335	0.339027	0.340302	0.335566	0.660973	0.659698	0.664434

In [20]:

```
metrics(y_pred, y_test.values).sort_index().rename('Validation metrics').to_frame().
```

Out[20]:

	acc	f1	fdr	fnr	fpr	precision	recall	specificity
Validation metrics	0.65932	0.6495	0.35249	0.348497	0.333333	0.64751	0.651503	0.666667

In [ ]:

```
```

In [1]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('processed.csv')
```

In [2]:
```
print(np.sum(data['label'] == 1), np.sum(data['label'] != 1))
```

```
4406 4515
```

In [3]:
```
## Optimize the Algorithm:
```

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

X_train, X_test, y_train, y_test = train_test_split(
        data.drop(['label'], axis = 1), data['label'], test_size = 0.3
, random_state = 0)
Cs = [0.1, 1, 10, 100, 500, 1000, 3000, 5000]
gammas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]
param_grid = {'C': Cs, 'gamma' : gammas}
svm = GridSearchCV(SVC(kernel='rbf'), param_grid = param_grid, cv = 5)
svm.fit(X_train, y_train)
```

Out[4]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0
=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.1, 1, 10, 100, 500, 1000, 3000, 5000], 'g
amma': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn
',
       scoring=None, verbose=0)
```

In [5]:
```
y_pred = svm.predict(X_test)
```

In [6]:
```
def model_outcomes(predictions, target):

    df = pd.DataFrame(index = range(len(target)), columns=['FP', 'FN', 
'TP', 'TN'])
    for i in df.index:
        if predictions[i] == 1 and target[i] == 1:
            df.loc[i, 'TP'] = 1
        elif predictions[i] == 1 and target[i] == 0:
            df.loc[i, 'FP'] = 1
        elif predictions[i] == 0 and target[i] == 1:
            df.loc[i, 'FN'] = 1
        elif predictions[i] == 0 and target[i] == 0:
            df.loc[i, 'TN'] = 1
    df = df.fillna(0)
    return df

def metrics(predictions, target):

    df = model_outcomes(predictions, target)
    acc = (np.sum(df['TP']) + np.sum(df['TN']))/len(df)
    recall = np.sum(df['TP'])/(np.sum(df['TP']) + np.sum(df['FN']))
    specificity = np.sum(df['TN'])/(np.sum(df['TN']) + np.sum(df['FP']
))
    precision = np.sum(df['TP'])/(np.sum(df['TP']) + np.sum(df['FP']))
    FNR = 1 - recall
    FPR = 1 - specificity
    FDR = np.sum(df['FP'])/(np.sum(df['FP']) + np.sum(df['TP']))
    F1 = 2*(precision*recall)/(precision + recall)
    return pd.Series(data = [acc, recall, specificity, precision, FNR, 
FPR, FDR, F1], index = ['acc', 'recall', \

'specificity', 'precision', 'fnr', 'fpr', 'fdr', 'f1'])
```

In [7]:
```
## Predict Probability, Metric: ROC
```

In [8]:
```
from sklearn.metrics import roc_auc_score
```

In [9]:
```
roc_auc_score(y_true = y_train, y_score = svm.best_estimator_.decision
_function(X_train))
```

Out[9]: 0.6862581905801083

In [10]:
```
roc_auc_score(y_true = y_test, y_score = svm.best_estimator_.decision_
function(X_test))
```

Out[10]: 0.5681371727397673

In [11]:
```
train_metric = metrics(svm.predict(X_train), y_train.values)
```

In [12]:
```
test_metric = metrics(y_pred, y_test.values)
```

In [14]:
```
pd.DataFrame(columns=train_metric.index, data = train_metric.values.reshape(1,-1)).rename(index = {0:'Training metrics'})
```

Out[14]:

	acc	recall	specificity	precision	fnr	fpr	fdr	f1
Training metrics	0.634689	0.62882	0.64051	0.634328	0.37118	0.35949	0.365672	0.631562

In [15]:
```
pd.DataFrame(columns=test_metric.index, data = test_metric.values.reshape(1,-1)).rename(index = {0:'Testing metrics'})
```

Out[15]:

	acc	recall	specificity	precision	fnr	fpr	fdr	f1
Testing metrics	0.54576	0.54973	0.542029	0.530112	0.45027	0.457971	0.469888	0.539743

In [16]:
```
svm.best_params_
```

Out[16]: {'C': 5000, 'gamma': 1e-05}

In [ ]:

In [ ]: