Microsoft Malware Prediction

Team:
Julie Kirkpatrick
Thomas Dolan
Kaggle Team Name:
ThomasDolan (couldn't add Julie for some reason)

Overview of topics covered:

- Data Type Conversion/Loading Data
- Feature Engineering
  - Handling NaNs
  - Fixing Problematic Columns
- Feature Encoding
  - One Hot Encoding
- Modeling
  - Logistic Regression Model
  - K-Nearest Neighbor Model
  - LDA Model
- Submission

Step 1) Imports

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import warnings
import gc
from IPython.display import display

warnings.filterwarnings('ignore')
%matplotlib inline

style.use('seaborn')

pd.options.display.max_columns = None
```

Step 2) Load Data/Convert Data Types

Step 2a) Specify Data Types for Faster Loading (see type-determination.ipynb)

In [2]:
```
data_types = {'MachineIdentifier': 'object',
              'ProductName': 'category',
              'EngineVersion': 'category',
              'AppVersion': 'category',
              'AvSigVersion': 'category',
              'IsBeta': 'float32',
              'RtpStateBitfield': 'float32',
              'IsSxsPassiveMode': 'float32',
              'DefaultBrowsersIdentifier': 'float32',
              'AVProductStatesIdentifier': 'float32',
              'AVProductsInstalled': 'float32',
              'AVProductsEnabled': 'float32',
              'HasTpm': 'float32',
              'CountryIdentifier': 'float32',
              'CityIdentifier': 'float32',
              'OrganizationIdentifier': 'float32',
              'GeoNameIdentifier': 'float32',
              'LocaleEnglishNameIdentifier': 'float32',
              'Platform': 'category',
              'Processor': 'category',
              'OsVer': 'category',
              'OsBuild': 'float32',
              'OsSuite': 'float32',
              'OsPlatformSubRelease': 'category',
              'OsBuildLab': 'category',
              'SkuEdition': 'category',
              'IsProtected': 'float32',
              'AutoSampleOptIn': 'float32',
              'PuaMode': 'category',
              'SMode': 'float32',
              'IeVerIdentifier': 'float32',
              'SmartScreen': 'category',
              'Firewall': 'float32',
              'UacLuaenable': 'float32',
              'Census_MDC2FormFactor': 'category',
              'Census_DeviceFamily': 'category',
              'Census_OEMNameIdentifier': 'float32',
              'Census_OEMModelIdentifier': 'float32',
              'Census_ProcessorCoreCount': 'float32',
              'Census_ProcessorManufacturerIdentifier': 'float32',
              'Census_ProcessorModelIdentifier': 'float32',
              'Census_ProcessorClass': 'category',
              'Census_PrimaryDiskTotalCapacity': 'float32',
              'Census_PrimaryDiskTypeName': 'category',
              'Census_SystemVolumeTotalCapacity': 'float32',
              'Census_HasOpticalDiskDrive': 'float32',
              'Census_TotalPhysicalRAM': 'float32',
              'Census_ChassisTypeName': 'category',
              'Census_InternalPrimaryDiagonalDisplaySizeInInches': 'floa
t32',
              'Census_InternalPrimaryDisplayResolutionHorizontal': 'floa
t32',
              'Census_InternalPrimaryDisplayResolutionVertical': 'float3
2',
              'Census_PowerPlatformRoleName': 'category',
              'Census_InternalBatteryType': 'category',
```

```
                'Census_InternalBatteryNumberOfCharges': 'float32',
                'Census_OSVersion': 'category',
                'Census_OSArchitecture': 'category',
                'Census_OSBranch': 'category',
                'Census_OSBuildNumber': 'float32',
                'Census_OSBuildRevision': 'float32',
                'Census_OSEdition': 'category',
                'Census_OSSkuName': 'category',
                'Census_OSInstallTypeName': 'category',
                'Census_OSInstallLanguageIdentifier': 'float32',
                'Census_OSUILocaleIdentifier': 'float32',
                'Census_OSWUAutoUpdateOptionsName': 'category',
                'Census_IsPortableOperatingSystem': 'float32',
                'Census_GenuineStateName': 'category',
                'Census_ActivationChannel': 'category',
                'Census_IsFlightingInternal': 'float32',
                'Census_IsFlightsDisabled': 'float32',
                'Census_FlightRing': 'category',
                'Census_ThresholdOptIn': 'float32',
                'Census_FirmwareManufacturerIdentifier': 'float32',
                'Census_FirmwareVersionIdentifier': 'float32',
                'Census_IsSecureBootEnabled': 'float32',
                'Census_IsWIMBootEnabled': 'float32',
                'Census_IsVirtualDevice': 'float32',
                'Census_IsTouchEnabled': 'float32',
                'Census_IsPenCapable': 'float32',
                'Census_IsAlwaysOnAlwaysConnectedCapable': 'float32',
                'Wdft_IsGamer': 'float32',
                'Wdft_RegionIdentifier': 'float32',
                'HasDetections': 'float32'}
```

Step 2b) Load Data

In [3]:
```
df = pd.read_csv('../data/train.csv', dtype=data_types)
```

In [4]:
```
df = df.sample(50000)
```

In [5]:
```
test = pd.read_csv('../data/test.csv', dtype=data_types)
```

Step 2c) Convert any attributes that couldn't be converted in read_csv

In [6]:
```
bool_columns = ['IsBeta', 'IsSxsPassiveMode', 'HasTpm', 'AutoSampleOptIn', 'Census_HasOpticalDiskDrive',
       'Census_IsPortableOperatingSystem', 'Census_IsFlightsDisabled', 'Census_IsSecureBootEnabled',
       'Census_IsVirtualDevice', 'Census_IsTouchEnabled', 'Census_IsPenCapable',
       'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer', 'IsProtected', 'SMode',
       'Firewall', 'HasDetections']

unsigned_columns = ['CountryIdentifier', 'LocaleEnglishNameIdentifier',
       'OsBuild', 'OsSuite', 'Census_OSBuildNumber',
       'Census_OSBuildRevision', 'Census_OSUILocaleIdentifier']
```

In [7]:
```
def convert_cols_to_bool(df, cols):
    for col in cols:
        df[col] = df[col].astype('bool')

def convert_cols_to_unsigned(df, cols):
    for col in cols:
        df[col] = pd.to_numeric(df[col], downcast='unsigned')
```

In [8]:
```
convert_cols_to_bool(df, bool_columns)
convert_cols_to_unsigned(df, unsigned_columns)
```

In [9]:
```
bool_columns = ['IsBeta', 'IsSxsPassiveMode', 'HasTpm', 'AutoSampleOptIn', 'Census_HasOpticalDiskDrive',
       'Census_IsPortableOperatingSystem', 'Census_IsFlightsDisabled', 'Census_IsSecureBootEnabled',
       'Census_IsVirtualDevice', 'Census_IsTouchEnabled', 'Census_IsPenCapable',
       'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer', 'IsProtected', 'SMode',
       'Firewall']

unsigned_columns = ['CountryIdentifier', 'LocaleEnglishNameIdentifier',
       'OsBuild', 'OsSuite', 'Census_OSBuildNumber',
       'Census_OSBuildRevision', 'Census_OSUILocaleIdentifier']
```

In [10]:
```
convert_cols_to_bool(test, bool_columns)
convert_cols_to_unsigned(test, unsigned_columns)
```

In [11]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 83)
(7853253, 82)
```

Let's check out the final data types

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 4374155 to 8748389
Data columns (total 83 columns):
MachineIdentifier                                   50000 non-null obj
ect
ProductName                                         50000 non-null cat
egory
EngineVersion                                       50000 non-null cat
egory
AppVersion                                          50000 non-null cat
egory
AvSigVersion                                        50000 non-null cat
egory
IsBeta                                              50000 non-null boo
l
RtpStateBitfield                                    49818 non-null flo
at32
IsSxsPassiveMode                                    50000 non-null boo
l
DefaultBrowsersIdentifier                           2497 non-null floa
t32
AVProductStatesIdentifier                           49790 non-null flo
at32
AVProductsInstalled                                 49790 non-null flo
at32
AVProductsEnabled                                   49790 non-null flo
at32
HasTpm                                              50000 non-null boo
l
CountryIdentifier                                   50000 non-null uin
t8
CityIdentifier                                      48170 non-null flo
at32
OrganizationIdentifier                              34492 non-null flo
at32
GeoNameIdentifier                                   50000 non-null flo
at32
LocaleEnglishNameIdentifier                         50000 non-null uin
t16
Platform                                            50000 non-null cat
egory
Processor                                           50000 non-null cat
egory
OsVer                                               50000 non-null cat
egory
OsBuild                                             50000 non-null uin
t16
OsSuite                                             50000 non-null uin
t16
OsPlatformSubRelease                                50000 non-null cat
egory
OsBuildLab                                          50000 non-null cat
egory
SkuEdition                                          50000 non-null cat
egory
IsProtected                                         50000 non-null boo
l
```

```
AutoSampleOptIn                                   50000 non-null boo
l
PuaMode                                           11 non-null catego
ry
SMode                                             50000 non-null boo
l
IeVerIdentifier                                   49691 non-null flo
at32
SmartScreen                                       32086 non-null cat
egory
Firewall                                          50000 non-null boo
l
UacLuaenable                                      49936 non-null flo
at32
Census_MDC2FormFactor                             50000 non-null cat
egory
Census_DeviceFamily                               50000 non-null cat
egory
Census_OEMNameIdentifier                          49489 non-null flo
at32
Census_OEMModelIdentifier                         49449 non-null flo
at32
Census_ProcessorCoreCount                         49759 non-null flo
at32
Census_ProcessorManufacturerIdentifier            49759 non-null flo
at32
Census_ProcessorModelIdentifier                   49759 non-null flo
at32
Census_ProcessorClass                             201 non-null categ
ory
Census_PrimaryDiskTotalCapacity                   49685 non-null flo
at32
Census_PrimaryDiskTypeName                        49943 non-null cat
egory
Census_SystemVolumeTotalCapacity                  49685 non-null flo
at32
Census_HasOpticalDiskDrive                        50000 non-null boo
l
Census_TotalPhysicalRAM                           49528 non-null flo
at32
Census_ChassisTypeName                            49997 non-null cat
egory
Census_InternalPrimaryDiagonalDisplaySizeInInches 49719 non-null flo
at32
Census_InternalPrimaryDisplayResolutionHorizontal 49720 non-null flo
at32
Census_InternalPrimaryDisplayResolutionVertical   49720 non-null flo
at32
Census_PowerPlatformRoleName                      50000 non-null cat
egory
Census_InternalBatteryType                        14489 non-null cat
egory
Census_InternalBatteryNumberOfCharges             48502 non-null flo
at32
Census_OSVersion                                  50000 non-null cat
egory
Census_OSArchitecture                             50000 non-null cat
```

```
egory
Census_OSBranch                                    50000 non-null cat
egory
Census_OSBuildNumber                               50000 non-null uin
t16
Census_OSBuildRevision                             50000 non-null uin
t16
Census_OSEdition                                   50000 non-null cat
egory
Census_OSSkuName                                   50000 non-null cat
egory
Census_OSInstallTypeName                           50000 non-null cat
egory
Census_OSInstallLanguageIdentifier                 49693 non-null flo
at32
Census_OSUILocaleIdentifier                        50000 non-null uin
t8
Census_OSWUAutoUpdateOptionsName                   50000 non-null cat
egory
Census_IsPortableOperatingSystem                   50000 non-null boo
l
Census_GenuineStateName                            50000 non-null cat
egory
Census_ActivationChannel                           50000 non-null cat
egory
Census_IsFlightingInternal                         8628 non-null floa
t32
Census_IsFlightsDisabled                           50000 non-null boo
l
Census_FlightRing                                  50000 non-null cat
egory
Census_ThresholdOptIn                              18239 non-null flo
at32
Census_FirmwareManufacturerIdentifier              48963 non-null flo
at32
Census_FirmwareVersionIdentifier                   49082 non-null flo
at32
Census_IsSecureBootEnabled                         50000 non-null boo
l
Census_IsWIMBootEnabled                            18279 non-null flo
at32
Census_IsVirtualDevice                             50000 non-null boo
l
Census_IsTouchEnabled                              50000 non-null boo
l
Census_IsPenCapable                                50000 non-null boo
l
Census_IsAlwaysOnAlwaysConnectedCapable            50000 non-null boo
l
Wdft_IsGamer                                       50000 non-null boo
l
Wdft_RegionIdentifier                              48242 non-null flo
at32
HasDetections                                      50000 non-null boo
l
dtypes: bool(17), category(29), float32(29), object(1), uint16(5), uint
```

```
8(2)
memory usage: 9.7+ MB
```

Step 3) Dealing with Missing Data

Step 3a) Drop columns with a lot of missing data

First, let's keep a running list of the columns we will be dropping. We'll use it later on the test set

In [13]:
```
all_dropped = []
```

Find the columns with lots of missing data

In [14]:
```
def get_cols_with_many_nans(df, identifier, percentage=0.2):
    """
    Find columns within a dataframe that contain less than x% non-NaN va
lues.

    :param df (DataFrame): A pandas dataframe
    :param identifier (string): Any column with no NaNs, used to compute
the threshold (not actually altered)
    :param percentage (float): The lowest amount of non-NaN values you'r
e willing to tolerate.
    :return: (list): Columns that don't meet the minimum number of non-n
ull values.
    """
    result = []
    total_rows = df[identifier].count()
    threshold = int(total_rows * percentage)
    for col in df.columns:
        if df[col].isna().sum() > threshold:
            result.append(col)
    return result
```

In [15]:
```
cols_with_many_nans = get_cols_with_many_nans(df, 'MachineIdentifier')

all_dropped += cols_with_many_nans  # Add it to our running list

print("List of columns with a lot of missing values:\n")
print("COLUMN | % MISSING\n")
for col in cols_with_many_nans:
    print(f"{col} | {((df[col].isna().sum() / df['HasDetections'].count
()) * 100):2f}%")
```

```
List of columns with a lot of missing values:

COLUMN | % MISSING

DefaultBrowsersIdentifier | 95.006000%
OrganizationIdentifier | 31.016000%
PuaMode | 99.978000%
SmartScreen | 35.828000%
Census_ProcessorClass | 99.598000%
Census_InternalBatteryType | 71.022000%
Census_IsFlightingInternal | 82.744000%
Census_ThresholdOptIn | 63.522000%
Census_IsWIMBootEnabled | 63.442000%
```

Drop the columns with lots of missing data

In [16]:
```
df.drop(cols_with_many_nans, inplace=True, axis=1)
```

In [17]:
```
test.drop(cols_with_many_nans, inplace=True, axis=1)
```

In [18]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 74)
(7853253, 73)
```

Step 3b) Fill NaNs in rows

In [21]:
```
df.isna().sum()
```

Out[21]:

MachineIdentifier	0
ProductName	0
EngineVersion	0
AppVersion	0
AvSigVersion	0
IsBeta	0
RtpStateBitfield	182
IsSxsPassiveMode	0
AVProductStatesIdentifier	210
AVProductsInstalled	210
AVProductsEnabled	210
HasTpm	0
CountryIdentifier	0
CityIdentifier	1830
GeoNameIdentifier	0
LocaleEnglishNameIdentifier	0
Platform	0
Processor	0
OsVer	0
OsBuild	0
OsSuite	0
OsPlatformSubRelease	0
OsBuildLab	0
SkuEdition	0
IsProtected	0
AutoSampleOptIn	0
SMode	0
IeVerIdentifier	309
Firewall	0
UacLuaenable	64
...	...
Census_InternalPrimaryDisplayResolutionHorizontal	280
Census_InternalPrimaryDisplayResolutionVertical	280
Census_PowerPlatformRoleName	0
Census_InternalBatteryNumberOfCharges	1498
Census_OSVersion	0
Census_OSArchitecture	0
Census_OSBranch	0
Census_OSBuildNumber	0
Census_OSBuildRevision	0
Census_OSEdition	0
Census_OSSkuName	0
Census_OSInstallTypeName	0
Census_OSInstallLanguageIdentifier	307
Census_OSUILocaleIdentifier	0
Census_OSWUAutoUpdateOptionsName	0
Census_IsPortableOperatingSystem	0
Census_GenuineStateName	0
Census_ActivationChannel	0
Census_IsFlightsDisabled	0
Census_FlightRing	0
Census_FirmwareManufacturerIdentifier	1037
Census_FirmwareVersionIdentifier	918
Census_IsSecureBootEnabled	0
Census_IsVirtualDevice	0
Census_IsTouchEnabled	0
Census_IsPenCapable	0

```
Census_IsAlwaysOnAlwaysConnectedCapable                 0
Wdft_IsGamer                                            0
Wdft_RegionIdentifier                                1758
HasDetections                                           0
Length: 74, dtype: int64
```

In [36]:
```
def fillna_mean(df, col):
    if df[col].dtype.name == 'category':
        df[col] = df[col].cat.add_categories('UNK')
        df[col].fillna('UNK', inplace=True)
    else:
        mean = df[col].mean()
        df[col].fillna(mean, inplace=True)
```

In [37]:
```
test_cols = []
for col in test.columns:
    if test[col].isna().sum():
        test_cols.append(col)

df_cols = []
for col in df.columns:
    if df[col].isna().sum():
        df_cols.append(col)
```

In [38]:
```
for col in test_cols:
    fillna_mean(test, col)

for col in df_cols:
    fillna_mean(df, col)
```

In [41]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 74)
(7853253, 73)
```

Step 4) Drop or Fix Potentially Problematic Columns

Step 4a) Drop columns that have extremely high skew

Let's find a programmatic way to detect lopsided columns

In [42]:
```
def get_cols_with_high_skew(df, identifier, percentage=0.92):
    ret = []
    for col in df.columns:
        highest_amount_category = df[col].value_counts().max()
        total = df[col].count()
        skew = highest_amount_category / total
        if skew > percentage:
            ret.append(col)
    return ret
```

In [43]:
```
cols_with_high_skew = get_cols_with_high_skew(df, "HasDectections")
```

In [44]:
```
print("Columns with high skew:")
print("COLUMN | % SKEW")
for col in cols_with_high_skew:
    print(f"{col}  |  {(df[col].value_counts().max() / df[col].count())
* 100}%")
```

```
Columns with high skew:
COLUMN | % SKEW
ProductName  |  98.964%
IsBeta  |  99.998%
RtpStateBitfield  |  97.044%
IsSxsPassiveMode  |  98.348%
AVProductsEnabled  |  96.962%
HasTpm  |  98.795999999999999%
Platform  |  96.63199999999999%
OsVer  |  96.794%
IsProtected  |  94.624%
AutoSampleOptIn  |  99.988%
SMode  |  93.801999999999999%
Firewall  |  97.85000000000001%
UacLuaenable  |  99.304%
Census_DeviceFamily  |  99.82600000000001%
Census_HasOpticalDiskDrive  |  92.15%
Census_IsPortableOperatingSystem  |  99.946%
Census_IsFlightsDisabled  |  98.182%
Census_FlightRing  |  93.594%
Census_IsVirtualDevice  |  99.11%
Census_IsPenCapable  |  96.277999999999999%
Census_IsAlwaysOnAlwaysConnectedCapable  |  93.508%
```

In [45]:
```
all_dropped += cols_with_high_skew  # Add them to our running list
```

And now we drop them

In [46]:
```
df.drop(cols_with_high_skew, axis=1, inplace=True)
```

In [47]:
```
test.drop(cols_with_high_skew, axis=1, inplace=True)
```

In [48]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 53)
(7853253, 52)
```

Step 4b) Reducing Categorical Columns Uniques

After dropping rows earlier, we have some unused categories that we can get rid of.

In [49]:
```
def print_categorical_value_counts(df):
    out = []
    for col in df.columns:
        if col != 'MachineIdentifier' and df[col].dtype.name == 'categor
y':
            out.append((df[col].value_counts().count(), col))
    out.sort(key=lambda x: x[0], reverse=True)
    return out
```

In [50]:
```
out = print_categorical_value_counts(df)
out
```

Out[50]:
```
[(8531, 'AvSigVersion'),
 (663, 'OsBuildLab'),
 (469, 'Census_OSVersion'),
 (110, 'AppVersion'),
 (70, 'EngineVersion'),
 (53, 'Census_ChassisTypeName'),
 (33, 'Census_OSEdition'),
 (32, 'Census_OSBranch'),
 (30, 'Census_OSSkuName'),
 (13, 'Census_MDC2FormFactor'),
 (10, 'Census_PowerPlatformRoleName'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (5, 'Census_GenuineStateName'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

Now we can remove no-longer used categories

In [51]:
```
for col in out:
    col_name = col[1]
    df[col_name].cat.remove_unused_categories(inplace=True)

out = print_categorical_value_counts(df)
out

# We lost a few categories which will help us with our encoding
```

Out[51]:
```
[(3098, 'AvSigVersion'),
 (312, 'OsBuildLab'),
 (234, 'Census_OSVersion'),
 (72, 'AppVersion'),
 (42, 'EngineVersion'),
 (27, 'Census_ChassisTypeName'),
 (19, 'Census_OSEdition'),
 (18, 'Census_OSSkuName'),
 (14, 'Census_OSBranch'),
 (12, 'Census_MDC2FormFactor'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (8, 'Census_PowerPlatformRoleName'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (4, 'Census_GenuineStateName'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

In [52]:
```
out = print_categorical_value_counts(test)
out
```

Out[52]:
```
[(9357, 'AvSigVersion'),
 (674, 'OsBuildLab'),
 (475, 'Census_OSVersion'),
 (120, 'AppVersion'),
 (70, 'EngineVersion'),
 (49, 'Census_ChassisTypeName'),
 (37, 'Census_OSEdition'),
 (31, 'Census_OSSkuName'),
 (29, 'Census_OSBranch'),
 (14, 'Census_MDC2FormFactor'),
 (11, 'Census_PowerPlatformRoleName'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_GenuineStateName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

In [53]:
```
for col in out:
    col_name = col[1]
    test[col_name].cat.remove_unused_categories(inplace=True)

out = print_categorical_value_counts(test)
out
```

Out[53]:
```
[(9357, 'AvSigVersion'),
 (674, 'OsBuildLab'),
 (475, 'Census_OSVersion'),
 (120, 'AppVersion'),
 (70, 'EngineVersion'),
 (49, 'Census_ChassisTypeName'),
 (37, 'Census_OSEdition'),
 (31, 'Census_OSSkuName'),
 (29, 'Census_OSBranch'),
 (14, 'Census_MDC2FormFactor'),
 (11, 'Census_PowerPlatformRoleName'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_GenuineStateName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

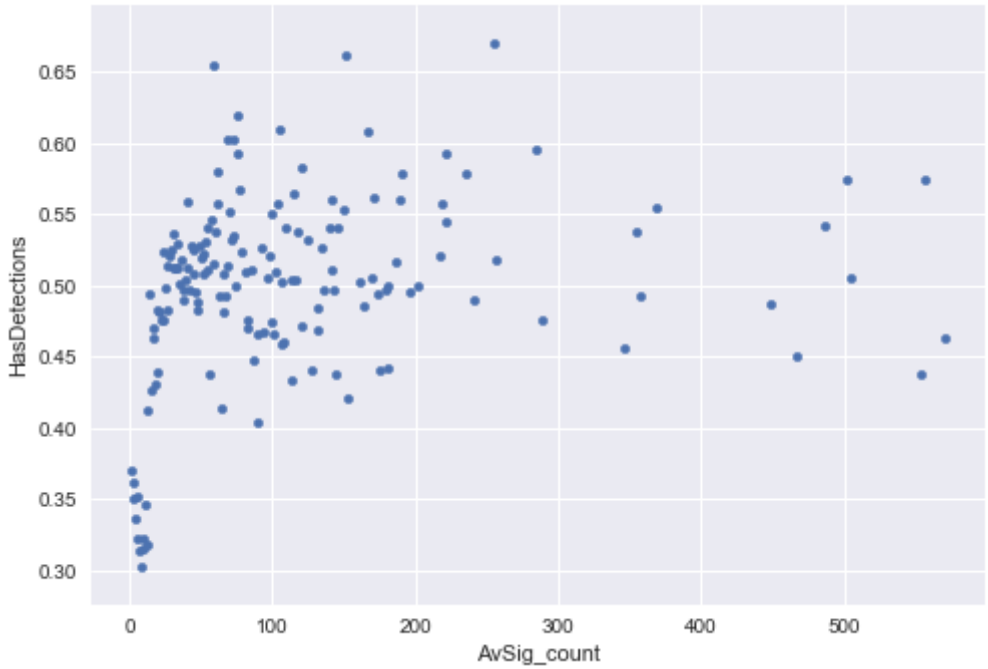
In [54]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 53)
(7853253, 52)
```

Now let's reduce the number of categories in columns with many unique categories

AvSigVersion

In [55]:
```
df['AvSig_count'] = df.groupby('AvSigVersion')['AvSigVersion'].transform
('count')
test['AvSig_count'] = test.groupby('AvSigVersion')['AvSigVersion'].trans
form('count')
pd.DataFrame(df.groupby('AvSig_count').HasDetections.mean()).reset_index
().plot(kind='scatter', x='AvSig_count', y='HasDetections');
```



No clear groupings when transformed into count, so we will leave it be.

In [56]:
```
df.drop('AvSigVersion', inplace=True, axis=1)
test.drop('AvSigVersion', inplace=True, axis=1)
```

Let's take a look at our categorical columns now:

In [57]:
```
out = print_categorical_value_counts(test)
out
```

Out[57]:
```
[(674, 'OsBuildLab'),
 (475, 'Census_OSVersion'),
 (120, 'AppVersion'),
 (70, 'EngineVersion'),
 (49, 'Census_ChassisTypeName'),
 (37, 'Census_OSEdition'),
 (31, 'Census_OSSkuName'),
 (29, 'Census_OSBranch'),
 (14, 'Census_MDC2FormFactor'),
 (11, 'Census_PowerPlatformRoleName'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_GenuineStateName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

Next up: OsBuildLab

In [58]:
```
test.OsBuildLab.value_counts()
```

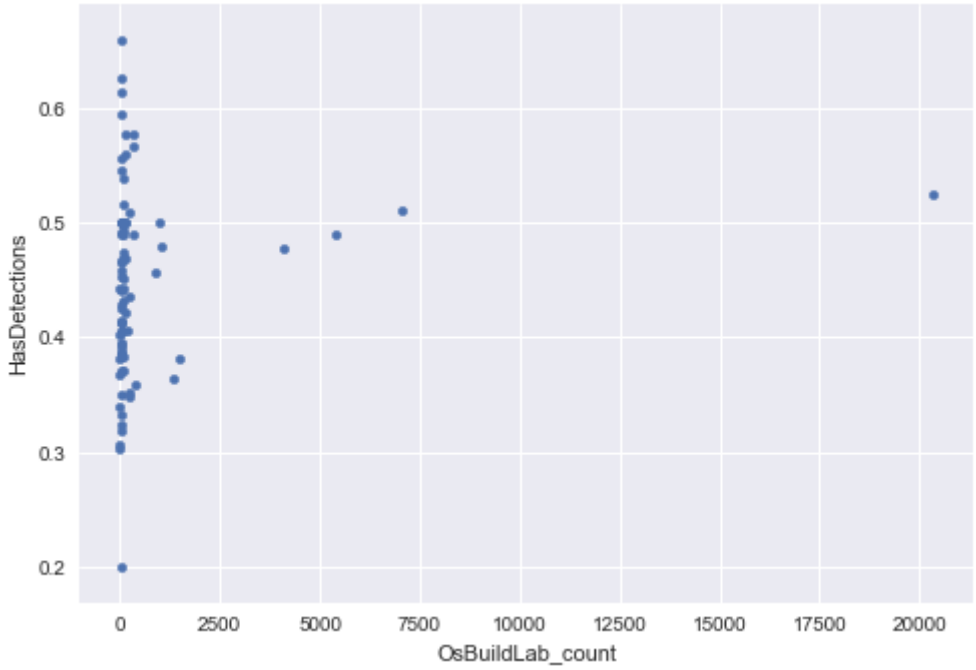
Out[58]:

17134.1.amd64fre.rs4_release.180410-1804	3628554
16299.15.amd64fre.rs3_release.170928-1534	731698
15063.0.amd64fre.rs2_release.170317-1834	570217
16299.431.amd64fre.rs3_release_svc_escrow.180502-1908	522601
16299.637.amd64fre.rs3_release_svc.180808-1748	277431
17763.1.amd64fre.rs5_release.180914-1434	266323
17134.1.x86fre.rs4_release.180410-1804	264521
14393.2189.amd64fre.rs1_release.180329-1711	163371
10240.17443.amd64fre.th1.170602-2340	162781
16299.15.x86fre.rs3_release.170928-1534	132176
10586.1176.amd64fre.th2_release_sec.170913-1848	131801
14393.0.amd64fre.rs1_release.160715-1616	57739
9600.19153.amd64fre.winblue_ltsb.180908-0600	49398
15063.0.x86fre.rs2_release.170317-1834	45868
10586.0.amd64fre.th2_release.151029-1700	32668
14393.2189.x86fre.rs1_release.180329-1711	31127
10586.1176.x86fre.th2_release_sec.170913-1848	30716
14393.2214.amd64fre.rs1_release_1.180402-1758	30351
10240.17443.x86fre.th1.170602-2340	30272
9600.19125.amd64fre.winblue_ltsb.180812-0703	27032
14393.693.amd64fre.rs1_release.161220-1747	26391
16299.637.x86fre.rs3_release_svc.180808-1748	21752
10240.16384.amd64fre.th1.150709-1700	21460
10586.162.amd64fre.th2_release_sec.160223-1728	17246
14393.447.amd64fre.rs1_release_inmarket.161102-0100	15992
17763.1.x86fre.rs5_release.180914-1434	14374
10586.672.amd64fre.th2_release_sec.161024-1825	14261
10586.494.amd64fre.th2_release_sec.160630-1736	14121
14393.2007.amd64fre.rs1_release.171231-1800	13749
14393.1593.amd64fre.rs1_release.170731-1934	13675
	...
17723.1000.x86fre.rs5_release.180720-1452	1
9600.18194.amd64fre.winblue_ltsb.160112-0600	1
18255.1000.x86fre.rs_prerelease.181003-1454	1
18219.1000.x86fre.rs_prerelease.180810-1721	1
18280.1000.amd64fre.rs_prerelease.181107-1441	1
17756.1.amd64fre.rs5_release.180905-1436	1
18287.1001.amd64fre.rs_prerelease.181117-1527	1
10240.16387.x86fre.th1_st1.150711-1429	1
7601.23338.amd64fre.win7sp1_ldr.160121-1716	1
18247.1000.x86fre.rs_prerelease.180921-1318	1
18256.1000.amd64fre.rs_prerelease.181004-1434	1
18260.1000.amd64fre.rsmaster.181010-1540	1
16024.1.amd64fre.rs5_release.180904-1423	1
7600.16988.x86fre.win7_gdr.120401-1505	1
14294.1944.amd64fre.rs1_release.171089-2100	1
7601.22656.amd64fre.win7sp1_ldr.140417-1532	1
16299.811.amd64fre.rs3_release_svc.181008-1831	1
9600.18292.x86fre.winblue_ltsb.160330-1744	1
7601.23313.x86fre.win7sp1_ldr.151230-0600	1
18248.1000.amd64fre.rs_prerelease.180924-1406	1
14421.1944.amd64fre.rs1_release.171268-2100	1
18255.1000.amd64fre.rs_onecore_dep.181003-1700	1
14218.1944.amd64fre.rs1_release.171165-2100	1
9600.17476.x86fre.winblue_r5.141029-1500	1
18287.1001.x86fre.rs_prerelease.181117-1527	1
7601.22908.x86fre.win7sp1_ldr.141211-1743	1

```
18249.1000.x86fre.rs_prerelease.180925-1413                1
10586.1177.x86fre.th2_release_inmarket.171027-1506         1
14320.1944.amd64fre.rs1_release.171244-2100                1
14216.1944.amd64fre.rs1_release.171261-2100                1
Name: OsBuildLab, Length: 674, dtype: int64
```

Let's see how OsBuildLab count correlates with HasDetections

In [59]:
```
df['OsBuildLab_count'] = df.groupby('OsBuildLab')['OsBuildLab'].transform('count')
pd.DataFrame(df.groupby('OsBuildLab_count').HasDetections.mean()).reset_index().plot(kind='scatter', x='OsBuildLab_count', y='HasDetections');
```



You've served your purpose, OsBuildLab_count. Now we will drop you.

In [60]:
```
df.drop('OsBuildLab_count', axis=1, inplace=True)
```

The grouping indicates 4 popular builds and a bunch of others. Let's split them into categories

In [61]:
```
def transform_OsBuildLab(x):
    if (df.OsBuildLab == x).sum() / (df.OsBuildLab == x).count() < .05:
        return 'other'
    else:
        return x
```

In [62]:
```
df['OsBuildLab_encoded'] = df.OsBuildLab.apply(transform_OsBuildLab)
test['OsBuildLab_encoded'] = test.OsBuildLab.apply(transform_OsBuildLab)
```

In [63]:
```
test.OsBuildLab_encoded.value_counts()
```

Out[63]:
```
17134.1.amd64fre.rs4_release.180410-1804                3628554
other                                                   2400183
16299.15.amd64fre.rs3_release.170928-1534                731698
15063.0.amd64fre.rs2_release.170317-1834                 570217
16299.431.amd64fre.rs3_release_svc_escrow.180502-1908    522601
Name: OsBuildLab_encoded, dtype: int64
```

In [64]:
```
df['OsBuildLab_encoded'] = df['OsBuildLab_encoded'].astype('category')
test['OsBuildLab_encoded'] = test['OsBuildLab_encoded'].astype('categor
y')
```

In [65]:
```
df.drop('OsBuildLab', axis=1, inplace=True)
test.drop('OsBuildLab', axis=1, inplace=True)
```

In [66]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 53)
(7853253, 52)
```

In [67]:
```
print_categorical_value_counts(test)
```

Out[67]:
```
[(475, 'Census_OSVersion'),
 (120, 'AppVersion'),
 (70, 'EngineVersion'),
 (49, 'Census_ChassisTypeName'),
 (37, 'Census_OSEdition'),
 (31, 'Census_OSSkuName'),
 (29, 'Census_OSBranch'),
 (14, 'Census_MDC2FormFactor'),
 (11, 'Census_PowerPlatformRoleName'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_GenuineStateName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (5, 'OsBuildLab_encoded'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

Next up: Census OSVersion

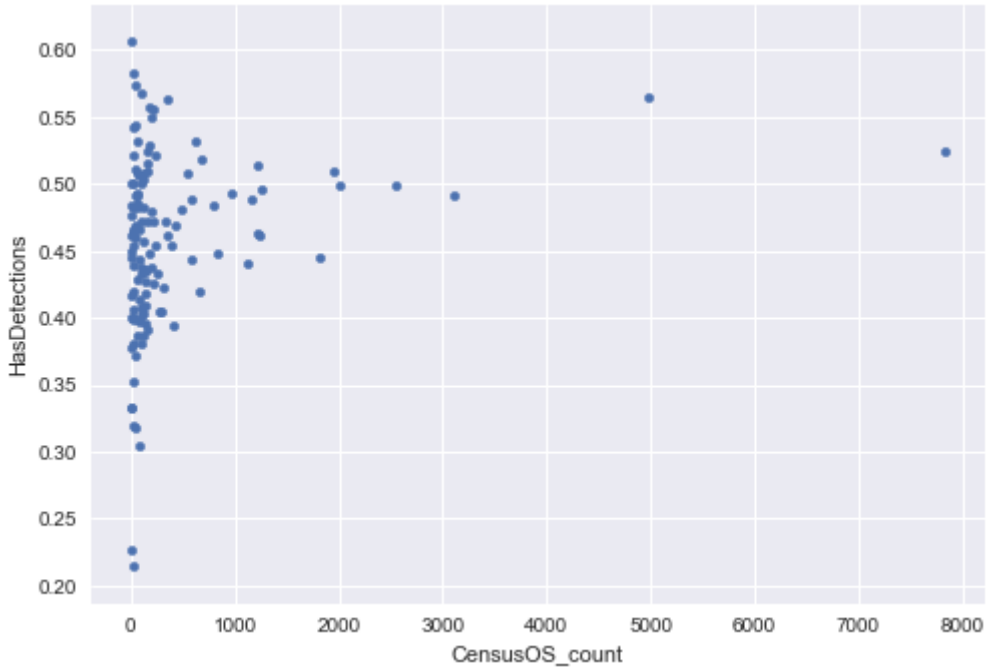
In [68]:
```
test.Census_OSVersion.value_counts()
```

Out[68]:

10.0.17134.345	1377565
10.0.17134.285	669674
10.0.17134.407	520122
10.0.17134.286	365793
10.0.16299.431	283978
10.0.17134.112	227059
10.0.10240.17443	195916
10.0.16299.371	195793
10.0.14393.2189	191389
10.0.10586.1176	164920
10.0.17134.376	152040
10.0.17134.228	151840
10.0.16299.125	146484
10.0.17134.320	131449
10.0.16299.726	126869
10.0.16299.15	123264
10.0.17134.1	119260
10.0.17763.55	117337
10.0.15063.1387	113180
10.0.16299.309	96775
10.0.17763.134	93335
10.0.16299.611	87497
10.0.16299.547	86075
10.0.16299.492	78985
10.0.15063.1324	75775
10.0.16299.192	71521
10.0.15063.0	68356
10.0.16299.665	63449
10.0.17134.165	57901
10.0.14393.0	56949
...	
10.0.9600.318	1
10.0.14393.1794	1
10.0.14246.1	1
6.3.9600.19182	1
10.0.18204.1001	1
10.0.17730.1000	1
10.0.18247.1000	1
10.0.14257.1000	1
10.0.16299.428	1
10.0.16251.0	1
10.0.15060.0	1
10.0.16299.188	1
10.0.14942.1000	1
10.0.18256.1000	1
10.0.15058.0	1
10.0.14342.1002	1
10.0.10586.240	1
10.0.14376.0	1
10.0.15063.1443	1
6.1.7601.23539	1
10.0.17035.1000	1
10.0.14294.286	1
10.0.16299.811	1
10.0.18248.1000	1
10.0.14332.1001	1
10.0.14279.1000	1

```
10.0.14421.345            1
10.0.14218.376            1
10.0.14320.1              1
10.0.18219.1000           1
Name: Census_OSVersion, Length: 475, dtype: int64
```

In [69]:
```
df['CensusOS_count'] = df.groupby('Census_OSVersion')['Census_OSVersion'
].transform('count')
pd.DataFrame(df.groupby('CensusOS_count').HasDetections.mean()).reset_in
dex().plot(kind='scatter', x='CensusOS_count', y='HasDetections');
```



In [70]:
```
df.drop('CensusOS_count', axis=1, inplace=True)
```

In [71]:
```
def transform_CensusOS(x):
    if (df.Census_OSVersion == x).sum() / (df.Census_OSVersion == x).cou
nt() < .03:
        return 'other'
    else:
        return x
```

In [72]:
```
df['CensusOS_encoded'] = df.Census_OSVersion.apply(transform_CensusOS)
test['CensusOS_encoded'] = test.Census_OSVersion.apply(transform_CensusO
S)
```

In [73]:
```
test.CensusOS_encoded.value_counts()
```

Out[73]:
```
other             6180933
10.0.17134.285     669674
10.0.16299.431     283978
10.0.17134.112     227059
10.0.16299.371     195793
10.0.17134.228     151840
10.0.16299.547      86075
10.0.17134.165      57901
Name: CensusOS_encoded, dtype: int64
```

In [74]:
```
df['CensusOS_encoded'] = df['CensusOS_encoded'].astype('category')
test['CensusOS_encoded'] = test['CensusOS_encoded'].astype('category')
```

In [75]:
```
df.drop('Census_OSVersion', axis=1, inplace=True)
test.drop('Census_OSVersion', axis=1, inplace=True)
```

In [76]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 53)
(7853253, 52)
```

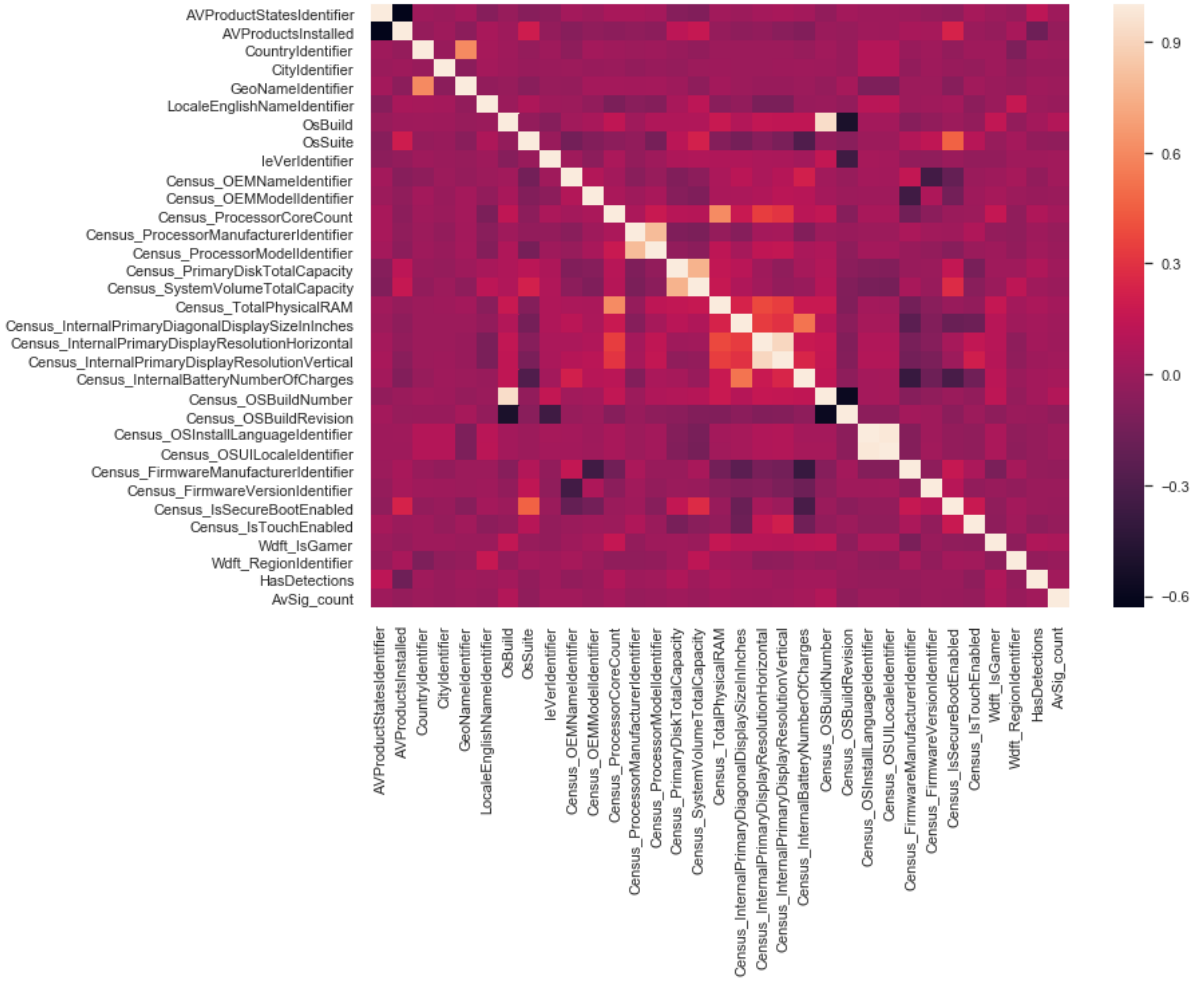
In [77]:
```
print_categorical_value_counts(test)
```

Out[77]:
```
[(120, 'AppVersion'),
 (70, 'EngineVersion'),
 (49, 'Census_ChassisTypeName'),
 (37, 'Census_OSEdition'),
 (31, 'Census_OSSkuName'),
 (29, 'Census_OSBranch'),
 (14, 'Census_MDC2FormFactor'),
 (11, 'Census_PowerPlatformRoleName'),
 (9, 'OsPlatformSubRelease'),
 (9, 'Census_OSInstallTypeName'),
 (8, 'SkuEdition'),
 (8, 'CensusOS_encoded'),
 (6, 'Census_OSWUAutoUpdateOptionsName'),
 (6, 'Census_GenuineStateName'),
 (6, 'Census_ActivationChannel'),
 (5, 'Census_PrimaryDiskTypeName'),
 (5, 'OsBuildLab_encoded'),
 (3, 'Processor'),
 (3, 'Census_OSArchitecture')]
```

120 unique categories is acceptable. Let's move on!

4c) Find Correlated Columns and Drop the Redundant Columns

In [78]:
```
# Let's do this on a smaller sample

df_sample = df.sample(10000)
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.heatmap(df_sample.corr());
```



In [79]:
```
cols_with_correlations = ['AVProductsInstalled', 'Census_OSBuildRevisio
n', 'Census_OSBuildNumber',
            'Census_FirmwareManufacturerIdentifier', 'Census_OEMNameIdent
ifier',
            'GeoNameIdentifier', 'Census_ProcessorModelIdentifier',
            'Census_InternalPrimaryDisplayResolutionVertical',
            'Census_OSUILocaleIdentifier', 'Census_TotalPhysicalRAM']

all_dropped += cols_with_correlations
```

In [80]:
```
df.drop(cols_with_correlations, axis=1, inplace=True)
```

In [81]:
```
test.drop(cols_with_correlations, axis=1, inplace=True)
```

In [82]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 43)
(7853253, 42)
```

4d) Transform Continuous Columns

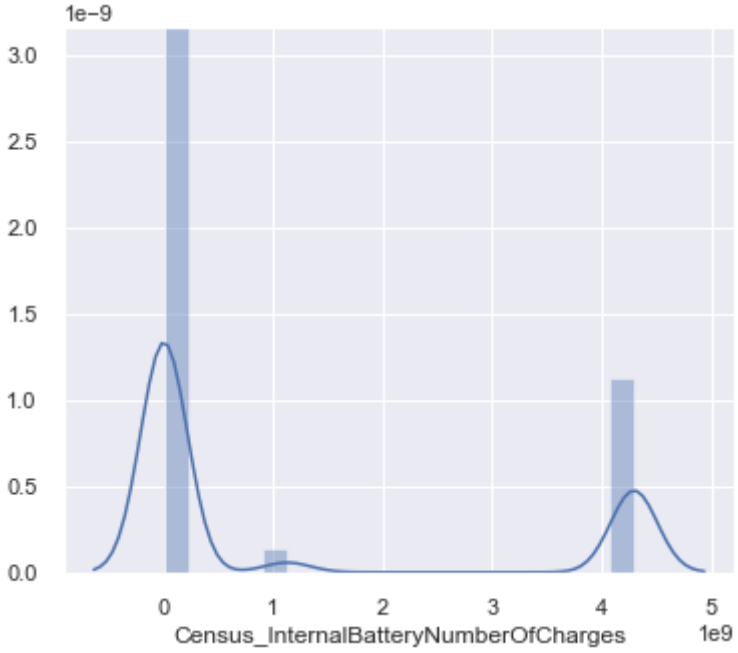
Let's define some helper functions

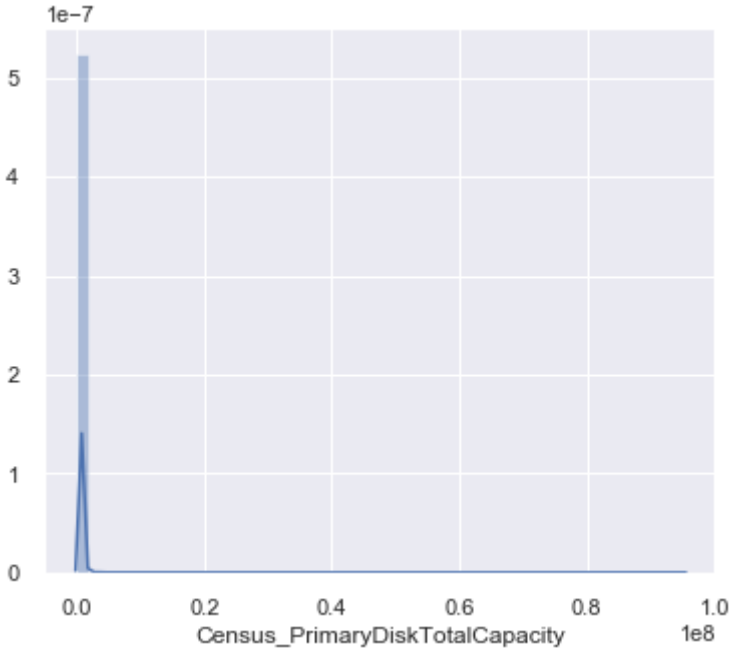
In [85]:
```
def report_ranges(df):
    """
    Gathers the ranges of numeric data in sorted order

    Parameters: Dataframe
    Returns: A list of tuples in the form of (range, column name)
    """
    out = []
    for col in df.columns:
        if not df[col].dtype.name in ['category', 'bool', 'object']:
            out.append((df[col].max() - df[col].min(), col))
    out.sort(key=lambda x: x[0], reverse=True)
    return out


def discretize(df, col_name, labels, new_name, quantiles=True):
    """
    Turns numeric data into categories, using pd.cut or pd.qcut, makes a
new column, drop the old one.
    Prints out the value_count of the new column

    Parameters: Dataframe, column name, list of labels, new column name,
quantile boolean
    Returns: None
    """
    bins = len(labels)
    if quantiles:
        df[new_name] = pd.qcut(df[col_name], bins, labels=labels, duplic
ates='drop')
    else:
        df[new_name] = pd.cut(df[col_name], bins)
    df.drop(col_name, axis=1, inplace=True)


def log_transform(df, col_name):
    """
    Applies a log1p transformation to a column.
    Plots the new data

    Parameters: Dataframe, column name
    Returns: None
    """
    df[col_name] = np.log1p(df[col_name])


def listify(n):
    """
    Provides a list from user inputs

    Parameters: n = number of items in list
    Returns: list of user inputs
    """
    lst = []
    for i in range(n):
        item = input("Enter item: ")
        lst.append(item)
```

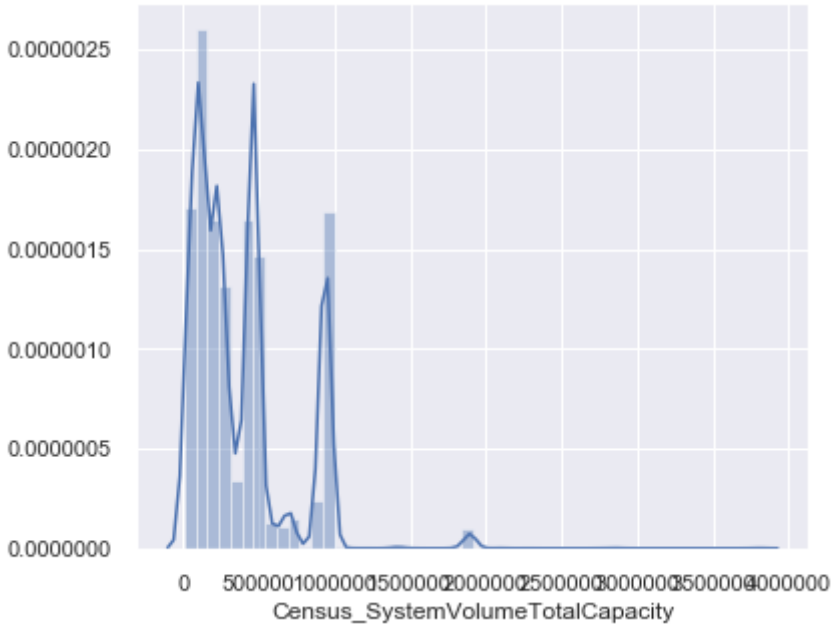
```
    return lst


def make_decisions_graphs(df):
    """
    Displays graphs of numerical data and allows user to make decisions
 on data

    Parameters: Dataframe
    Returns: Multidimensional Array with decision information
    """
    ranges = report_ranges(df)
    decisions = []

    for col in ranges:
        col_name = col[1]
        choice = [col_name]
        sns.set(rc={'figure.figsize': (6, 5)})
        sns.distplot(df[col_name])
        plt.show();

        func_to_call = int(input("""What would you like to do with this
 data?
        1. Discretize
        2. Log Transform
        3. Drop
        4. Nothing
        """))

        choice.append(func_to_call)

        if func_to_call == 1:
            type_cut = int(input("""What type of cut?
            1. cut
            2. qcut
            """))
            if type_cut == 1:
                quantiles = False
            else:
                quantiles = True

            n = int(input("How many labels will you use? "))
            labels = listify(n)

            choice += [labels]
            choice.append(col_name + "_CAT")
            choice.append(quantiles)

        decisions.append(choice)
    return (decisions)


def do_decisions(df, decisions):
    """ Applies the decisions from the make_decisions_graph function"

    Parameters: Dataframe, list of decisions from make_decisions_graph f
unction
```

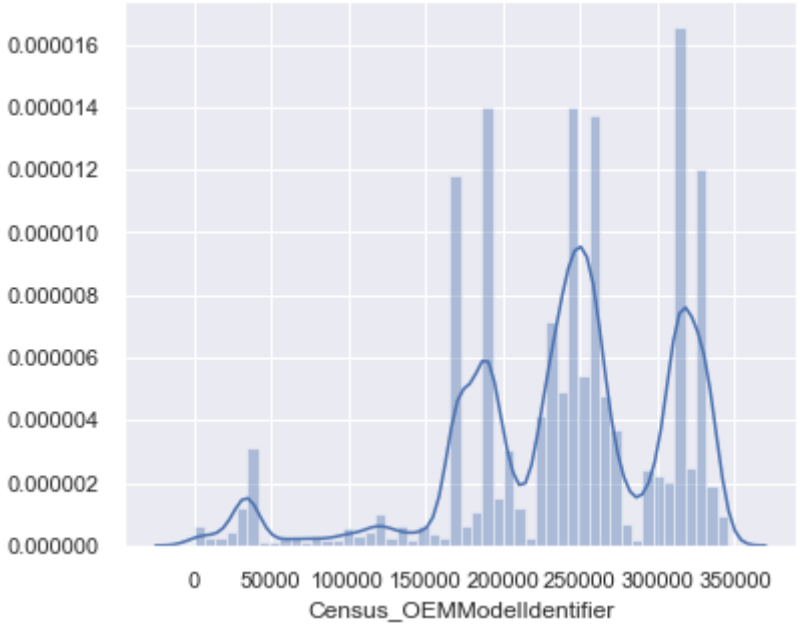
```
    Returns: None
    """
    for choice in decisions:
        col_name = choice[0]
        func_to_call = choice[1]

        if func_to_call == 1:
            discretize(df, col_name, choice[2], choice[3], quantiles=cho
ice[4])
        elif func_to_call == 2:
            log_transform(df, col_name)
        elif func_to_call == 3:
            df.drop([col_name], axis=1, inplace=True)
```

In [86]:
```
decisions = make_decisions_graphs(df)
```

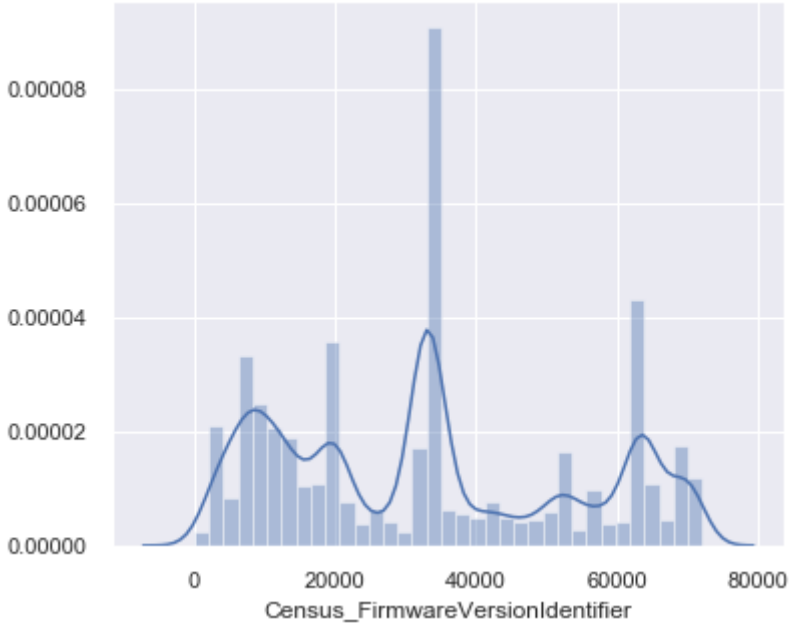

1e−9
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
Census_InternalBatteryNumberOfCharges
1e9

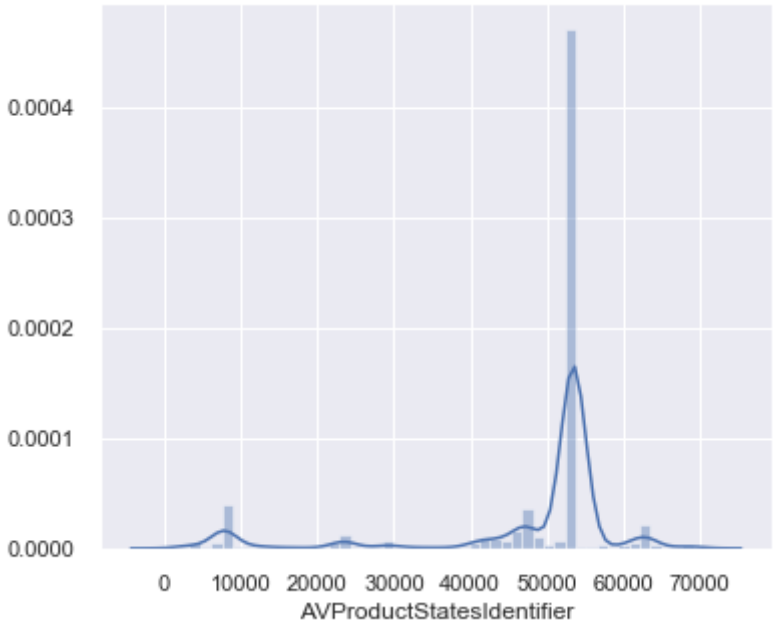

1e−7
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
Census_PrimaryDiskTotalCapacity
1e8

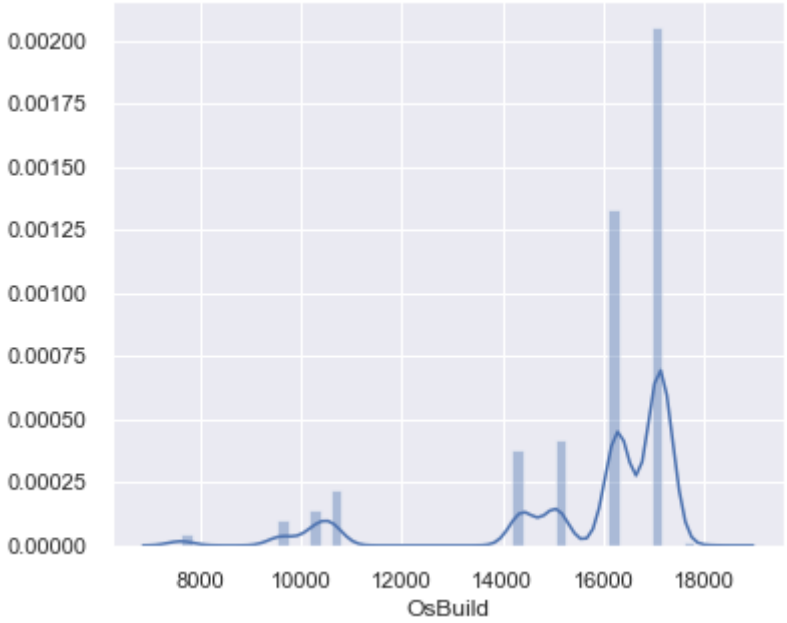

0.0000025
0.0000020
0.0000015
0.0000010
0.0000005
0.0000000
0
Census_SystemVolumeTotalCapacity

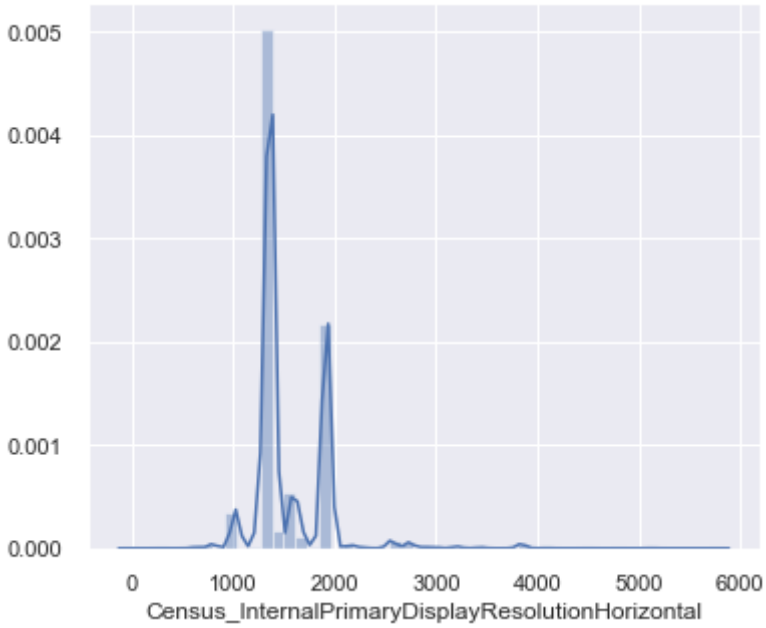

0.000016
0.000014
0.000012
0.000010
0.000008
0.000006
0.000004
0.000002
0.000000
0
50000
100000
150000
200000
250000
300000
350000
Census_OEMModelIdentifier

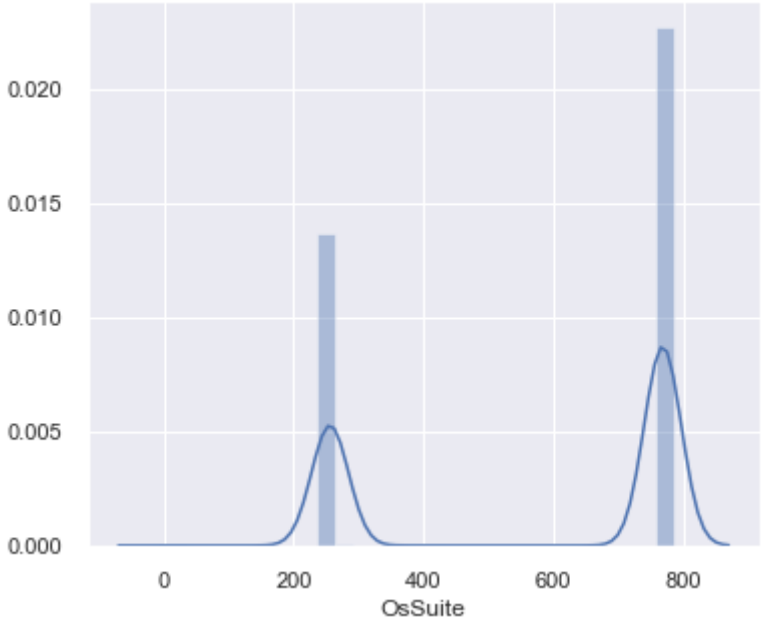

0.000014
0.000012
0.000010
0.000008
0.000006
0.000004
0.000002
0.000000
−25000
0
25000
50000
75000
100000
125000
150000
175000
CityIdentifier

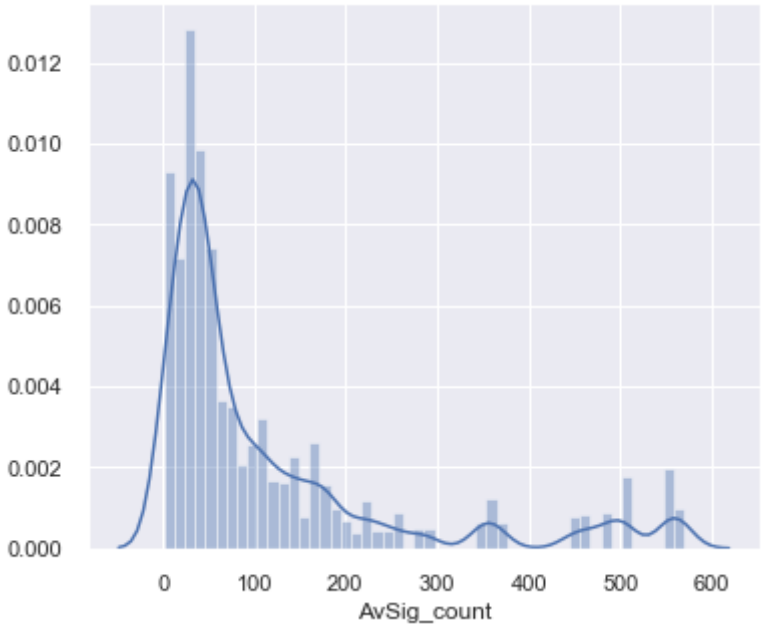

0.00008
0.00006
0.00004
0.00002
0.00000
0
20000
40000
60000
80000
Census_FirmwareVersionIdentifier

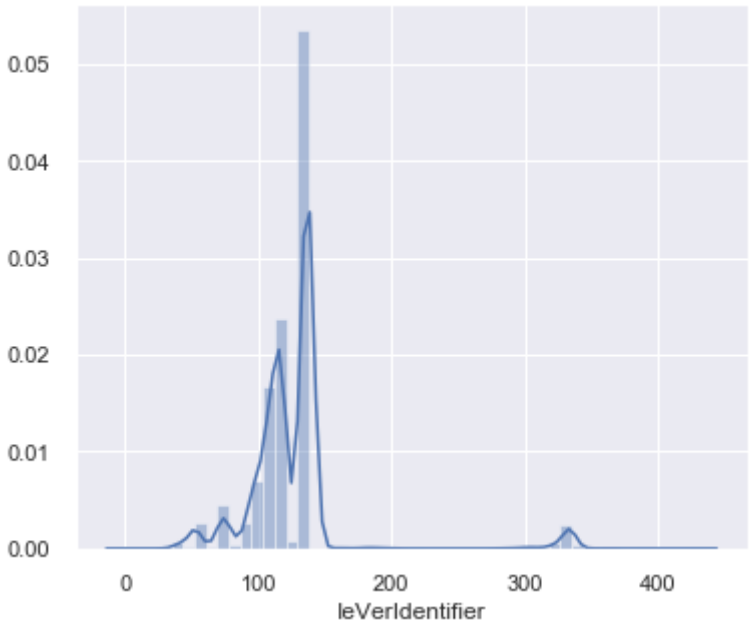

0.0004
0.0003
0.0002
0.0001
0.0000
0
10000
20000
30000
40000
50000
60000
70000
AVProductStatesIdentifier

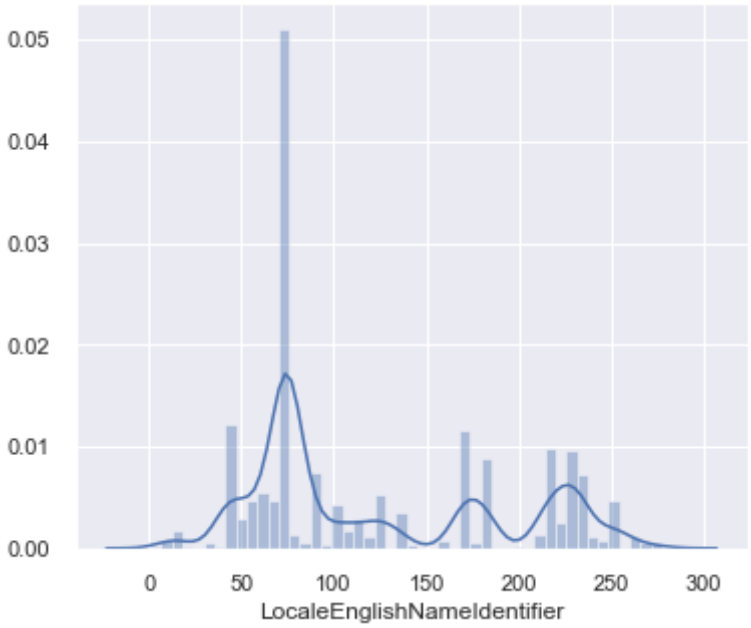

0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
8000
10000
12000
14000
16000
18000
OsBuild

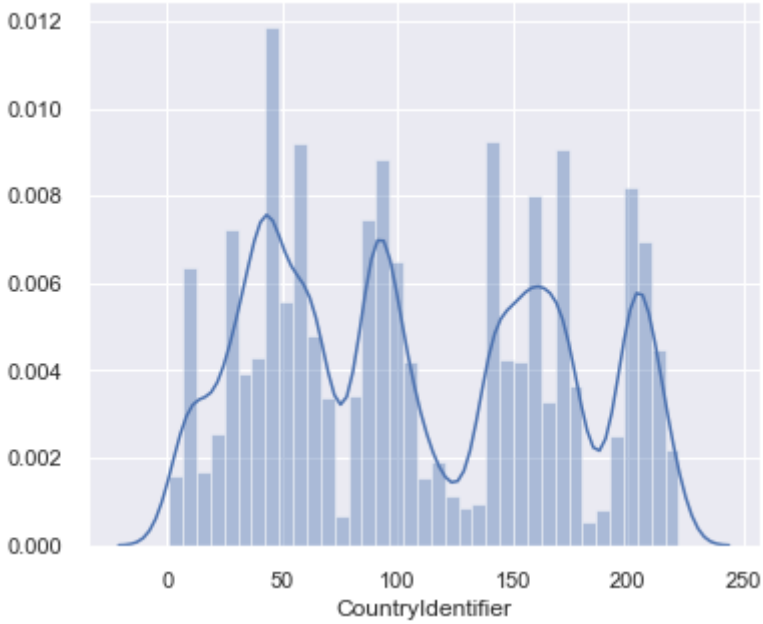

0.005
0.004
0.003
0.002
0.001
0.000
0
1000
2000
3000
4000
5000
6000
Census_InternalPrimaryDisplayResolutionHorizontal

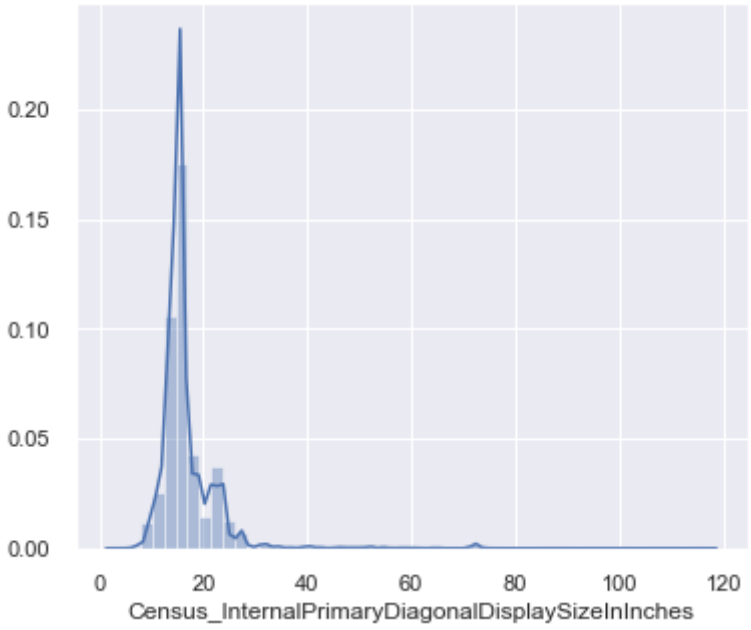

0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
OsSuite

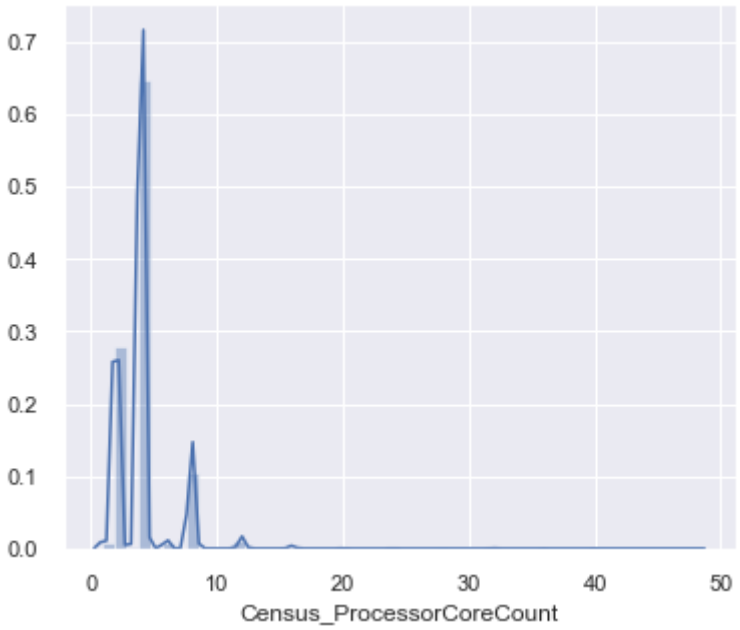

0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
100
200
300
400
500
600
AvSig_count

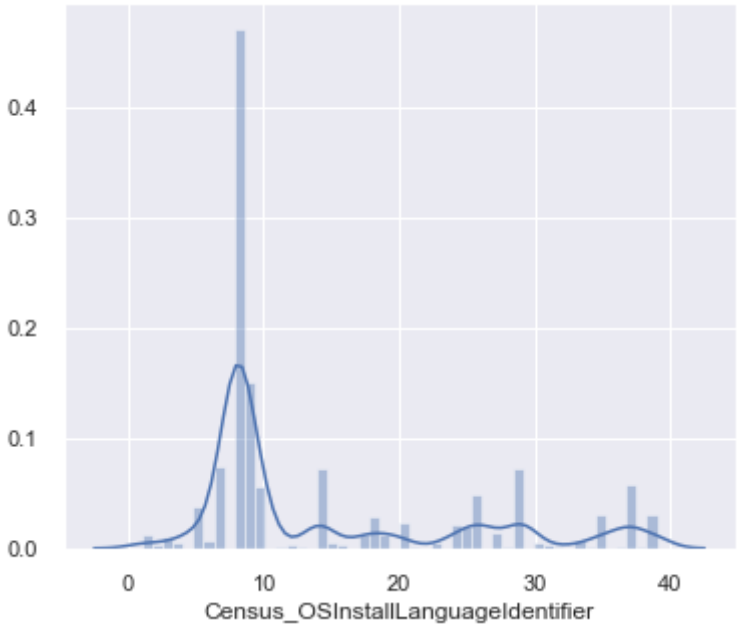

0.05
0.04
0.03
0.02
0.01
0.00
0
100
200
300
400
leVerIdentifier

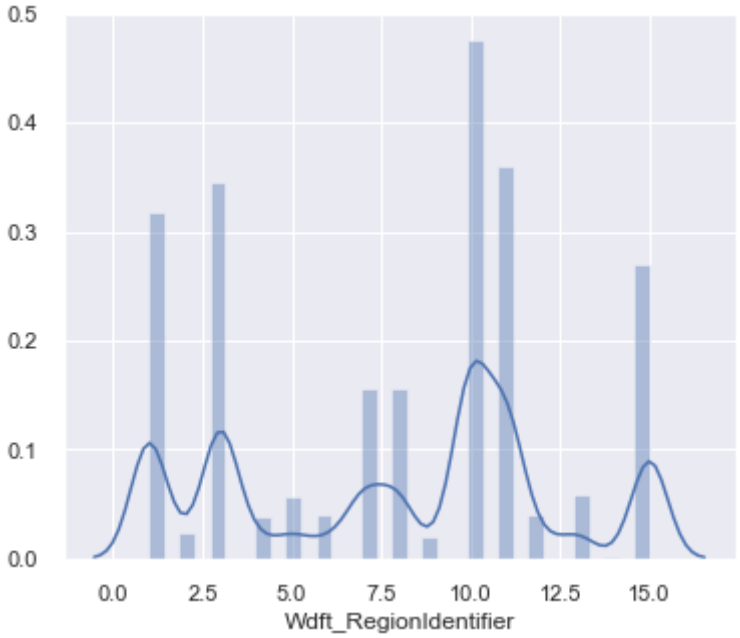

0.05
0.04
0.03
0.02
0.01
0.00
0
50
100
150
200
250
300
LocaleEnglishNameIdentifier

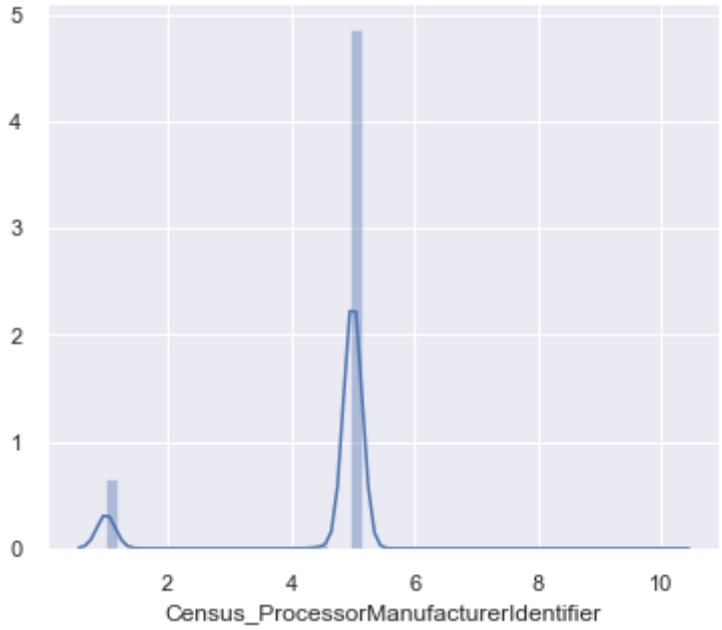

0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
50
100
150
200
250
CountryIdentifier


0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
Census_InternalPrimaryDiagonalDisplaySizeInInches


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
10
20
30
40
50
Census_ProcessorCoreCount


0.4
0.3
0.2
0.1
0.0
0
10
20
30
40
Census_OSInstallLanguageIdentifier


0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
Wdft_RegionIdentifier



In [87]:
```
do_decisions(df, decisions)
```

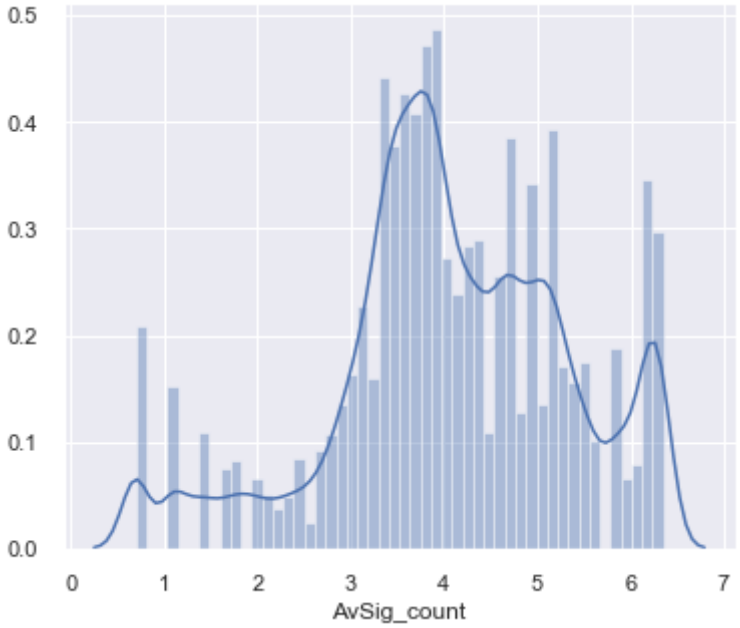
In [88]:
```
do_decisions(test, decisions)
```

In [89]:
```
print(df.shape)
print(test.shape)
```

```
(50000, 42)
(7853253, 41)
```

New distplot for a log-transformed column:

In [90]:
```
sns.distplot(df['AvSig_count']);
```



Step 5) One Hot Encoding

Concatenate the train/test dataframes so that the number of columns is the same after one hot encoding

In [108]:
```
df.reset_index(drop=True, inplace=True)
```

In [109]:
```
test_copy = test.copy()
df_copy = df.copy()
target = df_copy.HasDetections
df_copy.drop('HasDetections', axis=1, inplace=True)
bigdata = df_copy.append(test_copy, ignore_index=True)
```

In [110]:
```
bigdata.shape
```

Out[110]:
```
(7903253, 41)
```

In [111]:
```
bigdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7903253 entries, 0 to 7903252
Data columns (total 41 columns):
MachineIdentifier                                     object
EngineVersion                                         object
AppVersion                                            object
AVProductStatesIdentifier                             float32
CountryIdentifier                                     uint8
CityIdentifier                                        float32
LocaleEnglishNameIdentifier                           float32
Processor                                             category
OsBuild                                               float32
OsPlatformSubRelease                                  category
SkuEdition                                            category
IeVerIdentifier                                       float32
Census_MDC2FormFactor                                 object
Census_ProcessorCoreCount                             float32
Census_ProcessorManufacturerIdentifier                float32
Census_PrimaryDiskTypeName                            category
Census_SystemVolumeTotalCapacity                      float32
Census_ChassisTypeName                                object
Census_InternalPrimaryDiagonalDisplaySizeInInches     float32
Census_InternalPrimaryDisplayResolutionHorizontal     float32
Census_PowerPlatformRoleName                          object
Census_OSArchitecture                                 category
Census_OSBranch                                       object
Census_OSEdition                                      object
Census_OSSkuName                                      object
Census_OSInstallTypeName                              category
Census_OSInstallLanguageIdentifier                    float32
Census_OSWUAutoUpdateOptionsName                      category
Census_GenuineStateName                               object
Census_ActivationChannel                              category
Census_FirmwareVersionIdentifier                      float32
Census_IsSecureBootEnabled                            bool
Census_IsTouchEnabled                                 bool
Wdft_IsGamer                                          bool
Wdft_RegionIdentifier                                 float32
AvSig_count                                           float64
OsBuildLab_encoded                                    category
CensusOS_encoded                                      category
Census_InternalBatteryNumberOfCharges_CAT             category
Census_OEMModelIdentifier_CAT                         object
OsSuite_CAT                                           category
dtypes: bool(3), category(12), float32(13), float64(1), object(11), uin
t8(1)
memory usage: 1.2+ GB
```

In [112]:
```
test_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7853253 entries, 0 to 7853252
Data columns (total 41 columns):
MachineIdentifier                                     object
EngineVersion                                         category
AppVersion                                            category
AVProductStatesIdentifier                             float32
CountryIdentifier                                     uint8
CityIdentifier                                        float32
LocaleEnglishNameIdentifier                           float32
Processor                                             category
OsBuild                                               float32
OsPlatformSubRelease                                  category
SkuEdition                                            category
IeVerIdentifier                                       float32
Census_MDC2FormFactor                                 category
Census_ProcessorCoreCount                             float32
Census_ProcessorManufacturerIdentifier                float32
Census_PrimaryDiskTypeName                            category
Census_SystemVolumeTotalCapacity                      float32
Census_ChassisTypeName                                category
Census_InternalPrimaryDiagonalDisplaySizeInInches     float32
Census_InternalPrimaryDisplayResolutionHorizontal     float32
Census_PowerPlatformRoleName                          category
Census_OSArchitecture                                 category
Census_OSBranch                                       category
Census_OSEdition                                      category
Census_OSSkuName                                      category
Census_OSInstallTypeName                              category
Census_OSInstallLanguageIdentifier                    float32
Census_OSWUAutoUpdateOptionsName                      category
Census_GenuineStateName                               category
Census_ActivationChannel                              category
Census_FirmwareVersionIdentifier                      float32
Census_IsSecureBootEnabled                            bool
Census_IsTouchEnabled                                 bool
Wdft_IsGamer                                          bool
Wdft_RegionIdentifier                                 float32
AvSig_count                                           float64
OsBuildLab_encoded                                    category
CensusOS_encoded                                      category
Census_InternalBatteryNumberOfCharges_CAT             category
Census_OEMModelIdentifier_CAT                         category
OsSuite_CAT                                           category
dtypes: bool(3), category(22), float32(13), float64(1), object(1), uint
8(1)
memory usage: 704.0+ MB
```

In [113]:
```
dtypes = test_copy.dtypes.apply(lambda x: x.name).to_dict()
```

In [114]:
```
bigdata = bigdata.astype(dtype=dtypes)
```

In [115]:
```
bigdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7903253 entries, 0 to 7903252
Data columns (total 41 columns):
MachineIdentifier                                     object
EngineVersion                                         category
AppVersion                                            category
AVProductStatesIdentifier                             float32
CountryIdentifier                                     uint8
CityIdentifier                                        float32
LocaleEnglishNameIdentifier                           float32
Processor                                             category
OsBuild                                               float32
OsPlatformSubRelease                                  category
SkuEdition                                            category
IeVerIdentifier                                       float32
Census_MDC2FormFactor                                 category
Census_ProcessorCoreCount                             float32
Census_ProcessorManufacturerIdentifier                float32
Census_PrimaryDiskTypeName                            category
Census_SystemVolumeTotalCapacity                      float32
Census_ChassisTypeName                                category
Census_InternalPrimaryDiagonalDisplaySizeInInches     float32
Census_InternalPrimaryDisplayResolutionHorizontal     float32
Census_PowerPlatformRoleName                          category
Census_OSArchitecture                                 category
Census_OSBranch                                       category
Census_OSEdition                                      category
Census_OSSkuName                                      category
Census_OSInstallTypeName                              category
Census_OSInstallLanguageIdentifier                    float32
Census_OSWUAutoUpdateOptionsName                      category
Census_GenuineStateName                               category
Census_ActivationChannel                              category
Census_FirmwareVersionIdentifier                      float32
Census_IsSecureBootEnabled                            bool
Census_IsTouchEnabled                                 bool
Wdft_IsGamer                                          bool
Wdft_RegionIdentifier                                 float32
AvSig_count                                           float64
OsBuildLab_encoded                                    category
CensusOS_encoded                                      category
Census_InternalBatteryNumberOfCharges_CAT             category
Census_OEMModelIdentifier_CAT                         category
OsSuite_CAT                                           category
dtypes: bool(3), category(22), float32(13), float64(1), object(1), uint
8(1)
memory usage: 708.5+ MB
```

In [116]:
```
categoricals = []
for col in bigdata.columns:
    if bigdata[col].dtype.name == 'category':
        categoricals.append(col)

categoricals
```

Out[116]:
```
['EngineVersion',
 'AppVersion',
 'Processor',
 'OsPlatformSubRelease',
 'SkuEdition',
 'Census_MDC2FormFactor',
 'Census_PrimaryDiskTypeName',
 'Census_ChassisTypeName',
 'Census_PowerPlatformRoleName',
 'Census_OSArchitecture',
 'Census_OSBranch',
 'Census_OSEdition',
 'Census_OSSkuName',
 'Census_OSInstallTypeName',
 'Census_OSWUAutoUpdateOptionsName',
 'Census_GenuineStateName',
 'Census_ActivationChannel',
 'OsBuildLab_encoded',
 'CensusOS_encoded',
 'Census_InternalBatteryNumberOfCharges_CAT',
 'Census_OEMModelIdentifier_CAT',
 'OsSuite_CAT']
```

In [117]:
```
bigdata = pd.get_dummies(bigdata, prefix=categoricals, columns=categoricals)
```

In [118]:
```
train = bigdata.iloc[:50000]
test = bigdata.iloc[50000:]
```

In [119]:
```
train.tail()
```

Out[119]:

	MachineIdentifier	AVProductStatesIdentifier	CountryIdentifier	CityIdentif
49995	03fa9046d62636e23c0df9b45ad0a153	10.690308	160	16561
49996	1659c5414e4c2ad7f0eef4c7d03ff94b	8.980424	203	14378
49997	7b6098fb38fd2f44dfab8e85b101c709	10.886464	43	1041
49998	4e8bc325501c976bcbd50c6c5b12808b	10.762975	147	7779
49999	fb0885c7f92835a13d2b9f938cd1d4f7	8.980424	118	7067

In [120]:
```
print(train.shape)
print(test.shape)
print(target.shape)
```

```
(50000, 462)
(7853253, 462)
(50000,)
```

Step 6) Modeling

In [121]:
```
train_id = train.MachineIdentifier
train.drop('MachineIdentifier', axis=1, inplace=True)
test_id = test.MachineIdentifier
test.drop('MachineIdentifier', axis=1, inplace=True)
```

Side step: downcast float64 'AvSigValue' and drop a column with infinite values

In [125]:
```
train['AvSig_count'] = pd.to_numeric(train['AvSig_count'], downcast='flo
at')
test['AvSig_count'] = pd.to_numeric(test['AvSig_count'], downcast='floa
t')
```

In [130]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Columns: 461 entries, AVProductStatesIdentifier to OsSuite_CAT_(400.0,
784.0]
dtypes: bool(3), float32(14), uint8(444)
memory usage: 24.0 MB
```

In [137]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Columns: 461 entries, AVProductStatesIdentifier to OsSuite_CAT_(400.0,
784.0]
dtypes: bool(3), float32(14), uint8(444)
memory usage: 24.0 MB
```

In [139]:
```
for col in train.columns:
    if train[col].dtype.name not in ['category', 'bool']:
        if not np.all(np.isfinite(train[col])):
            print(col)
```

```
Census_InternalPrimaryDisplayResolutionHorizontal
```

In [141]:
```
for col in test.columns:
    if test[col].dtype.name not in ['category', 'bool']:
        if not np.all(np.isfinite(test[col])):
            print(col)
```

```
Census_InternalPrimaryDisplayResolutionHorizontal
```

In [142]:
```
train.drop('Census_InternalPrimaryDisplayResolutionHorizontal', axis=1,
inplace=True)
test.drop('Census_InternalPrimaryDisplayResolutionHorizontal', axis=1, i
nplace=True)
```

Modeling time.

Logistic Regression: 0.50748

In [145]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr_model = lr.fit(train, target)
```

In [146]:
```
lr_model.score(train, target)
```

Out[146]:
```
0.50748
```

KNN: 0.75634

In [149]:
```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier(n_neighbors=2)
kn_model = kn.fit(train, target)
kn_model.score(train, target)
```

Out[149]:
```
0.75634
```

What does y hat look like?

In [151]:
```
yhat_p = kn_model.predict_proba(test)
```

In [155]:
```
yhat_p[:100][:,1]
```

Out[155]:
```
array([0.5, 0. , 0. , 0.5, 0. , 0.5, 0. , 0.5, 0.5, 0. , 0.5, 0.5, 0.5,
       1. , 0.5, 1. , 0.5, 0. , 0. , 0.5, 0.5, 0. , 1. , 0. , 0. , 0.5,
       1. , 0.5, 0.5, 0.5, 0. , 0.5, 0.5, 1. , 1. , 1. , 0. , 0.5, 0.5,
       0.5, 0. , 1. , 0.5, 0. , 1. , 0.5, 1. , 1. , 1. , 0.5, 0. , 1. ,
       0.5, 0.5, 0.5, 0. , 1. , 0. , 0.5, 0.5, 1. , 0. , 0. , 0.5, 0.5,
       0.5, 0. , 1. , 1. , 1. , 0. , 1. , 0.5, 1. , 0. , 1. , 0. , 1. ,
       0.5, 1. , 1. , 0. , 0. , 0. , 0. , 0.5, 0. , 0.5, 0. , 0.5, 0. ,
       0.5, 1. , 0. , 0.5, 1. , 0. , 0.5, 0.5, 0.5])
```

This makes sense given that we are using a KNN classifier.

In [156]:
```
yhat_pLR = lr_model.predict_proba(test[:750])
```

In [160]:
```
yhat_pLR[:100][:,1]
```

Out[160]:
```
array([0.49639099, 0.49360778, 0.49120099, 0.4917872 , 0.48835651,
       0.49354924, 0.4899292 , 0.49959812, 0.49044846, 0.4940216 ,
       0.49811449, 0.4879621 , 0.49457564, 0.49660045, 0.48969212,
       0.4953022 , 0.49390005, 0.48946384, 0.49509917, 0.49736754,
       0.48916172, 0.4932928 , 0.49093448, 0.48921514, 0.491059  ,
       0.49573462, 0.49127622, 0.49505696, 0.49148926, 0.48915978,
       0.49357764, 0.48971026, 0.49211826, 0.49588643, 0.49282218,
       0.4972599 , 0.49418712, 0.48916266, 0.49294626, 0.49077605,
       0.49300624, 0.49752708, 0.49095558, 0.49156499, 0.49339805,
       0.49041013, 0.48922233, 0.49178576, 0.49748925, 0.49072585,
       0.48912833, 0.49799595, 0.49276281, 0.49767362, 0.495861  ,
       0.49045469, 0.49252057, 0.49573523, 0.48979369, 0.49512637,
       0.49598284, 0.49529429, 0.49026448, 0.48706292, 0.49577878,
       0.49752143, 0.49728024, 0.4899769 , 0.49302145, 0.4915689 ,
       0.49386021, 0.49573292, 0.4957324 , 0.49281351, 0.4918573 ,
       0.49261647, 0.49458184, 0.49496429, 0.49204408, 0.49282889,
       0.49211336, 0.49881865, 0.49673849, 0.49394282, 0.49529991,
       0.49106762, 0.49495533, 0.49038335, 0.49664275, 0.48963817,
       0.48902728, 0.49235652, 0.49908478, 0.4993306 , 0.4909747 ,
       0.49002433, 0.49640419, 0.49075222, 0.49269731, 0.4894827 ])
```

Logistic Regression looks a little better for a submission.

Let's try one more model that we learned in class: LDA

In [162]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

LD = LinearDiscriminantAnalysis()
LD_model = LD.fit(train, target)
LD_model.score(train, target)
```

Out[162]:
```
0.6056
```

In [166]:
```
yhat_LD = LD_model.predict_proba(test)
```

In [167]:
```
yhat_LD[:100][:,1]
```

Out[167]:
```
array([0.86603582, 0.56809786, 0.57347395, 0.90900952, 0.63551295,
       0.45125773, 0.78606643, 0.16061337, 0.88058885, 0.19975021,
       0.85077676, 0.23887925, 0.91106503, 0.94602042, 0.87764619,
       0.91586529, 0.93410268, 0.537641  , 0.93030534, 0.88536546,
       0.30311638, 0.56323679, 0.57647963, 0.90878617, 0.508206  ,
       0.9517082 , 0.63200466, 0.64979236, 0.46156346, 0.89980518,
       0.35861787, 0.31295564, 0.52642507, 0.30415582, 0.46582421,
       0.54540915, 0.58770969, 0.49897851, 0.59071027, 0.39145019,
       0.83219476, 0.54289214, 0.78570002, 0.99996918, 0.9102223 ,
       0.43041029, 0.57897005, 0.54656414, 0.54885565, 0.13486572,
       0.56527545, 0.33921058, 0.47737635, 0.73678238, 0.91134414,
       0.48185077, 0.71862451, 0.57838264, 0.92868518, 0.5744218 ,
       0.92278963, 0.50282227, 0.54827766, 0.92449115, 0.18890559,
       0.62386074, 0.33011431, 0.16842234, 0.94153232, 0.90626375,
       0.53236912, 0.47694247, 0.31904617, 0.91709449, 0.46745027,
       0.7038623 , 0.49034847, 0.90054746, 0.44898947, 0.78460488,
       0.66952087, 0.30176186, 0.54167884, 0.53414444, 0.40998469,
       0.59389169, 0.55319859, 0.4858978 , 0.89628406, 0.63232745,
       0.90243552, 0.60717865, 0.58360322, 0.48580105, 0.6314519 ,
       0.92044884, 0.66132391, 0.57871844, 0.89633884, 0.89914386])
```

Looks good! Let's go with that!

In [170]:
```
submissionLD = pd.DataFrame({'MachineIdentifier': test_id,
                             'HasDetections': yhat_LD[:,1]}) # Create a su
bmission DataFrame
submissionLD.to_csv('../submissions/MalwareSubmissionsLD.csv', index=Fal
se) # Export to csv!
```

Conclusions

The models

We ran this on three separate classifiers that we used in our homeworks. The logistic regression model performed only slightly better than if we had randomly guessed the outcomes (Score: .507) However, the K Nearest Neighbors classifier performed much better (0.75) with a n-neighbor parameter of 2. Unfortunately, there is a good chance that the KNN model overfit our data. Also, this model may have performed better if the range of some of the features was reduced. The last model we ran was Linear Discriminant Analysis, which performed worse than KNN but better than LR (0.60). Surpisingly, this wasn't far off from our final score in Kaggle of 0.58!

Final Kaggle Score: 0.58. Why so low?

The possible reasons for our model scoring so low are:

- Small train dataset
  - We subsetted our dataset to 50,000 samples in the interest of building a model in a reasonal amount of time (builing a model on a larger dataset would have taken a long time to run). In hindsight we probably could have upped it a bit.
- Rudimentary models
  - We stuck to the models learned in class. Other competitors used more sophisticated models and hyperparameter tuning. Eventually, when we feel more comfortable with more advanced models, we might try to use one to get a better score.
- Lots of lost columns
  - We dropped many columns in our data cleaning for one reason or another. This is partly due to the fact that we wanted a smaller dataset, but also because we didn't know how to deal with some of them.
- Incorrect data types
  - Another potential problem was that we may have miscategorized some columns like "CityIdentifier" as a continuous type rather than categorical. However, we felt that we didn't know how to properly convert it to a categorical. It would have added too many columns to our dataframe after one hot encoding if we changed it to categorical.

In [ ]: