In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:

```
def analysis(col, tops = 10):
    temp = train[col].value_counts()
    temp = temp.iloc[:tops].index
    #temp = train.index
    temp_df = train[train[col].isin(temp)]
#     prob = temp_df[col].value_counts(normalize=True)
#     draw = np.random.choice(prob.index, p=prob, size=len(temp_df))
#     output = pd.Series(draw).value_counts(normalize=True).rename('simu
lated')
#     zeros = set(temp_df[col].dropna().unique()).difference(set(output.
index))
#     output = output.append(pd.Series([0 for i in zeros], index = zero
s)) / (temp_df[col].value_counts())
    temp_df['shuffle'] = temp_df['HasDetections'].sample(replace=False,
n=len(temp_df)).reset_index(drop=True)
    output = temp_df[temp_df['shuffle'] == 1][col].value_counts() / temp
_df[col].value_counts()
    pd.DataFrame({'train_data': temp_df[temp_df['HasDetections'] == 1][c
ol].value_counts()/ temp_df[col].value_counts(),
                             'random_data': output}).plot(kind = 'bar', figs
ize=(20,10))
    plt.title('Percent of Has detections by {} (most of the catogaries)'
.format(col))

    display(pd.DataFrame({'train_data': temp_df[temp_df['HasDetections']
== 1][col].value_counts()/ temp_df[col].value_counts(),
                             'random_data': output}))
    return stats.ks_2samp(temp_df[temp_df['HasDetections'] == 1][col].va
lue_counts(normalize = True),
                    output)




#stats.chi2_contingency([temp_df.groupby(col).HasDetections.mean(),
#                  temp_df.groupby(col).random_data.mean()])
```

In [3]:

```
COLS = [
    'HasDetections',
    'AVProductStatesIdentifier','AVProductsInstalled', 'AVProductsEnable
d'
]
```

In [4]:

```
train = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS)
```

In [5]:
```
train.head()
```

Out[5]:

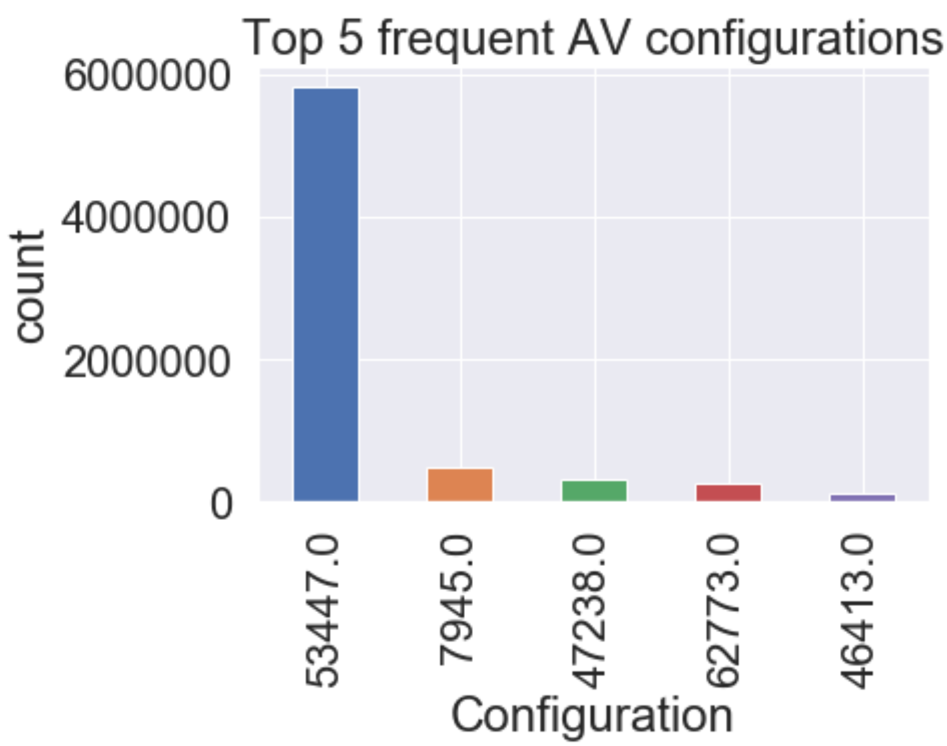
	AVProductStatesIdentifier	AVProductsInstalled	AVProductsEnabled	HasDetections
0	53447.0	1.0	1.0	0
1	53447.0	1.0	1.0	0
2	53447.0	1.0	1.0	0
3	53447.0	1.0	1.0	1
4	53447.0	1.0	1.0	1

In [6]:
```
#General analysis
```

In [7]:
```
#1.1 AVProductStatesIdentifier

#Top 20 categories detection
```

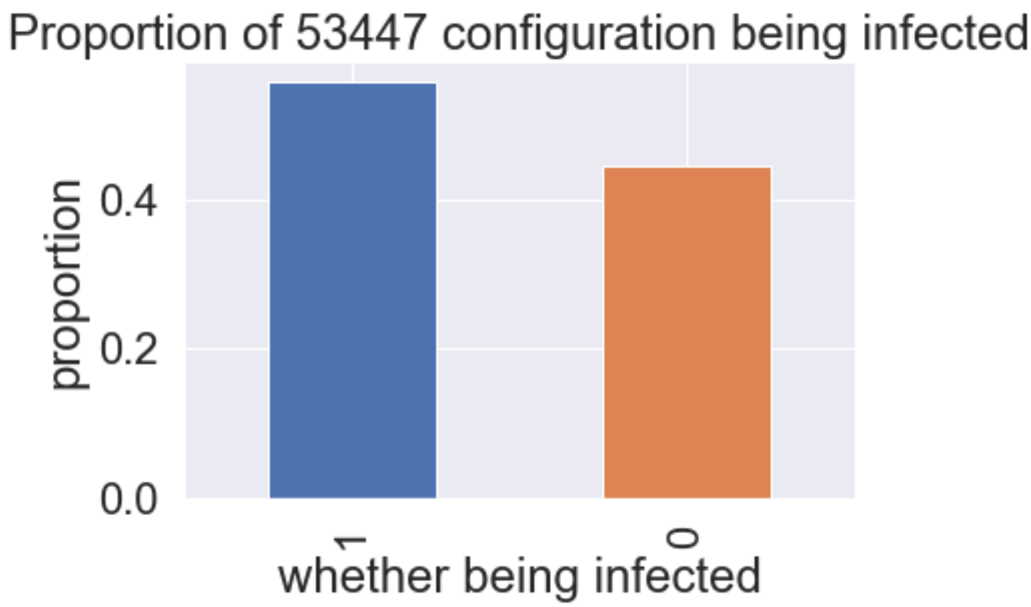
In [8]:
```
train[COLS[1]].value_counts().iloc[:5].plot(kind='bar', title= 'Top 5 fr
equent AV configurations')
plt.xlabel("Configuration")
plt.ylabel("count")
```

Out[8]:
```
Text(0, 0.5, 'count')
```



In [9]:
```
train[train[COLS[1]]==53447.0].HasDetections.value_counts(normalize=True
).plot("bar", title='Proportion of 53447 configuration being infected')
plt.xlabel("whether being infected")
plt.ylabel("proportion")
```

Out[9]:
```
Text(0, 0.5, 'proportion')
```



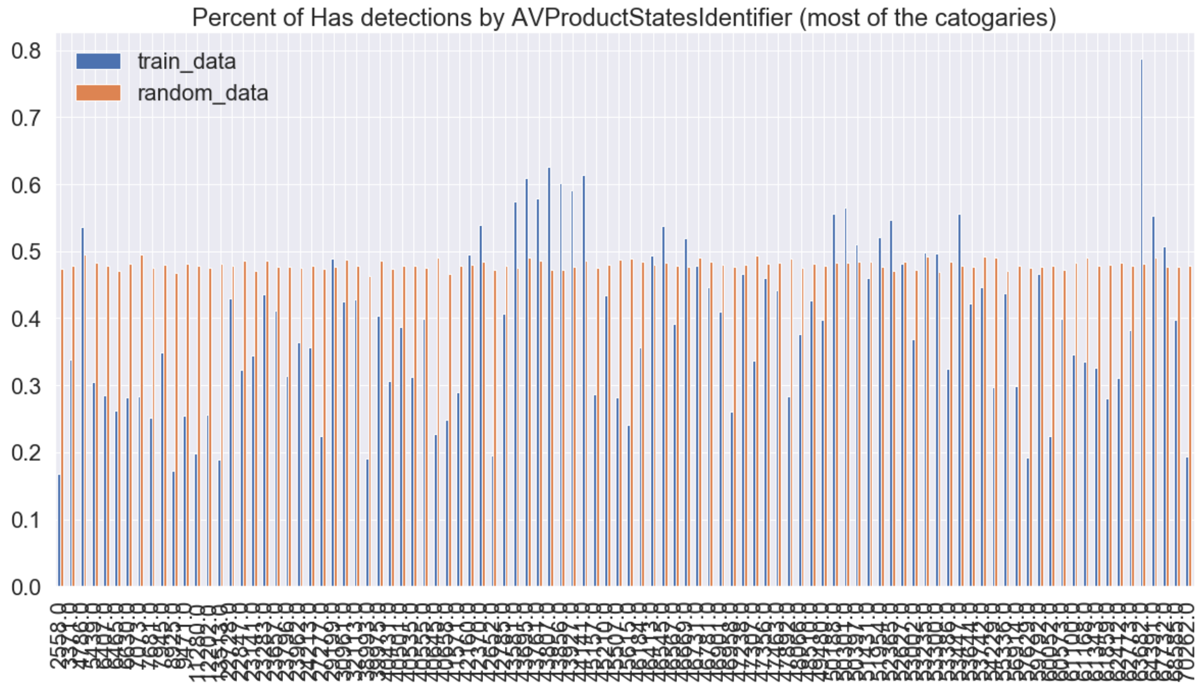
In [10]:
```
analysis(COLS[1], 100)
```

	train_data	random_data
2558.0	0.168525	0.474461
3371.0	0.338305	0.477813
4786.0	0.536552	0.495636
5439.0	0.304536	0.482259
6407.0	0.285132	0.479037
6465.0	0.262006	0.471441
6630.0	0.282516	0.481675
7073.0	0.284089	0.495114
7681.0	0.252316	0.475068
7945.0	0.348334	0.479703
8925.0	0.173160	0.467787
9471.0	0.254769	0.481998
11280.0	0.198155	0.477631
12202.0	0.256114	0.475834
13513.0	0.189224	0.480837
22728.0	0.429469	0.478450
22847.0	0.322525	0.485222
23141.0	0.344138	0.470654
23283.0	0.435090	0.486606
23657.0	0.411926	0.477298
23796.0	0.313957	0.477504
23962.0	0.364146	0.474878
24213.0	0.357153	0.477913
27277.0	0.223556	0.473394
29199.0	0.489096	0.476560
30961.0	0.425086	0.487019
32113.0	0.427595	0.478979
38993.0	0.191431	0.463096
39975.0	0.403295	0.485373
40431.0	0.306254	0.474026
...	...	...
50397.0	0.510341	0.484150
51431.0	0.460695	0.485192
51954.0	0.521178	0.476204

	train_data	random_data
52365.0	0.546349	0.470471
52627.0	0.480683	0.484372
53002.0	0.369352	0.471840
53235.0	0.498149	0.492694
53300.0	0.496026	0.469536
53386.0	0.324058	0.484497
53447.0	0.556365	0.479063
53644.0	0.422093	0.476636
53742.0	0.446748	0.492356
54229.0	0.296696	0.491099
55336.0	0.437007	0.471176
56914.0	0.298531	0.478456
57629.0	0.192626	0.475020
59792.0	0.466745	0.476236
60052.0	0.224161	0.478859
60573.0	0.398634	0.472131
61100.0	0.345621	0.483156
61168.0	0.334874	0.490935
61343.0	0.325536	0.478360
61859.0	0.280933	0.480332
62412.0	0.311499	0.483536
62773.0	0.382132	0.478700
63682.0	0.787928	0.480876
64391.0	0.553157	0.490700
67732.0	0.507315	0.477141
68585.0	0.398314	0.477391
70262.0	0.194309	0.478992

100 rows × 2 columns

Out[10]:
```
Ks_2sampResult(statistic=0.99, pvalue=1.2251433537012255e-44)
```



In [11]:
```
# hypothesis: Different Antivirius product will have different performance over the virius detection
```

In [ ]:

In [ ]:

In [ ]:

In [12]:
```
train[COLS[2]].value_counts()
```

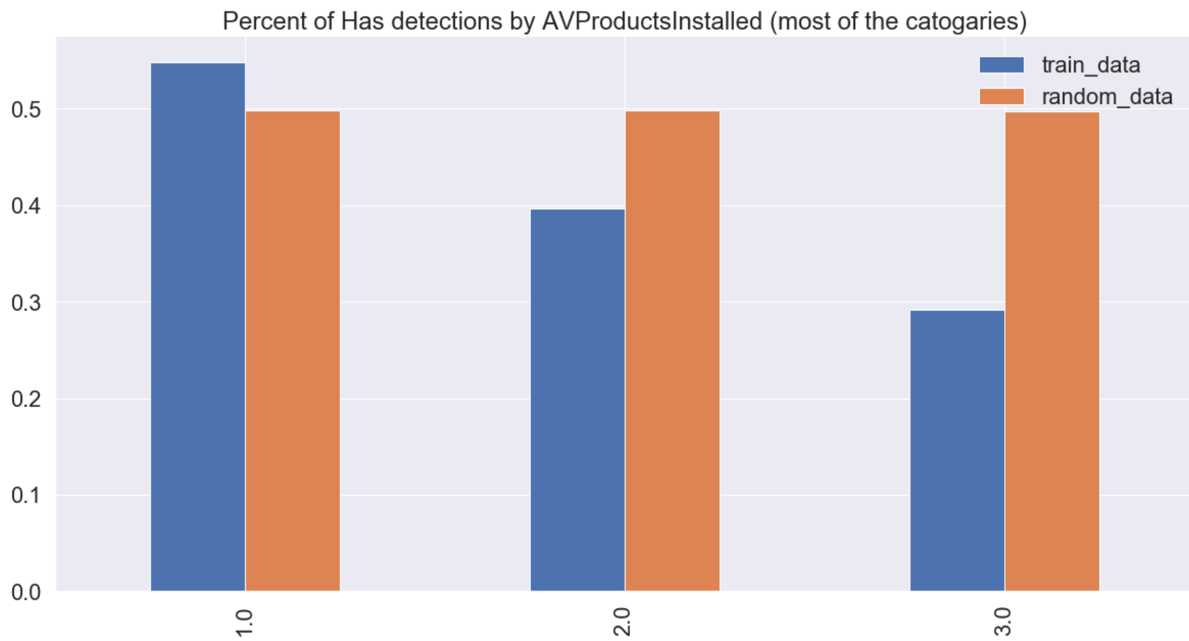
Out[12]:
```
1.0    6208893
2.0    2459008
3.0     208103
4.0       8757
5.0        471
6.0         28
7.0          1
0.0          1
Name: AVProductsInstalled, dtype: int64
```

In [13]:
```
# hypothesis: Different Antivirius product installed will have different performance over the virius detection
```

In [14]:
```
analysis(COLS[2], 3)
```

	train_data	random_data
1.0	0.548581	0.498079
2.0	0.396906	0.497881
3.0	0.291596	0.497114

Out[14]:
```
Ks_2sampResult(statistic=0.6666666666666666, pvalue=0.3197243332709645)
```



In [15]:
```
#Need deep analysis
```

In [16]:
```
# hypothesis: Different Antivirius product installed will have different
performance over the virius detection
```

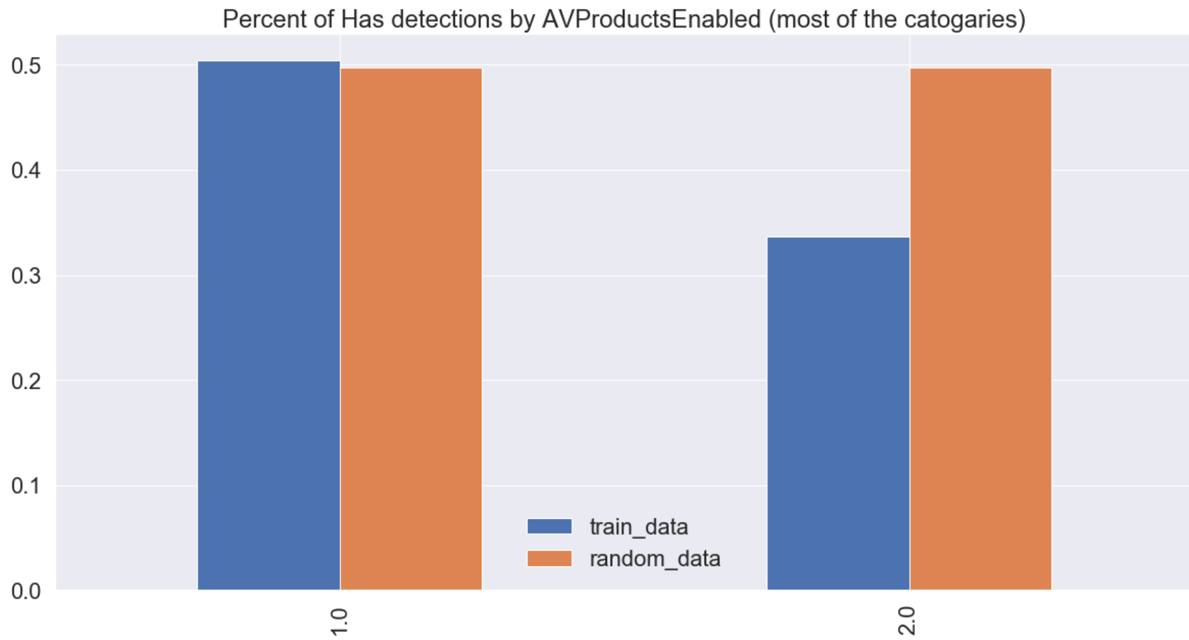
In [27]:
```
train[COLS[3]].value_counts()
```

Out[27]:
```
1.0    8654101
2.0     198652
0.0      25958
3.0       6075
4.0        453
5.0         23
Name: AVProductsEnabled, dtype: int64
```

In [17]:
```
analysis(COLS[3], 2)
```

	train_data	random_data
1.0	0.504636	0.496984
2.0	0.336422	0.497433

Out[17]:
```
Ks_2sampResult(statistic=0.5, pvalue=0.8438198245415606)
```



In [18]:
```
#Need deep analysis
```

In [ ]:

In [ ]:

In [ ]:

In [19]:
```
# trial w/ random forest
```

In [20]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                          remainder='drop')
    clf = RandomForestClassifier(n_estimators=7, max_depth=60)
    pl = Pipeline(steps=[('preprocessor', preproc),
                        ('clf', clf)
                        ])
    return pl
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                        , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [22]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'], index = COLS[1:])
for i in COLS[1:]:
    pl = skl([i])
    pl.fit(X_train, y_train)
    pred_score = pl.score(X_test, y_test)
    rand_score = pl.score(X_test, y_random)
    output.loc[i, 'Observation accuracy'] = pred_score
    output.loc[i, 'Random_Data accuracy'] = rand_score
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['combined', 'Observation accuracy'] = pred_score
output.loc['combined', 'Random_Data accuracy'] = rand_score
```

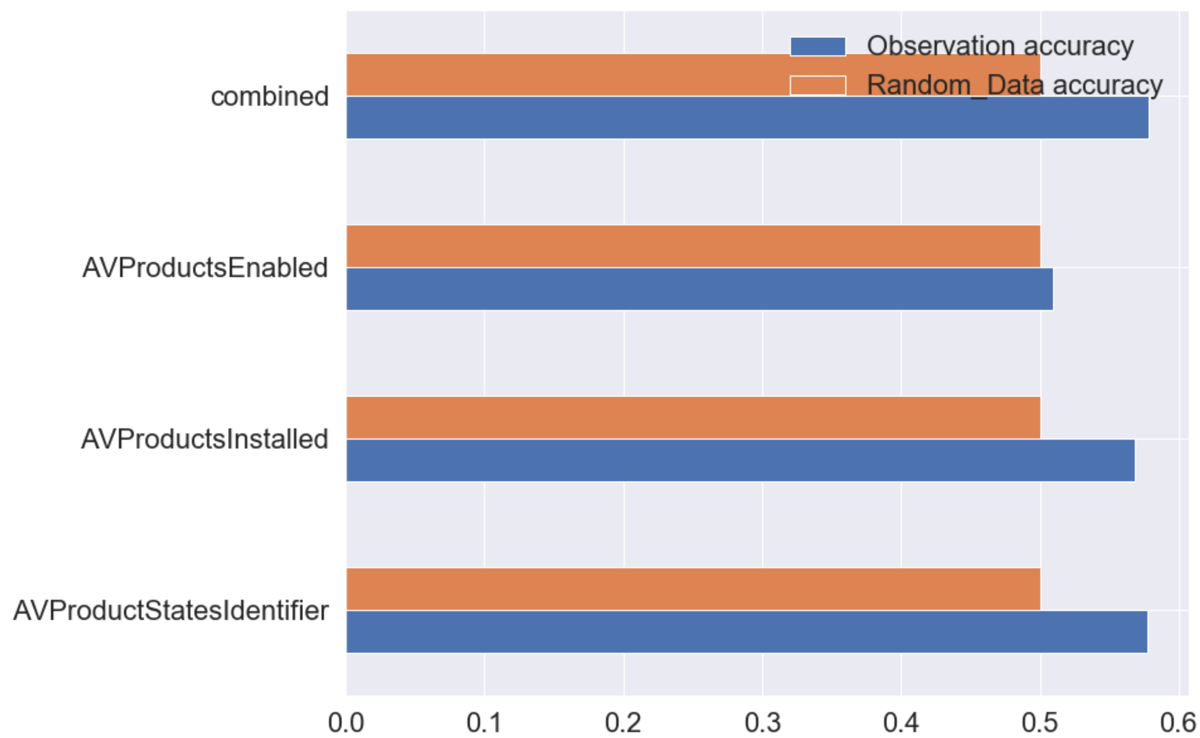
In [23]:
```
output
```

Out[23]:

	Observation accuracy	Random_Data accuracy
AVProductStatesIdentifier	0.577934	0.500461
AVProductsInstalled	0.568239	0.500398
AVProductsEnabled	0.509382	0.500699
combined	0.578303	0.500456

In [30]:
```
output.plot(kind = 'barh', ylim = (0.45, 0.65), figsize=[12,10])
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1a22697438>



In [25]:
```
#Conclusion, when using random forest clustering, 'AVProductStatesIdenti
fier' will dominate the performance
#of prediction, compare the comparison with random data, 'AVProductState
sIdentifier' have a significant imporvement
#in identifying malware.
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
def analysis(col, tops = 10):
    temp = train[col].value_counts()
    temp = temp.iloc[:tops].index
    #temp = train.index
    temp_df = train[train[col].isin(temp)]
#     prob = temp_df[col].value_counts(normalize=True)
#     draw = np.random.choice(prob.index, p=prob, size=len(temp_df))
#     output = pd.Series(draw).value_counts(normalize=True).rename('simu
lated')
#     zeros = set(temp_df[col].dropna().unique()).difference(set(output.
index))
#     output = output.append(pd.Series([0 for i in zeros], index = zero
s)) / (temp_df[col].value_counts())
    temp_df['shuffle'] = temp_df['HasDetections'].sample(replace=False,
n=len(temp_df)).reset_index(drop=True)
    output = temp_df[temp_df['shuffle'] == 1][col].value_counts() / temp
_df[col].value_counts()
    pd.DataFrame({'train_data': temp_df[temp_df['HasDetections'] == 1][c
ol].value_counts()/ temp_df[col].value_counts(),
                         'random_data': output}).plot(kind = 'bar', figs
ize=(20,10))
    plt.title('Percent of Has detections by {} (most of the catogaries)'
.format(col))

    display(pd.DataFrame({'train_data': temp_df[temp_df['HasDetections']
== 1][col].value_counts()/ temp_df[col].value_counts(),
                         'random_data': output}))
    return stats.ks_2samp(temp_df[temp_df['HasDetections'] == 1][col].va
lue_counts(normalize = True),
                    output)




#stats.chi2_contingency([temp_df.groupby(col).HasDetections.mean(),
#                 temp_df.groupby(col).random_data.mean()])
```

In [3]:
```
COLS = [
    'HasDetections',
    'Platform',
    'OsBuild'
]
```

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

In [5]:
```
train.head()
```

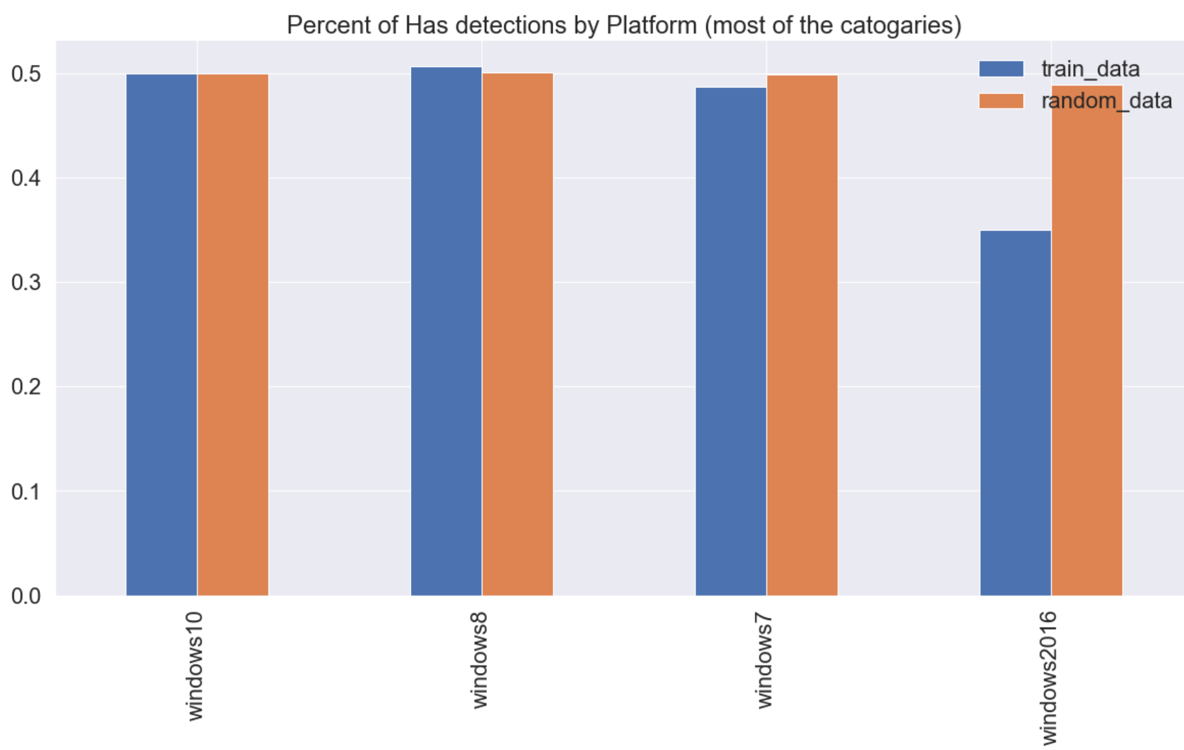
Out[5]:

	Platform	OsBuild	HasDetections
0	windows10	17134	0
1	windows10	17134	0
2	windows10	17134	0
3	windows10	17134	1
4	windows10	17134	1

In [6]:
```
analysis(COLS[1])
```

	train_data	random_data
windows10	0.500032	0.499803
windows8	0.506720	0.500540
windows7	0.486511	0.498930
windows2016	0.349593	0.489040

Out[6]:
```
Ks_2sampResult(statistic=0.75, pvalue=0.10749046502096637)
```

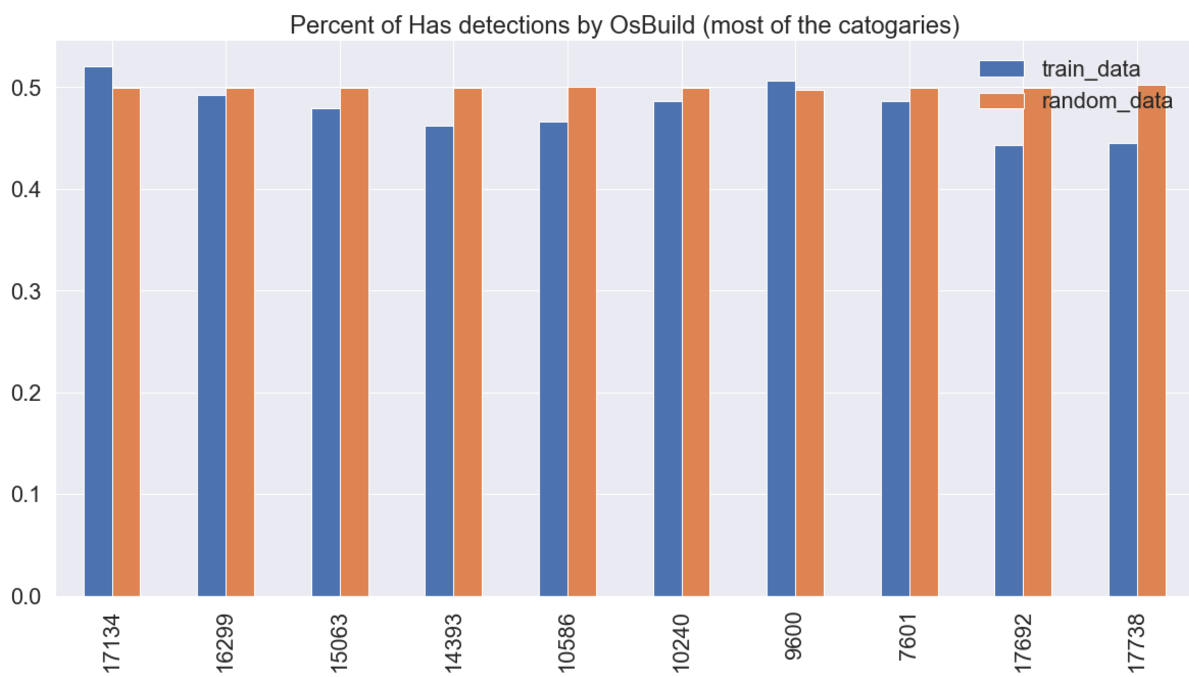


In [7]:
```
# virius and platform is not likely revelent
```

In [8]:
```
analysis(COLS[2])
```

	train_data	random_data
17134	0.520727	0.498950
16299	0.492128	0.498968
15063	0.478875	0.499143
14393	0.462269	0.499039
10586	0.465831	0.500046
10240	0.486584	0.499478
9600	0.506720	0.497872
7601	0.486432	0.499368
17692	0.443467	0.499058
17738	0.445117	0.502421

Out[8]:
```
Ks_2sampResult(statistic=1.0, pvalue=1.8879793657162556e-05)
```



In [9]:
```
# We assmue malware detection may have no significant relation with oper
ating system
```

In [ ]:
```

```

In [10]:
```
# random forest clustering to comfirm
```

In [11]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                           remainder='drop')
    clf = RandomForestClassifier(n_estimators=7, max_depth=60)
    pl = Pipeline(steps=[('preprocessor', preproc),
                     ('clf', clf)
                     ])
    return pl
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                    , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [13]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'], index = COLS[1:])
for i in COLS[1:]:
    pl = skl([i])
    pl.fit(X_train, y_train)
    pred_score = pl.score(X_test, y_test)
    rand_score = pl.score(X_test, y_random)
    output.loc[i, 'Observation accuracy'] = pred_score
    output.loc[i, 'Random_Data accuracy'] = rand_score
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['combined', 'Observation accuracy'] = pred_score
output.loc['combined', 'Random_Data accuracy'] = rand_score
```

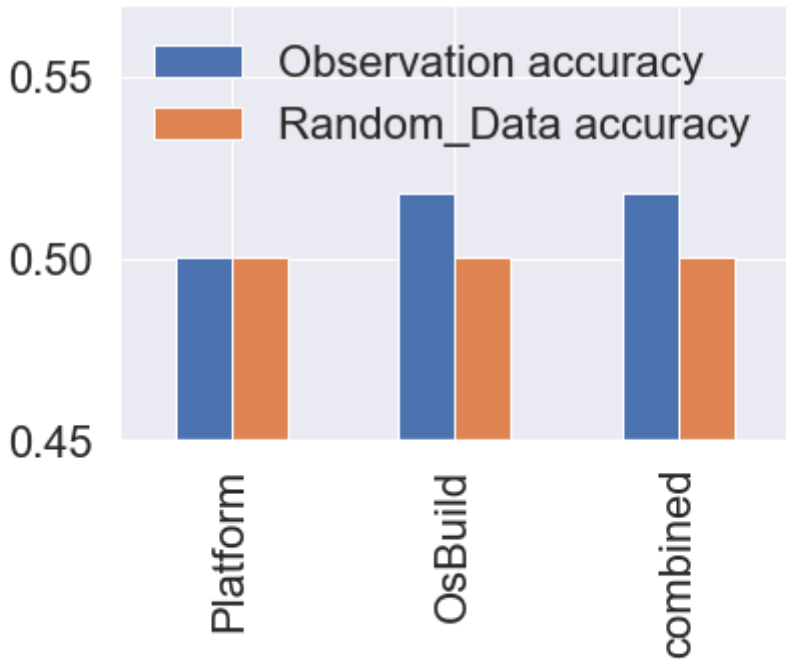
In [14]:
```
output
```

Out[14]:

	Observation accuracy	Random_Data accuracy
Platform	0.500503	0.500177
OsBuild	0.518036	0.500423
combined	0.518036	0.500423

In [15]:
```
output.plot(kind = 'bar', ylim = (0.45, 0.57))
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x23d8044c908>



In [16]:
```
#Conclusion, In general, Operating system has a slightly influence to ma
lware detection (not very significant)
#'OSBuild' will have a more significant influence when we proceed random
forest clustering,
#and 'Platform' may have no affect to malware detection. When we combine
two 'OSBuild' will dominate the
#clf.
```

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
def analysis(col, tops = 10):
    temp = train[col].value_counts()
    temp = temp.iloc[:tops].index
    #temp = train.index
    temp_df = train[train[col].isin(temp)]
#     prob = temp_df[col].value_counts(normalize=True)
#     draw = np.random.choice(prob.index, p=prob, size=len(temp_df))
#     output = pd.Series(draw).value_counts(normalize=True).rename('simu
lated')
#     zeros = set(temp_df[col].dropna().unique()).difference(set(output.
index))
#     output = output.append(pd.Series([0 for i in zeros], index = zero
s)) / (temp_df[col].value_counts())
    temp_df['shuffle'] = temp_df['HasDetections'].sample(replace=False,
n=len(temp_df)).reset_index(drop=True)
    output = temp_df[temp_df['shuffle'] == 1][col].value_counts() / temp
_df[col].value_counts()
    pd.DataFrame({'train_data': temp_df[temp_df['HasDetections'] == 1][c
ol].value_counts()/ temp_df[col].value_counts(),
                         'random_data': output}).plot(kind = 'bar', figs
ize=(20,10))
    plt.title('Percent of Has detections by {} (most of the catogaries)'
.format(col))

    display(pd.DataFrame({'train_data': temp_df[temp_df['HasDetections']
== 1][col].value_counts()/ temp_df[col].value_counts(),
                         'random_data': output}))
    return stats.ks_2samp(temp_df[temp_df['HasDetections'] == 1][col].va
lue_counts(normalize = True),
                   output)




#stats.chi2_contingency([temp_df.groupby(col).HasDetections.mean(),
#                   temp_df.groupby(col).random_data.mean()])
```

In [3]:
```
COLS = [
    'HasDetections',
    'Census_ProcessorCoreCount',
    'Census_PrimaryDiskTotalCapacity',
    'Processor'
]
```

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

In [5]:
```
train.head()
```

Out[5]:

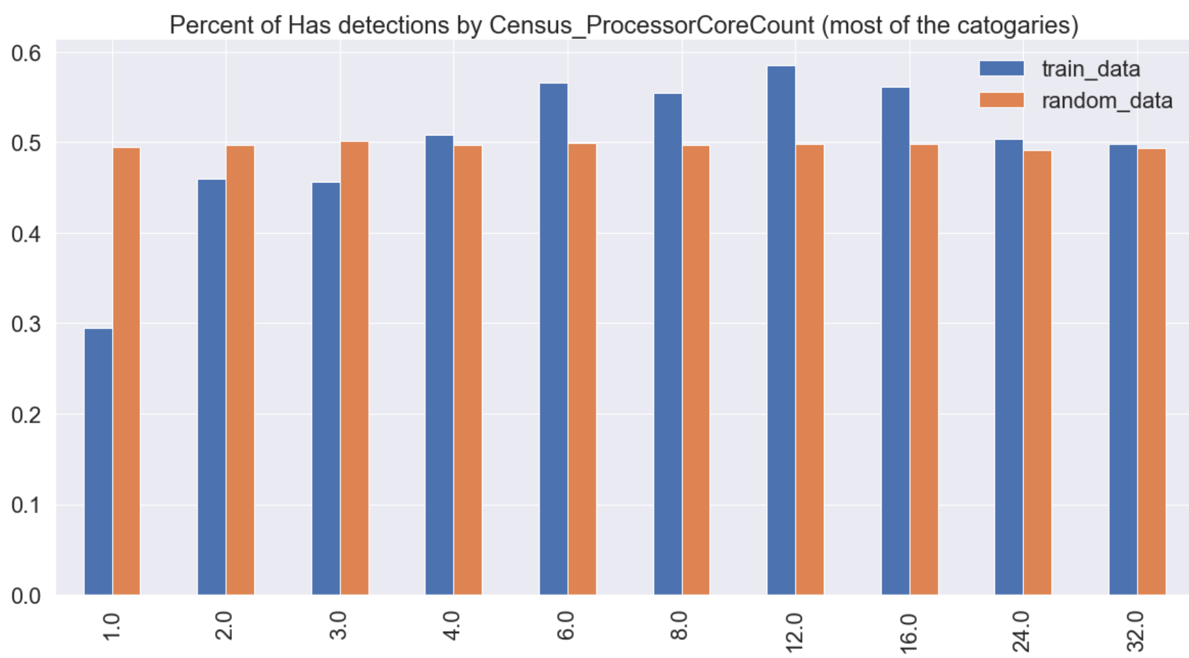
	Processor	Census_ProcessorCoreCount	Census_PrimaryDiskTotalCapacity	HasDetections
0	x64	4.0	476940.0	0
1	x64	4.0	476940.0	0
2	x64	4.0	114473.0	0
3	x64	4.0	238475.0	1
4	x64	4.0	476940.0	1

In [6]:
```
#barplot of random_data and chi-square test statiscs over the proportion

#only takes majority of large data to proceed analyis
```

In [7]:
```
analysis(COLS[1])
```

	train_data	random_data
1.0	0.295042	0.494843
2.0	0.459875	0.496916
3.0	0.456038	0.501915
4.0	0.507915	0.497158
6.0	0.566400	0.498798
8.0	0.555008	0.496822
12.0	0.584691	0.497994
16.0	0.561587	0.498032
24.0	0.503519	0.491608
32.0	0.498596	0.493446

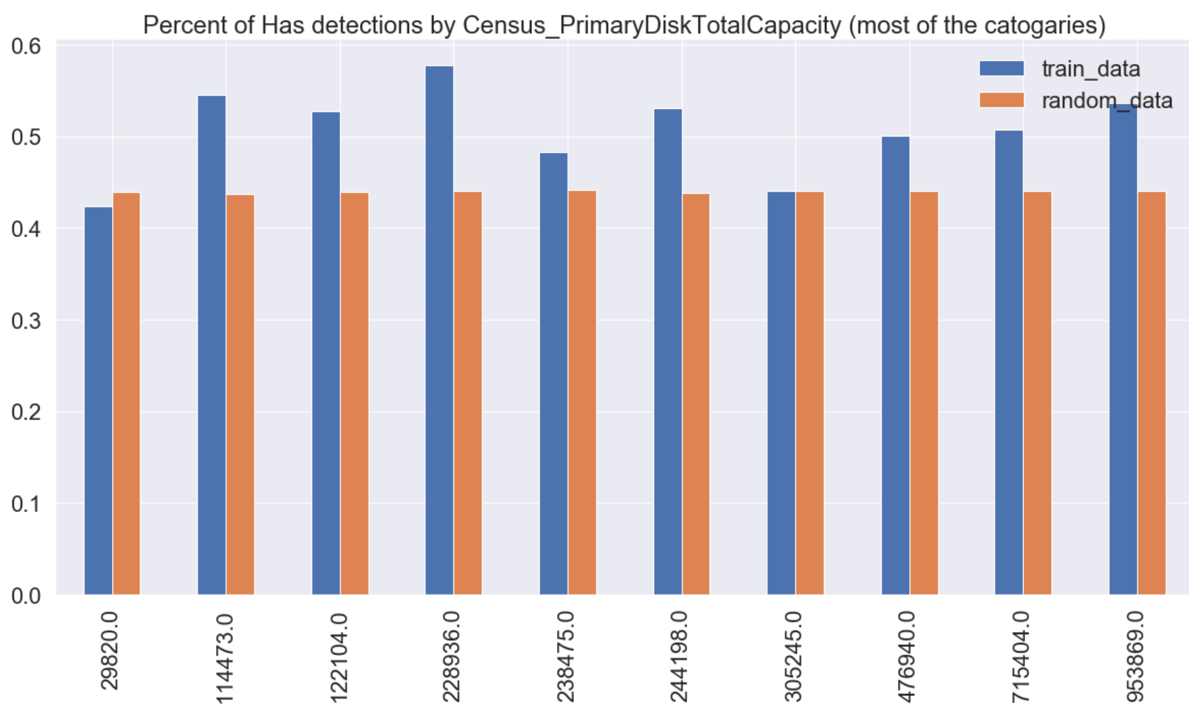
Out[7]:
```
Ks_2sampResult(statistic=0.9, pvalue=0.00017011925273829756)
```



In [8]:
```
analysis(COLS[2])
```

	train_data	random_data
29820.0	0.424125	0.439374
114473.0	0.545365	0.437663
122104.0	0.527792	0.439861
228936.0	0.577324	0.440526
238475.0	0.483430	0.441772
244198.0	0.530764	0.438167
305245.0	0.440135	0.440824
476940.0	0.500258	0.440021
715404.0	0.507006	0.440239
953869.0	0.536908	0.440147

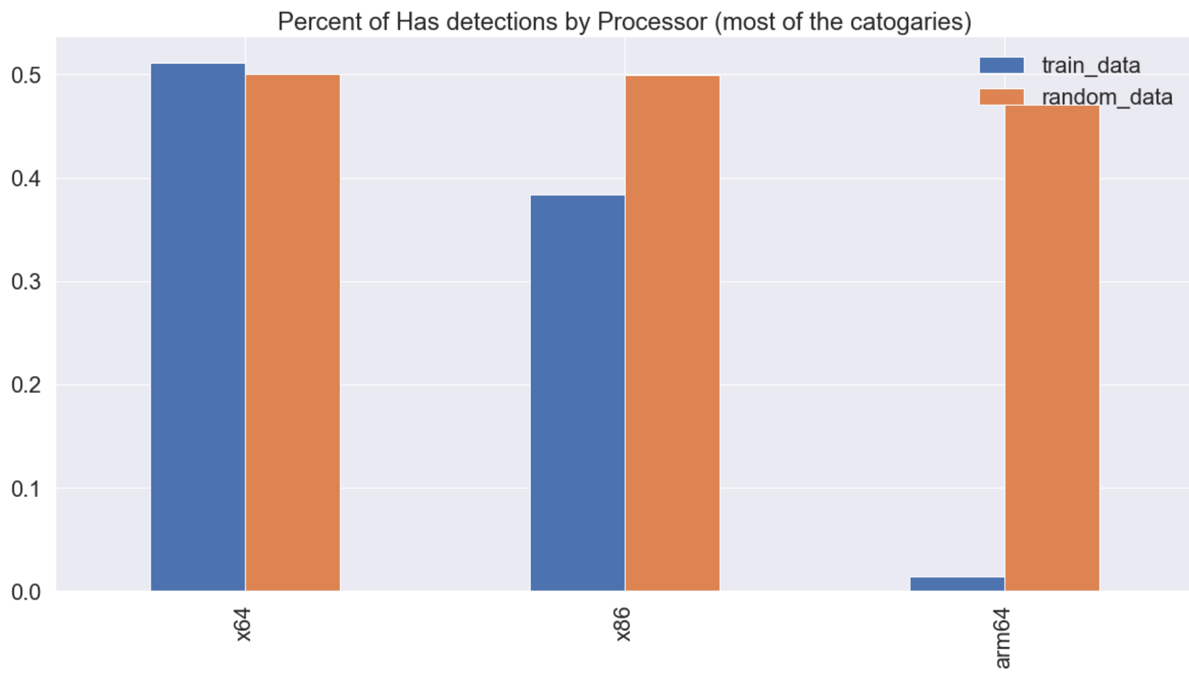
Out[8]:
```
Ks_2sampResult(statistic=1.0, pvalue=1.8879793657162556e-05)
```



In [9]:
```
analysis(COLS[3])
```

	train_data	random_data
x64	0.511446	0.499851
x86	0.384202	0.499226
arm64	0.014451	0.471098

Out[9]:
```
Ks_2sampResult(statistic=0.6666666666666666, pvalue=0.3197243332709645
6)
```



In [10]:
```
#First step assumption:

#Based on plot and statistics above, we first assmue Processor > TotalDi
skCapacity > Processor Core count
```

In [ ]:

In [11]:
```
# deep study
```

In [12]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                          remainder='drop')
    clf = RandomForestClassifier(n_estimators=7, max_depth=60)
    pl = Pipeline(steps=[('preprocessor', preproc),
                         ('clf', clf)
                         ])
    return pl
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [14]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'], index = COLS[1:])
for i in COLS[1:]:
    pl = skl([i])
    pl.fit(X_train, y_train)
    pred_score = pl.score(X_test, y_test)
    rand_score = pl.score(X_test, y_random)
    output.loc[i, 'Observation accuracy'] = pred_score
    output.loc[i, 'Random_Data accuracy'] = rand_score
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['combined', 'Observation accuracy'] = pred_score
output.loc['combined', 'Random_Data accuracy'] = rand_score
```

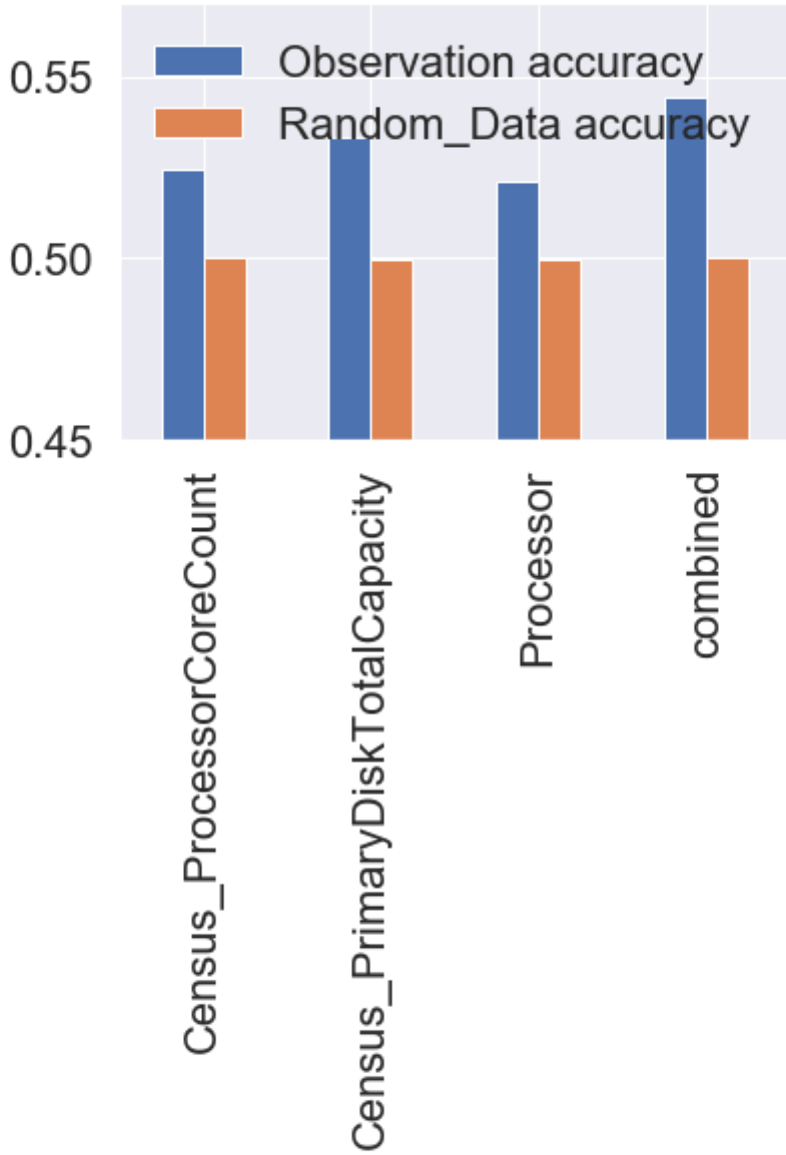
In [15]:
```
output
```

Out[15]:

	Observation accuracy	Random_Data accuracy
Census_ProcessorCoreCount	0.524082	0.499893
Census_PrimaryDiskTotalCapacity	0.533193	0.499547
Processor	0.521055	0.49966
combined	0.543938	0.499912

In [16]:
```
output.plot(kind = 'bar', ylim = (0.45, 0.57))
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x21b00045748>



In [17]:
```
# Conclusion, hardware can influence the prediction under random forest
 classifer of malware
# The features combined has significant imporvement, which means it help
with malware detection
# when we combines features.
```

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
def analysis(col, tops = 10):
    temp = train[col].value_counts()
    temp = temp.iloc[:tops].index
    #temp = train.index
    temp_df = train[train[col].isin(temp)]
#     prob = temp_df[col].value_counts(normalize=True)
#     draw = np.random.choice(prob.index, p=prob, size=len(temp_df))
#     output = pd.Series(draw).value_counts(normalize=True).rename('simu
lated')
#     zeros = set(temp_df[col].dropna().unique()).difference(set(output.
index))
#     output = output.append(pd.Series([0 for i in zeros], index = zero
s)) / (temp_df[col].value_counts())
    temp_df['shuffle'] = temp_df['HasDetections'].sample(replace=False,
n=len(temp_df)).reset_index(drop=True)
    output = temp_df[temp_df['shuffle'] == 1][col].value_counts() / temp
_df[col].value_counts()
    pd.DataFrame({'train_data': temp_df[temp_df['HasDetections'] == 1][c
ol].value_counts()/ temp_df[col].value_counts(),
                              'random_data': output}).plot(kind = 'bar', figs
ize=(20,10))
    plt.title('Percent of Has detections by {} (most of the catogaries)'
.format(col))

    display(pd.DataFrame({'train_data': temp_df[temp_df['HasDetections']
== 1][col].value_counts()/ temp_df[col].value_counts(),
                              'random_data': output}))
    return stats.ks_2samp(temp_df[temp_df['HasDetections'] == 1][col].va
lue_counts(normalize = True),
                    output)




#stats.chi2_contingency([temp_df.groupby(col).HasDetections.mean(),
#                   temp_df.groupby(col).random_data.mean()])
```

In [3]:
```
COLS = [
    'HasDetections',
    'IsBeta',
    'ProductName'
]
```

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

In [5]:
```
train.head()
```

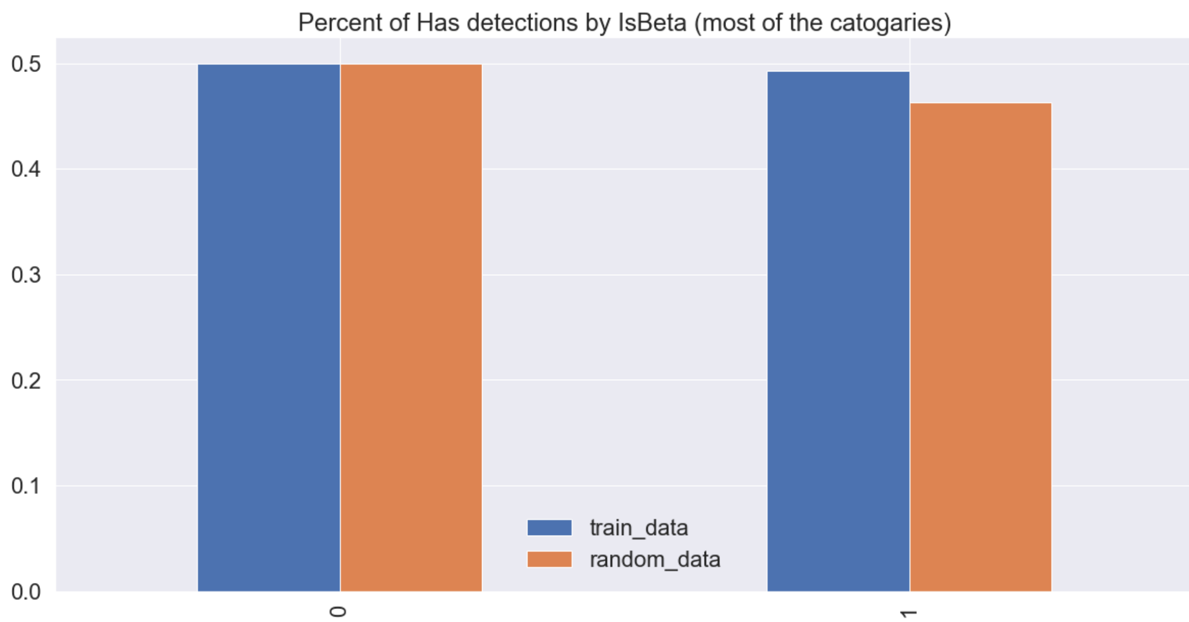
Out[5]:

	ProductName	IsBeta	HasDetections
0	win8defender	0	0
1	win8defender	0	0
2	win8defender	0	0
3	win8defender	0	1
4	win8defender	0	1

In [6]:
```
analysis(COLS[1])
```

	train_data	random_data
0	0.499793	0.499793
1	0.492537	0.462687

Out[6]:
```
Ks_2sampResult(statistic=0.5, pvalue=0.8438198245415606)
```



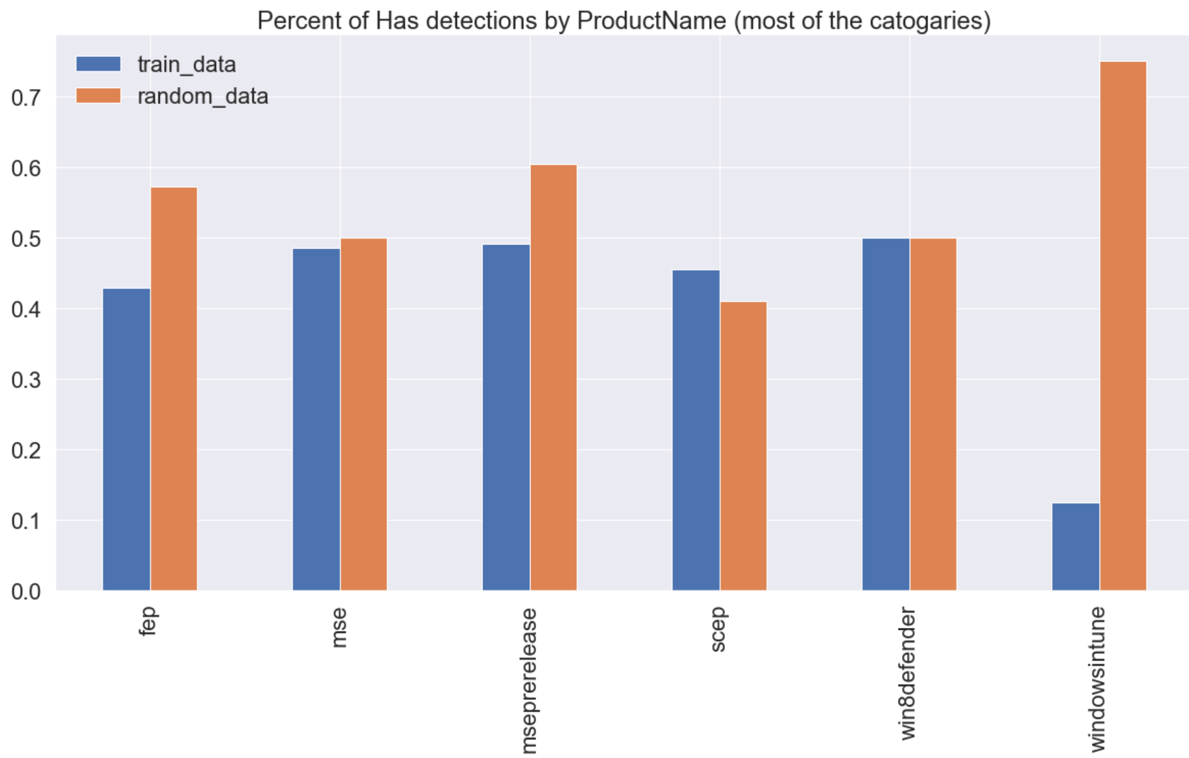
In [7]:
```
train.groupby('ProductName').HasDetections.mean()
```

Out[7]:
```
ProductName
fep                0.428571
mse                0.484448
mseprerelease      0.490566
scep               0.454545
win8defender       0.499958
windowsintune      0.125000
Name: HasDetections, dtype: float64
```

In [8]:
```
analysis(COLS[2])
```

	train_data	random_data
fep	0.428571	0.571429
mse	0.484448	0.499626
mseprerelease	0.490566	0.603774
scep	0.454545	0.409091
win8defender	0.499958	0.499794
windowsintune	0.125000	0.750000

Out[8]:
```
Ks_2sampResult(statistic=0.8333333333333334, pvalue=0.01223815312587811
2)
```



In [9]:
```
# We assume there has significantly difference between Defender State an
d Malware detection
```

In [ ]:
```
```

In [10]:
```
# random forest clustering to confirm our assumption
```

In [11]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                          remainder='drop')
    clf = RandomForestClassifier(n_estimators=7, max_depth=60)
    pl = Pipeline(steps=[('preprocessor', preproc),
                    ('clf', clf)
                    ])
    return pl
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                   , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [13]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'], index = COLS[1:])
for i in COLS[1:]:
    pl = skl([i])
    pl.fit(X_train, y_train)
    pred_score = pl.score(X_test, y_test)
    rand_score = pl.score(X_test, y_random)
    output.loc[i, 'Observation accuracy'] = pred_score
    output.loc[i, 'Random_Data accuracy'] = rand_score
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['combined', 'Observation accuracy'] = pred_score
output.loc['combined', 'Random_Data accuracy'] = rand_score
```

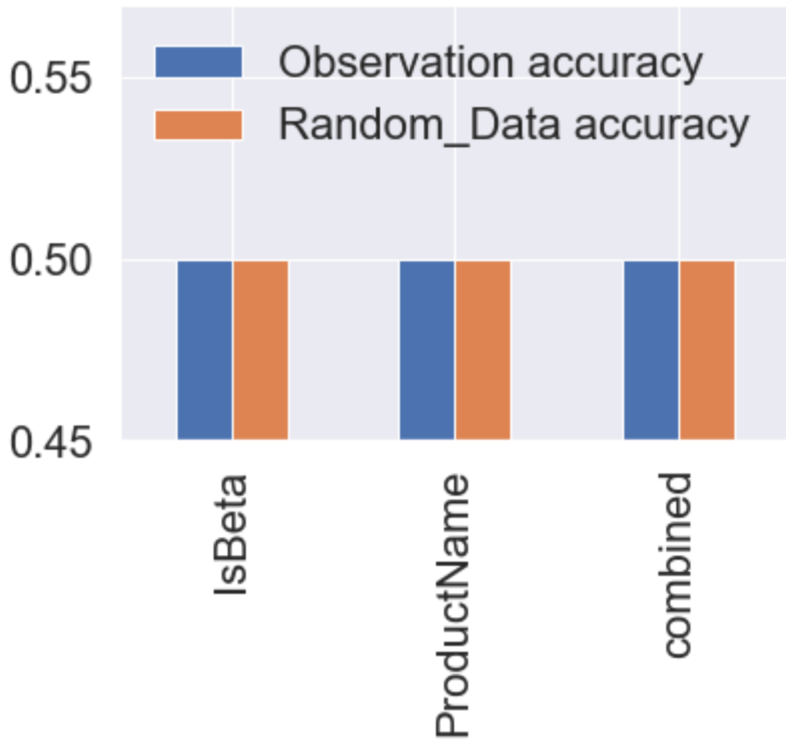
In [14]:
```
output
```

Out[14]:

	Observation accuracy	Random_Data accuracy
IsBeta	0.500057	0.500054
ProductName	0.500058	0.500058
combined	0.500056	0.500057

In [15]:
```
output.plot(kind = 'bar', ylim = (0.45, 0.57))
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27d00074a20>
```



In [16]:
```
# Conclusion: defender state has no influence to malware detection.
```

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
def analysis(col, tops = 10):
    temp = train[col].value_counts()
    temp = temp.iloc[:tops].index
    #temp = train.index
    temp_df = train[train[col].isin(temp)]
#     prob = temp_df[col].value_counts(normalize=True)
#     draw = np.random.choice(prob.index, p=prob, size=len(temp_df))
#     output = pd.Series(draw).value_counts(normalize=True).rename('simu
lated')
#     zeros = set(temp_df[col].dropna().unique()).difference(set(output.
index))
#     output = output.append(pd.Series([0 for i in zeros], index = zero
s)) / (temp_df[col].value_counts())
    temp_df['shuffle'] = temp_df['HasDetections'].sample(replace=False,
n=len(temp_df)).reset_index(drop=True)
    output = temp_df[temp_df['shuffle'] == 1][col].value_counts() / temp
_df[col].value_counts()
    pd.DataFrame({'train_data': temp_df[temp_df['HasDetections'] == 1][c
ol].value_counts()/ temp_df[col].value_counts(),
                            'random_data': output}).plot(kind = 'bar', figs
ize=(20,10))
    plt.title('Percent of Has detections by {} (most of the catogaries)'
.format(col))

    display(pd.DataFrame({'train_data': temp_df[temp_df['HasDetections']
== 1][col].value_counts()/ temp_df[col].value_counts(),
                            'random_data': output}))
    return stats.ks_2samp(temp_df[temp_df['HasDetections'] == 1][col].va
lue_counts(normalize = True),
                   output)




#stats.chi2_contingency([temp_df.groupby(col).HasDetections.mean(),
#                  temp_df.groupby(col).random_data.mean()])
```

In [3]:
```
COLS = [
    'HasDetections',
    'GeoNameIdentifier',
    'CountryIdentifier'
]
```

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

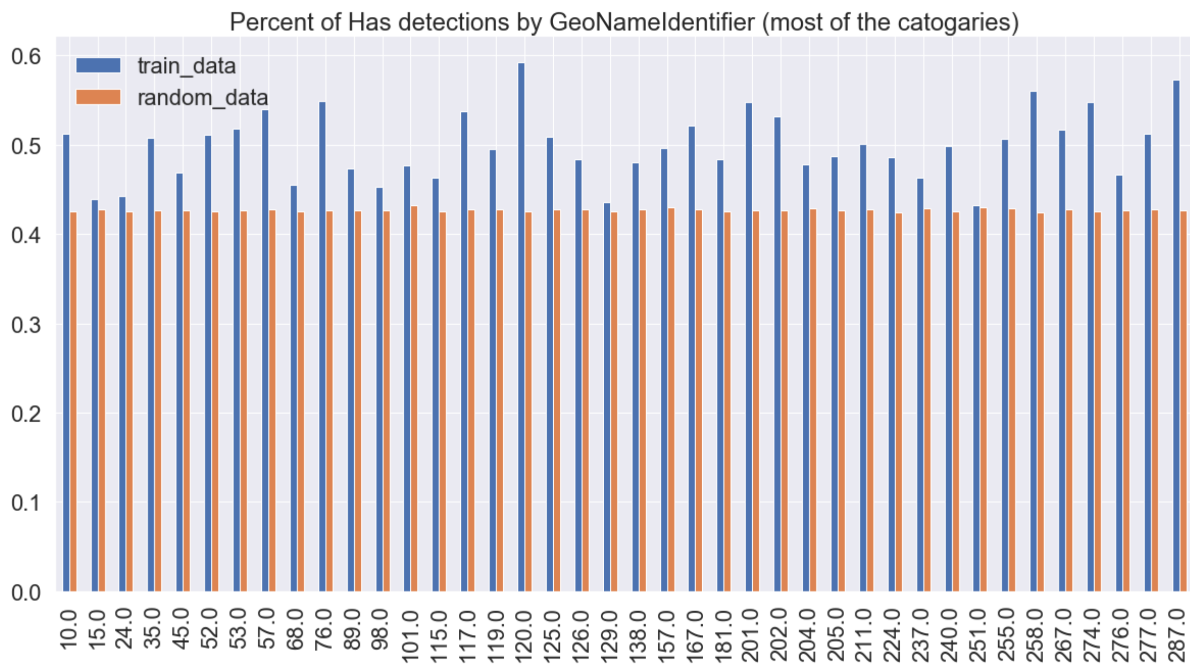
In [5]:
```
#top 40 contries analysis
```

In [6]:
```
analysis(COLS[1], 40)
```

	train_data	random_data
10.0	0.511848	0.425225
15.0	0.439316	0.427997
24.0	0.441928	0.424914
35.0	0.507216	0.426260
45.0	0.469069	0.425998
52.0	0.510999	0.425478
53.0	0.517883	0.426859
57.0	0.540070	0.427874
68.0	0.454544	0.425157
76.0	0.548960	0.426389
89.0	0.473733	0.426760
98.0	0.452187	0.426772
101.0	0.476334	0.431922
115.0	0.463466	0.425081
117.0	0.537920	0.427493
119.0	0.495623	0.427324
120.0	0.592210	0.425788
125.0	0.508966	0.427463
126.0	0.484070	0.427763
129.0	0.435529	0.425826
138.0	0.480256	0.428146
157.0	0.495774	0.429770
167.0	0.521564	0.427963
181.0	0.483826	0.425507
201.0	0.547603	0.426698
202.0	0.532145	0.425885
204.0	0.478396	0.429171
205.0	0.487162	0.426353
211.0	0.501151	0.427972
224.0	0.486248	0.424333
237.0	0.463296	0.428922
240.0	0.498015	0.425391
251.0	0.431757	0.429458
255.0	0.506378	0.428539

	train_data	random_data
258.0	0.559801	0.424347
267.0	0.516551	0.427381
274.0	0.547439	0.424971
276.0	0.466328	0.426719
277.0	0.511857	0.427246
287.0	0.573277	0.426770

Out[6]:
```
Ks_2sampResult(statistic=1.0, pvalue=6.133847783205273e-19)
```



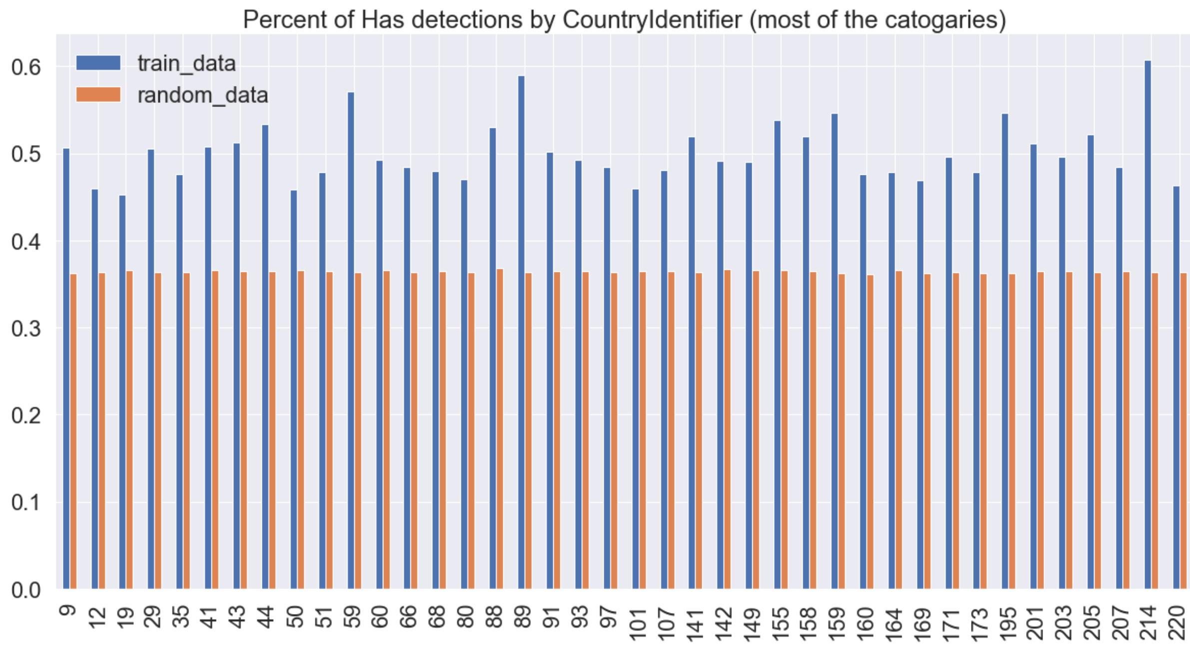
In [7]:
```
#from the histogram, we see there is a difference in top 4 counties
```

In [8]:
```
analysis(COLS[2], 40)
```

	train_data	random_data
9	0.506709	0.362701
12	0.459381	0.363193
19	0.452238	0.365839
29	0.505318	0.363857
35	0.476030	0.363344
41	0.507615	0.366280
43	0.512030	0.365131
44	0.533652	0.365011
50	0.458017	0.366223
51	0.478736	0.364130
59	0.570547	0.363931
60	0.492523	0.365513
66	0.484162	0.363982
68	0.479183	0.364715
80	0.469614	0.363214
88	0.530356	0.367622
89	0.589220	0.363697
91	0.501763	0.364579
93	0.492203	0.364471
97	0.483811	0.363961
101	0.459183	0.364113
107	0.481192	0.365178
141	0.519668	0.363257
142	0.490838	0.367251
149	0.490176	0.365425
155	0.538468	0.366261
158	0.519582	0.364786
159	0.546358	0.361953
160	0.475467	0.361555
164	0.477821	0.366351
169	0.468479	0.362477
171	0.496332	0.363675
173	0.477802	0.362556
195	0.546919	0.362091

	train_data	random_data
201	0.510665	0.364352
203	0.496419	0.364436
205	0.521958	0.363683
207	0.483938	0.364686
214	0.606910	0.363608
220	0.463472	0.363520

Out[8]: Ks_2sampResult(statistic=1.0, pvalue=6.133847783205273e-19)



In [9]:
```
# We assume there is no significant influence when malware detection
```

In [10]:
```
# random forest clustering to confirm
```

In [11]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                            remainder='drop')
    clf = RandomForestClassifier(n_estimators=7, max_depth=60)
    pl = Pipeline(steps=[('preprocessor', preproc),
                        ('clf', clf)
                        ])
    return pl
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                                , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [13]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'], index = COLS[1:])
for i in COLS[1:]:
    pl = skl([i])
    pl.fit(X_train, y_train)
    pred_score = pl.score(X_test, y_test)
    rand_score = pl.score(X_test, y_random)
    output.loc[i, 'Observation accuracy'] = pred_score
    output.loc[i, 'Random_Data accuracy'] = rand_score
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['combined', 'Observation accuracy'] = pred_score
output.loc['combined', 'Random_Data accuracy'] = rand_score
```

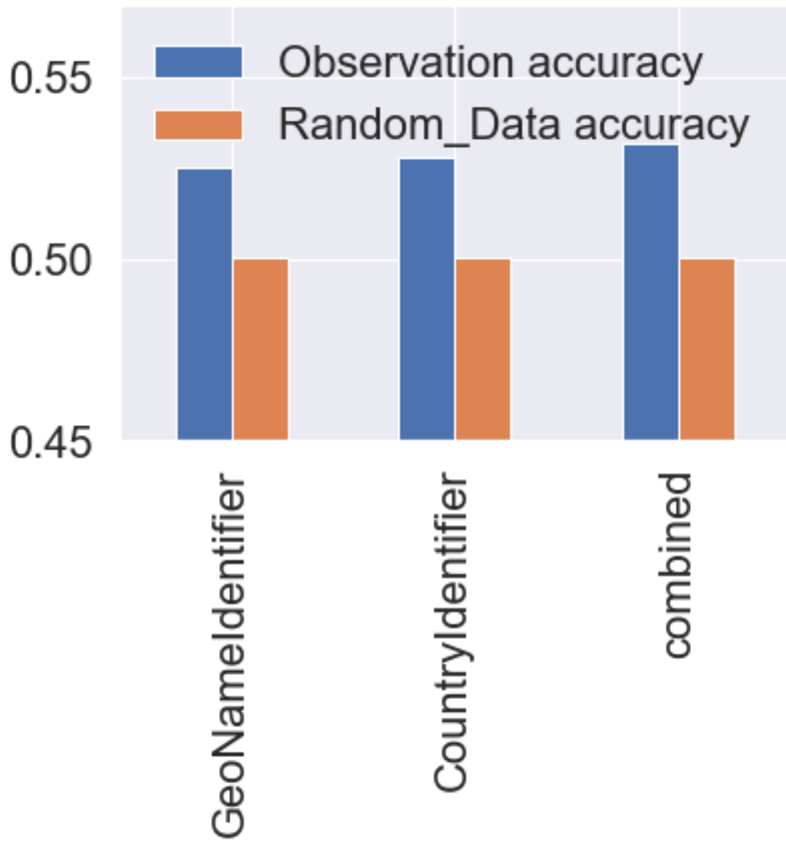
In [14]:
```
output
```

Out[14]:

	Observation accuracy	Random_Data accuracy
GeoNameIdentifier	0.525165	0.500321
CountryIdentifier	0.528054	0.500553
combined	0.532115	0.500415

In [15]:
```
output.plot(kind = 'bar', ylim = (0.45, 0.57))
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17886078898>
```



In [ ]:
```

```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
COLS1 = [
    'HasDetections',
    'AVProductStatesIdentifier','AVProductsInstalled', 'AVProductsEnable
d'
]
COLS2 = [
    'HasDetections',
    'Platform',
    'OsBuild'
]
COLS3 = [
    'HasDetections',
    'Census_ProcessorCoreCount',
    'Census_PrimaryDiskTotalCapacity',
    'Processor'
]
COLS4 = [
    'HasDetections',
    'IsBeta',
    'ProductName'
]
COLS5 = [
    'HasDetections',
    'GeoNameIdentifier',
    'CountryIdentifier'
]
```

In [3]:
```
train_1 = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS1)
train_2 = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS2)
train_3 = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS3)
train_4 = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS4)
train_5 = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS5)
```

In [4]:
```
train_1.head()
```

Out[4]:

	AVProductStatesIdentifier	AVProductsInstalled	AVProductsEnabled	HasDetections
0	53447.0	1.0	1.0	0
1	53447.0	1.0	1.0	0
2	53447.0	1.0	1.0	0
3	53447.0	1.0	1.0	1
4	53447.0	1.0	1.0	1

In [28]:
```
train_1.describe()
```

Out[28]:

	AVProductStatesIdentifier	AVProductsInstalled	AVProductsEnabled	HasDetections
count	8.885262e+06	8.885262e+06	8.885262e+06	8.921483e+06
mean	4.784001e+04	1.326779e+00	1.020967e+00	4.997927e-01
std	1.403237e+04	5.229272e-01	1.675544e-01	5.000000e-01
min	3.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	4.948000e+04	1.000000e+00	1.000000e+00	0.000000e+00
50%	5.344700e+04	1.000000e+00	1.000000e+00	0.000000e+00
75%	5.344700e+04	2.000000e+00	1.000000e+00	1.000000e+00
max	7.050700e+04	7.000000e+00	5.000000e+00	1.000000e+00

In [ ]:
```

```

In [5]:
```
train_2.head()
```

Out[5]:

	Platform	OsBuild	HasDetections
0	windows10	17134	0
1	windows10	17134	0
2	windows10	17134	0
3	windows10	17134	1
4	windows10	17134	1

In [29]:
```
train_2.describe()
```

Out[29]:

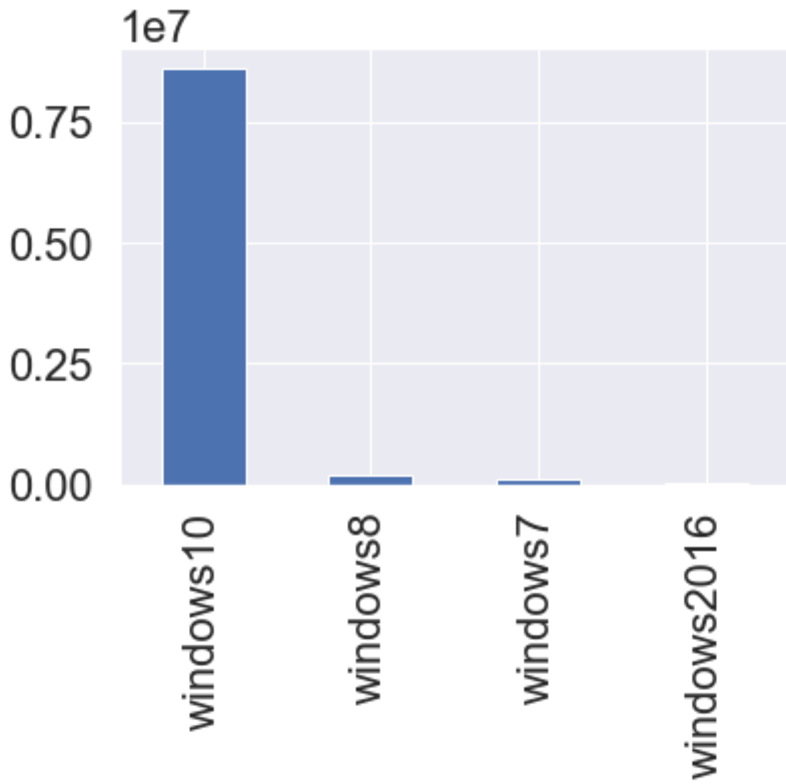
	OsBuild	HasDetections
count	8.921483e+06	8.921483e+06
mean	1.571997e+04	4.997927e-01
std	2.190685e+03	5.000000e-01
min	7.600000e+03	0.000000e+00
25%	1.506300e+04	0.000000e+00
50%	1.629900e+04	0.000000e+00
75%	1.713400e+04	1.000000e+00
max	1.824400e+04	1.000000e+00

In [30]:
```
train_2.Platform.value_counts()
```

Out[30]:
```
windows10      8618715
windows8        194508
windows7         93889
windows2016      14371
Name: Platform, dtype: int64
```

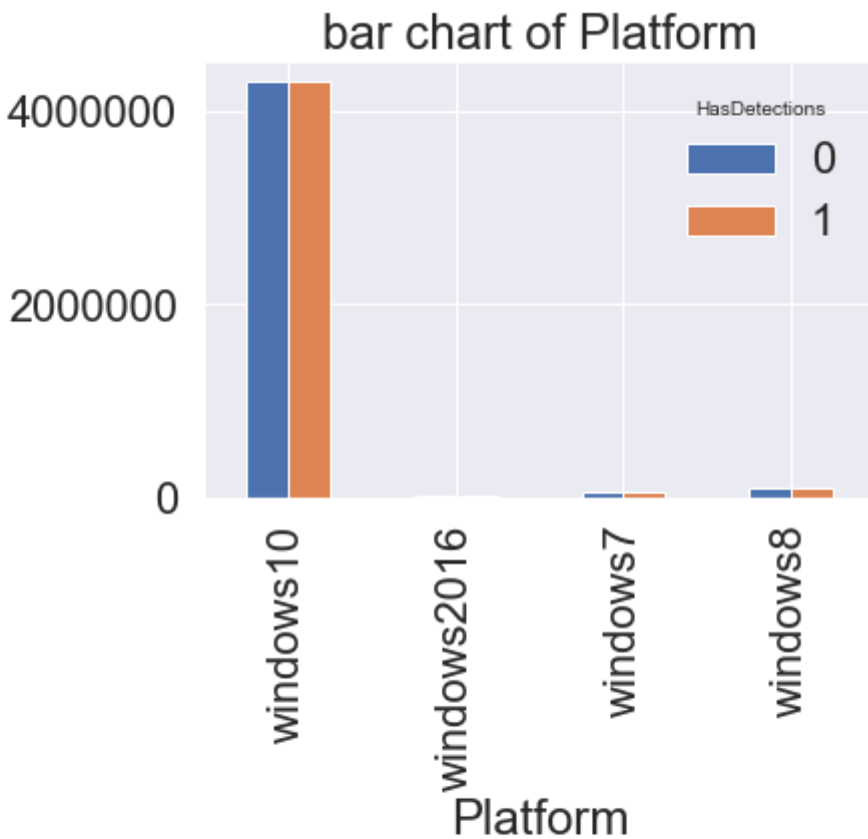
In [31]:
```
train_2.Platform.value_counts().plot(kind = 'bar')
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x18391944710>
```



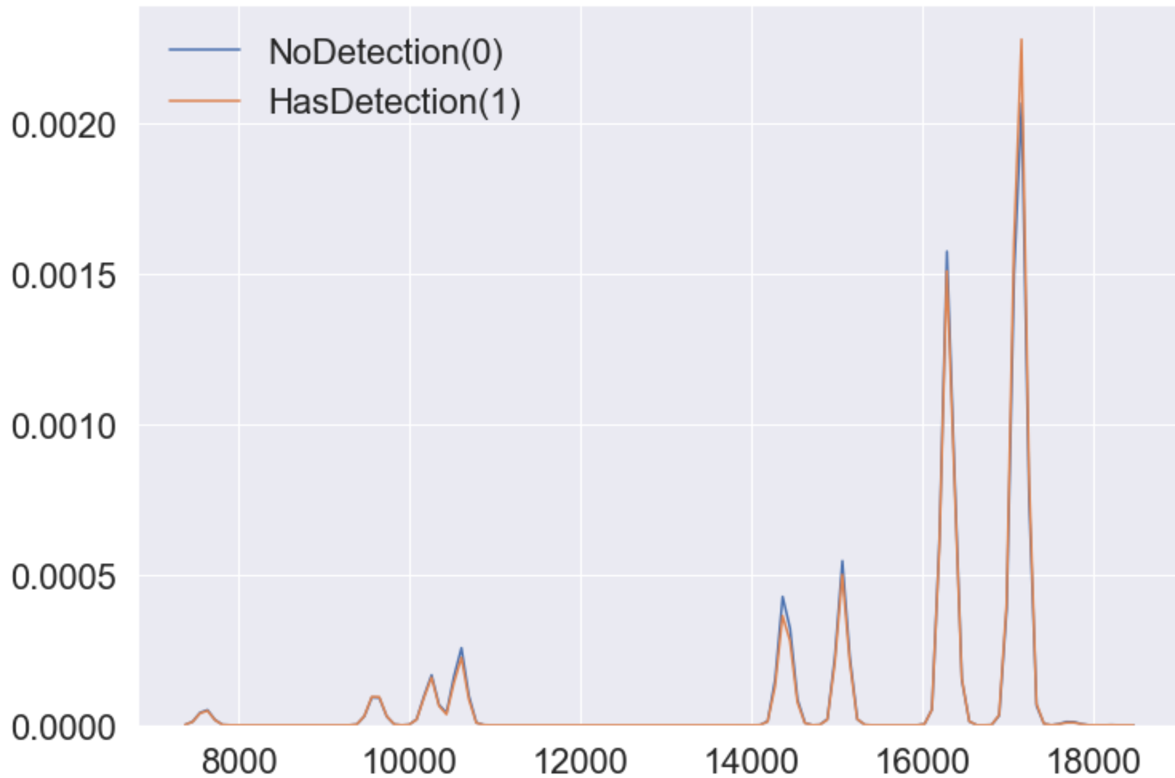
In [23]:
```
train_2.pivot_table(index = 'Platform', columns = 'HasDetections', aggfu
nc = 'size').plot(kind = 'bar')
plt.title('bar chart of {}'.format('Platform'))
```

Out[23]:
```
Text(0.5,1,'bar chart of Platform')
```



In [14]:
```
fig, ax = plt.subplots(figsize=(11.7, 8.27))
sns.kdeplot(train_2.loc[train_2['HasDetections'] == 0, 'OsBuild'], label
='NoDetection(0)')
sns.kdeplot(train_2.loc[train_2['HasDetections'] == 1, 'OsBuild'], label
='HasDetection(1)')
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2868e0d2128>



In [ ]:

In [ ]:

In [9]:
```
train_3.head()
```

Out[9]:

	Processor	Census_ProcessorCoreCount	Census_PrimaryDiskTotalCapacity	HasDetections
0	x64	4.0	476940.0	0
1	x64	4.0	476940.0	0
2	x64	4.0	114473.0	0
3	x64	4.0	238475.0	1
4	x64	4.0	476940.0	1

In [32]:
```
train_3.describe()
```

Out[32]:

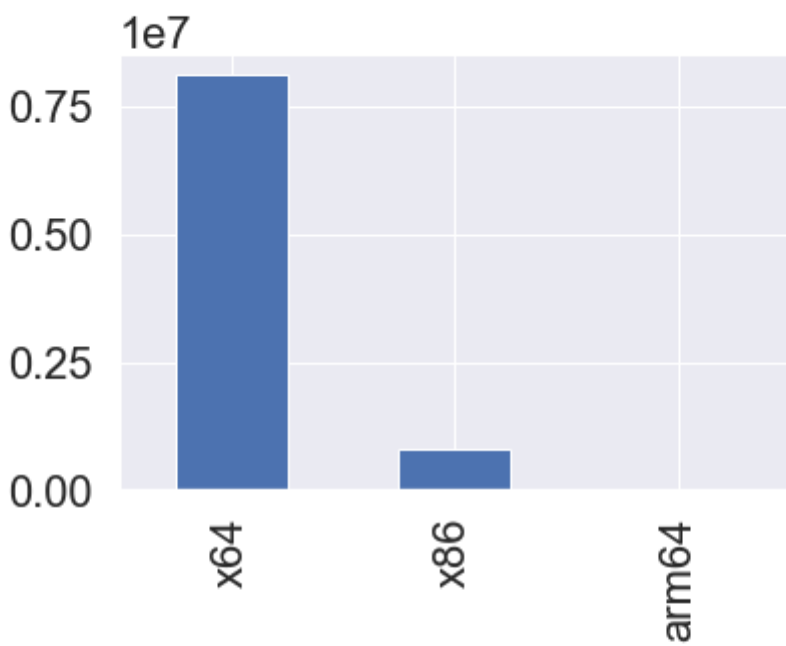
	Census_ProcessorCoreCount	Census_PrimaryDiskTotalCapacity	HasDetections
count	8.880177e+06	8.868467e+06	8.921483e+06
mean	3.989696e+00	3.089053e+06	4.997927e-01
std	2.082553e+00	4.451634e+09	5.000000e-01
min	1.000000e+00	0.000000e+00	0.000000e+00
25%	2.000000e+00	2.393720e+05	0.000000e+00
50%	4.000000e+00	4.769400e+05	0.000000e+00
75%	4.000000e+00	9.538690e+05	1.000000e+00
max	1.920000e+02	8.160437e+12	1.000000e+00

In [33]:
```
train_3.Processor.value_counts()
```

Out[33]:
```
x64      8105435
x86       815702
arm64        346
Name: Processor, dtype: int64
```

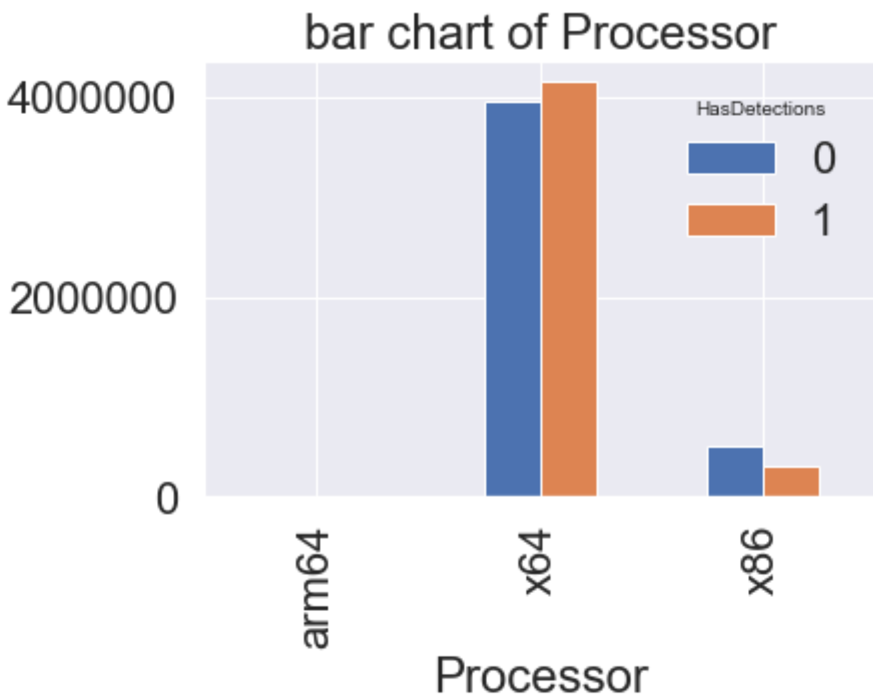
In [34]:
```
train_3.Processor.value_counts().plot(kind = 'bar')
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x18391999dd8>
```



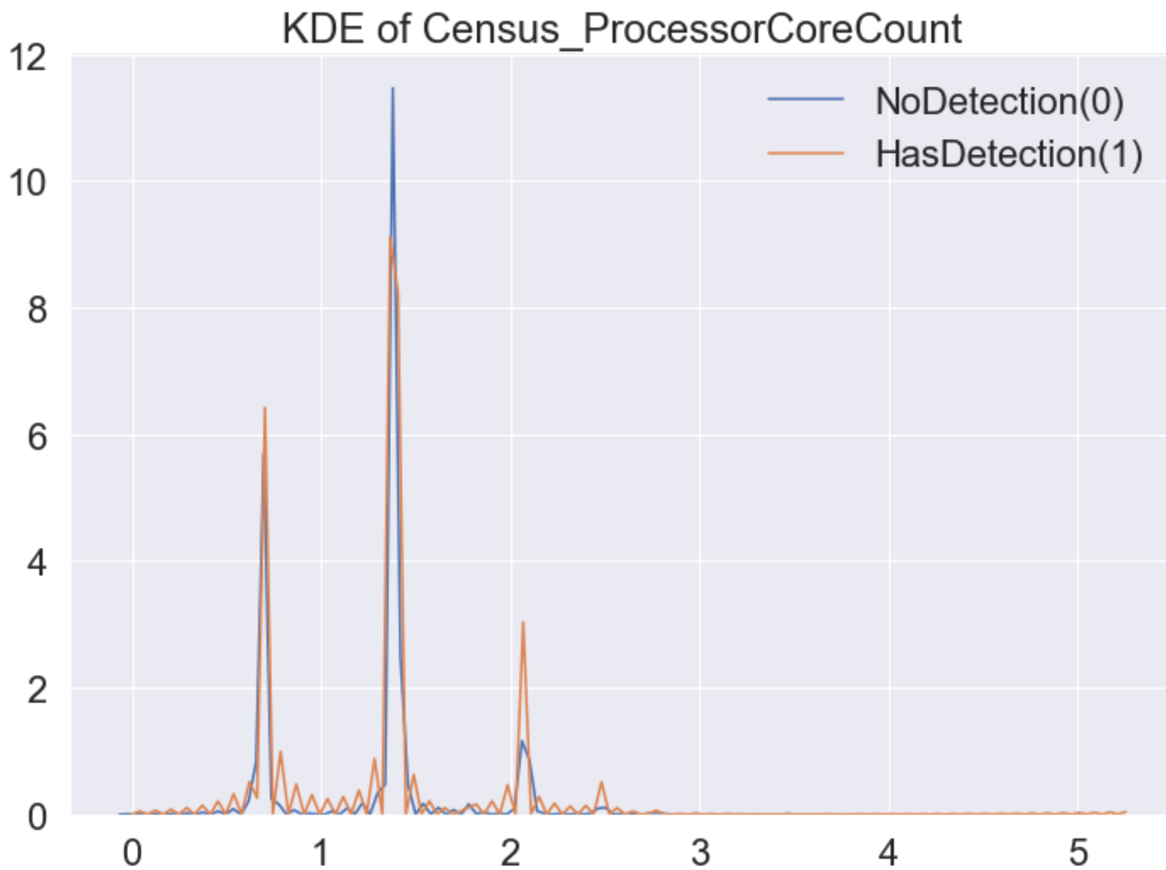
In [24]:
```
train_3.pivot_table(index = 'Processor', columns = 'HasDetections', aggf
unc = 'size').plot(kind = 'bar')
plt.title('bar chart of {}'.format('Processor'))
```

Out[24]:
```
Text(0.5,1,'bar chart of Processor')
```



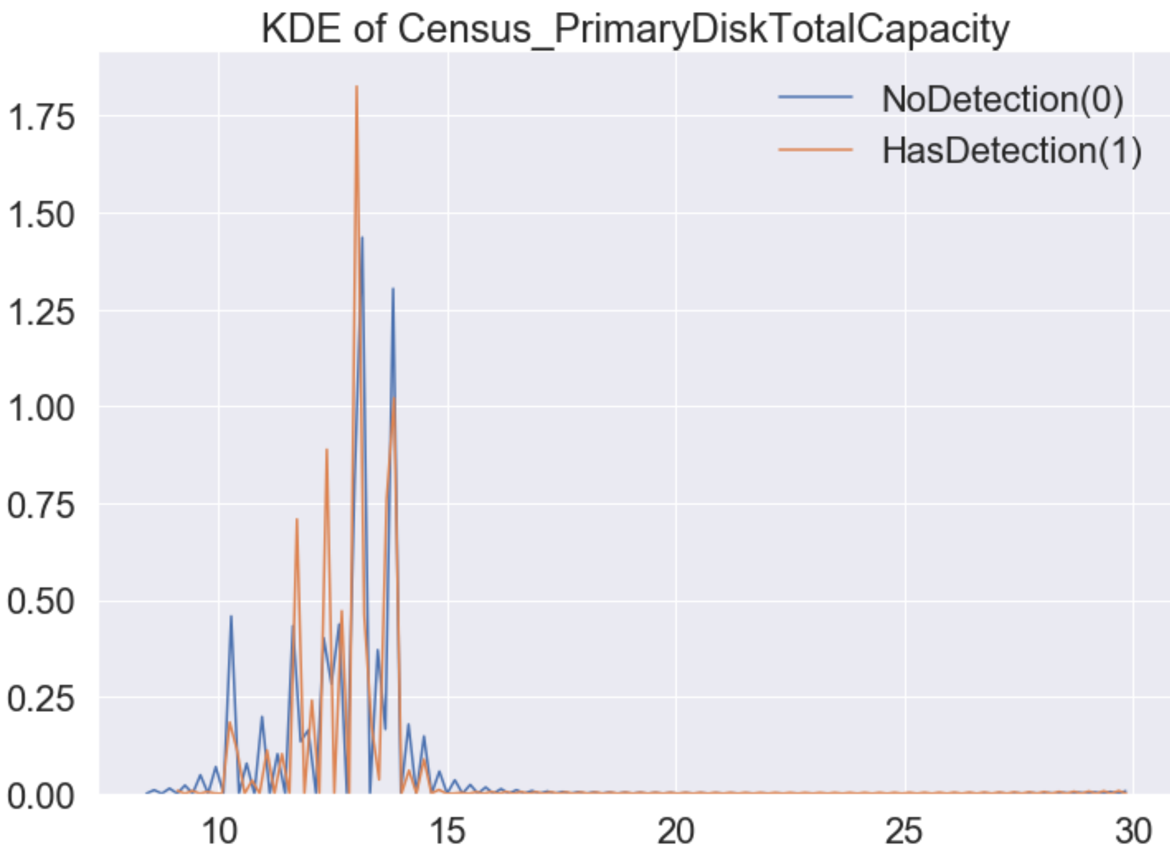
In [69]:
```
fig, ax = plt.subplots(figsize=(11.7, 8.27))
sns.kdeplot(np.log(train_3.loc[train_3['HasDetections'] == 0, 'Census_Pr
ocessorCoreCount']), label='NoDetection(0)')
sns.kdeplot(np.log(train_3.loc[train_3['HasDetections'] == 1, 'Census_Pr
ocessorCoreCount']), label='HasDetection(1)')
plt.title('KDE of {}'.format('Census_ProcessorCoreCount'))
```

Out[69]:
```
Text(0.5,1,'KDE of Census_ProcessorCoreCount')
```



In [70]:
```
fig, ax = plt.subplots(figsize=(11.7, 8.27))
sns.kdeplot(np.log(train_3.loc[train_3['HasDetections'] == 0, 'Census_Pr
imaryDiskTotalCapacity']), label='NoDetection(0)')
sns.kdeplot(np.log(train_3.loc[train_3['HasDetections'] == 1, 'Census_Pr
imaryDiskTotalCapacity']), label='HasDetection(1)')

plt.title('KDE of {}'.format('Census_PrimaryDiskTotalCapacity'))
```

Out[70]: Text(0.5,1,'KDE of Census_PrimaryDiskTotalCapacity')



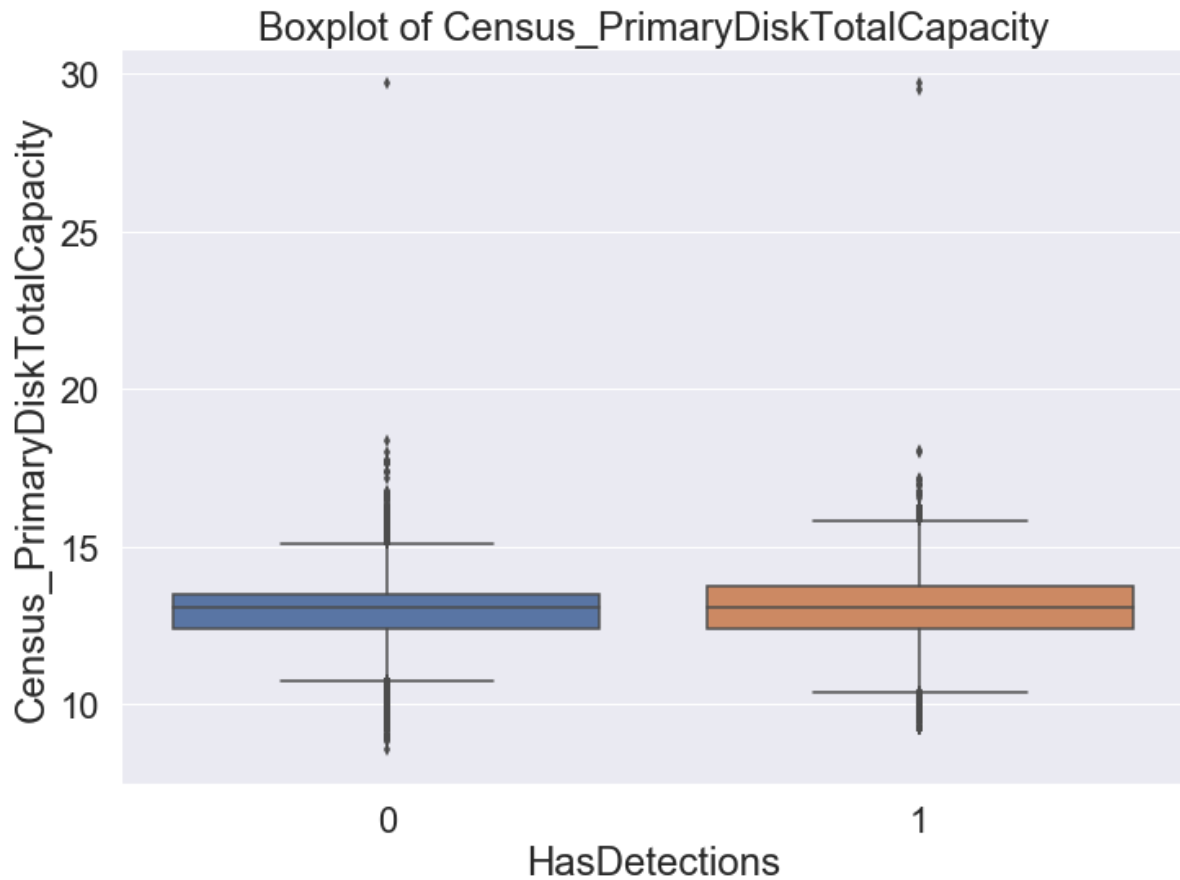
In [71]:
```
log_train_3 = train_3.copy()
log_train_3['Census_PrimaryDiskTotalCapacity'] = np.log(log_train_3['Cen
sus_PrimaryDiskTotalCapacity'])
```

In [72]:
```
# 16TB = 16777216MB which is the largest capacity available, we use it a
s the cutoff to avoid outliers
np.log(16777216)
```

Out[72]: 16.635532333438686

In [74]:
```
fig, ax = plt.subplots(figsize=(11.7, 8.27))
ax = sns.boxplot(data=log_train_3, x='HasDetections', y='Census_PrimaryD
iskTotalCapacity')

plt.title('Boxplot of {}'.format('Census_PrimaryDiskTotalCapacity'))
```

Out[74]: Text(0.5,1,'Boxplot of Census_PrimaryDiskTotalCapacity')



In [ ]:

In [ ]:

In [ ]:

In [10]:
```
train_4.head()
```

Out[10]:

	ProductName	IsBeta	HasDetections
0	win8defender	0	0
1	win8defender	0	0
2	win8defender	0	0
3	win8defender	0	1
4	win8defender	0	1

In [35]:
```
train_4.describe()
```

Out[35]:

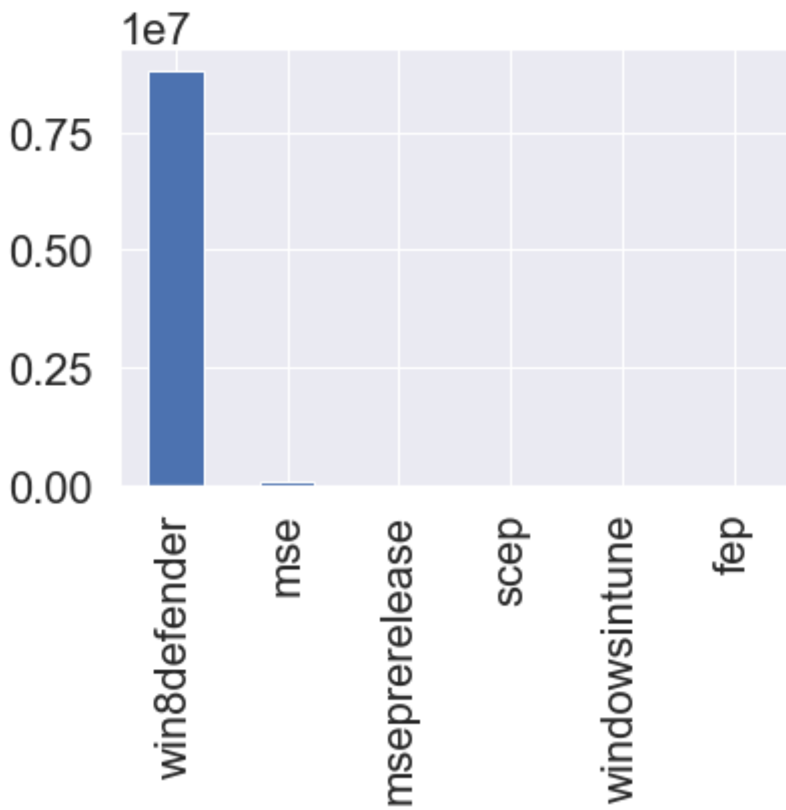
	IsBeta	HasDetections
count	8.921483e+06	8.921483e+06
mean	7.509962e-06	4.997927e-01
std	2.740421e-03	5.000000e-01
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	0.000000e+00	1.000000e+00
max	1.000000e+00	1.000000e+00

In [36]:
```
train_4.ProductName.value_counts()
```

Out[36]:
```
win8defender     8826520
mse                94873
mseprerelease         53
scep                  22
windowsintune          8
fep                    7
Name: ProductName, dtype: int64
```

In [38]:
```
train_4.ProductName.value_counts().plot(kind = 'bar')
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x18391a311d0>

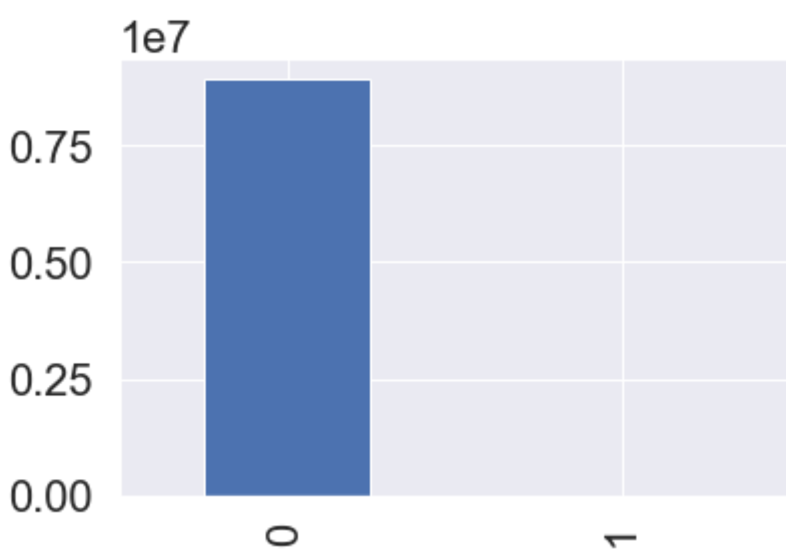


In [39]:
```
train_4.IsBeta.value_counts()
```

Out[39]:
```
0    8921416
1         67
Name: IsBeta, dtype: int64
```

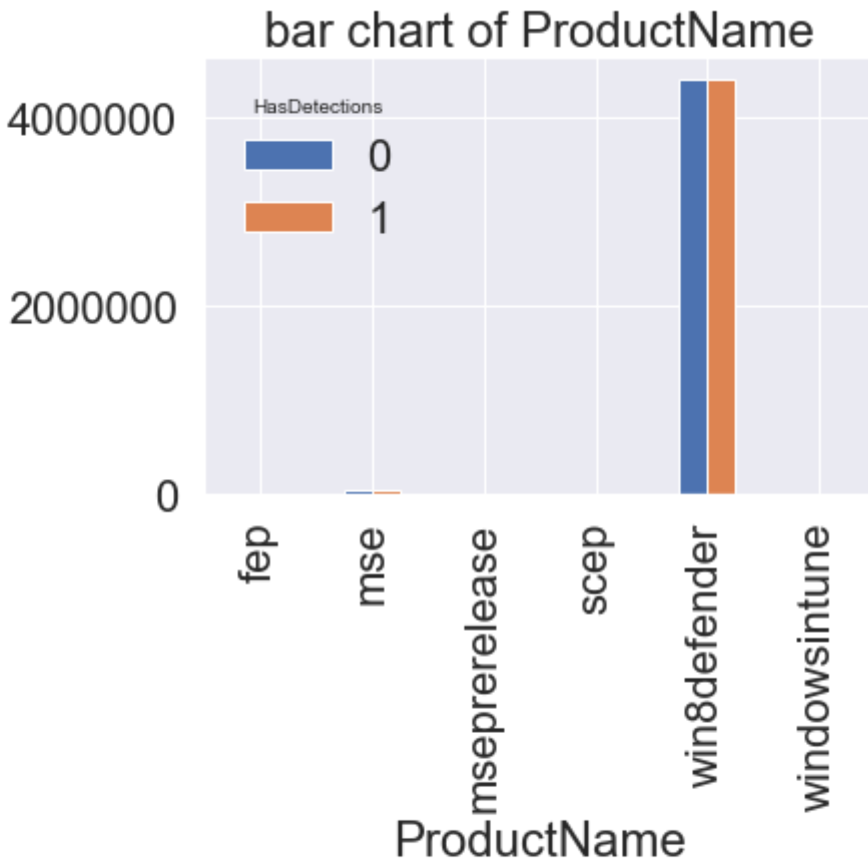
In [40]:
```
train_4.IsBeta.value_counts().plot(kind = 'bar')
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x18391a89e80>



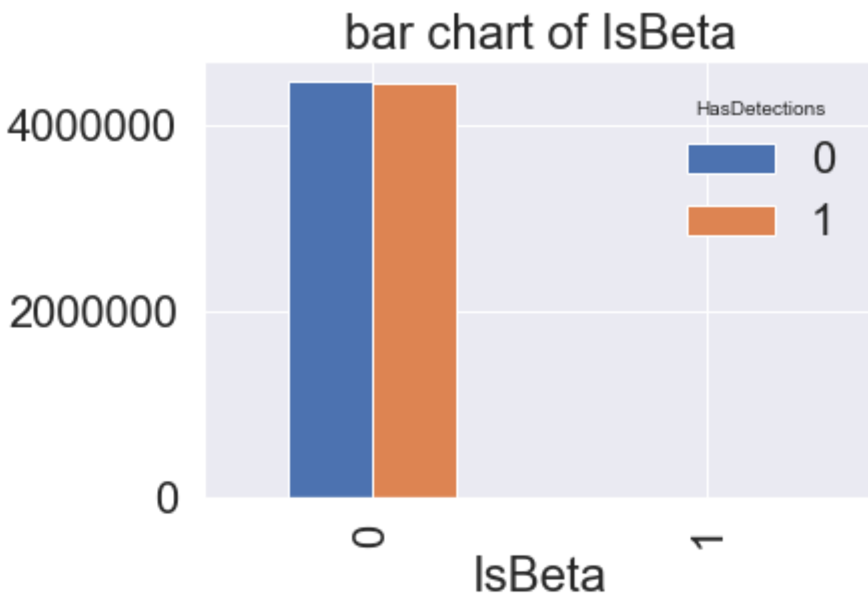
In [26]:
```
train_4.pivot_table(index = 'ProductName', columns = 'HasDetections', ag
gfunc = 'size').plot(kind = 'bar')
plt.title('bar chart of {}'.format('ProductName'))
```

Out[26]:
```
Text(0.5,1,'bar chart of ProductName')
```



In [27]:
```
train_4.pivot_table(index = 'IsBeta', columns = 'HasDetections', aggfunc
= 'size').plot(kind = 'bar')
plt.title('bar chart of {}'.format('IsBeta'))
```

Out[27]: Text(0.5,1,'bar chart of IsBeta')



In [ ]:
```
#special analysis of isbeta
```

In [46]:
```
len(train_4[ (train_4.HasDetections == 1) & (train_4.IsBeta == 1)]) / (l
en(train_4[ (train_4.IsBeta == 1)]))
```

Out[46]: 0.4925373134328358

In [47]:
```
len(train_4[ (train_4.HasDetections == 1) & (train_4.IsBeta == 0)]) / (l
en(train_4[ (train_4.IsBeta == 0)]))
```

Out[47]: 0.4997927459049102

In [48]:
```
# same, isbeta ignored
```

In [11]:
```
train_5.head()
```

Out[11]:

	CountryIdentifier	GeoNameIdentifier	HasDetections
0	29	35.0	0
1	93	119.0	0
2	86	64.0	0
3	88	117.0	1
4	18	277.0	1

In [49]:
```
train_5.CountryIdentifier.value_counts()
```

Out[49]:
```
43     397172
29     347991
141    333411
93     283625
171    280572
60     231981
201    220622
207    211645
66     208579
89     200516
97     195161
214    191269
158    184766
44     182707
9      172594
107    168997
41     160533
68     160158
51     159940
203    158058
35     140027
160    132251
142    131907
195    131685
149    129578
205    117245
155    110779
164    108549
173     94129
159     91592
          ...
74        775
192       740
182       696
134       689
196       681
198       656
123       654
75        643
114       590
126       566
64        565
28        553
215       543
105       507
5         459
174       449
14        446
79        444
187       438
216       379
200       355
10        327
128       303
212       299
186       227
165       213
```

```
37         212
193        207
161        206
217        120
Name: CountryIdentifier, Length: 222, dtype: int64
```

In [54]:
```
train_5.CountryIdentifier.nunique()
```

Out[54]:
```
222
```

In [55]:
```
#222 countries
```

In [53]:
```
train_5.GeoNameIdentifier.value_counts()
```

Out[53]:
```
277.0    1531929
211.0     423166
53.0      408807
89.0      360798
240.0     346568
35.0      345904
167.0     339845
276.0     296774
267.0     215812
126.0     198021
98.0      184459
119.0     181876
138.0     172941
255.0     162193
57.0      155478
10.0      143023
52.0      140200
204.0     137451
120.0     128907
181.0     127368
45.0      114902
205.0     114506
202.0     112056
224.0     101510
157.0      99616
201.0      92651
117.0      89426
258.0      85291
129.0      84929
15.0       78629
          ...
215.0         27
231.0         19
37.0          18
95.0          14
292.0         13
217.0         12
259.0         11
124.0          9
249.0          7
242.0          6
280.0          6
169.0          5
116.0          5
260.0          5
290.0          4
219.0          3
265.0          3
136.0          3
210.0          2
72.0           2
278.0          1
55.0           1
92.0           1
106.0          1
51.0           1
13.0           1
```

```
14.0             1
197.0            1
279.0            1
132.0            1
Name: GeoNameIdentifier, Length: 292, dtype: int64
```

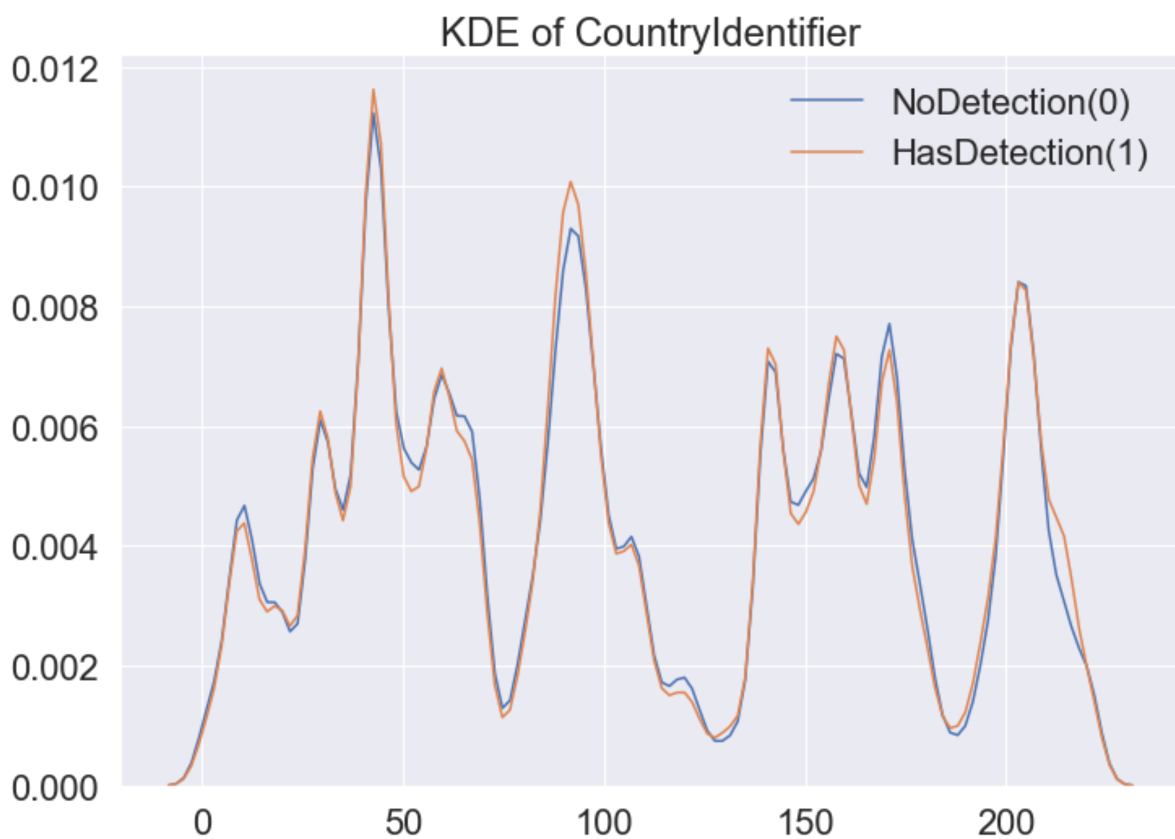
In [56]:
```
train_5.GeoNameIdentifier.nunique()
```

Out[56]:
```
292
```

In [57]:
```
# 292 Geonames
```

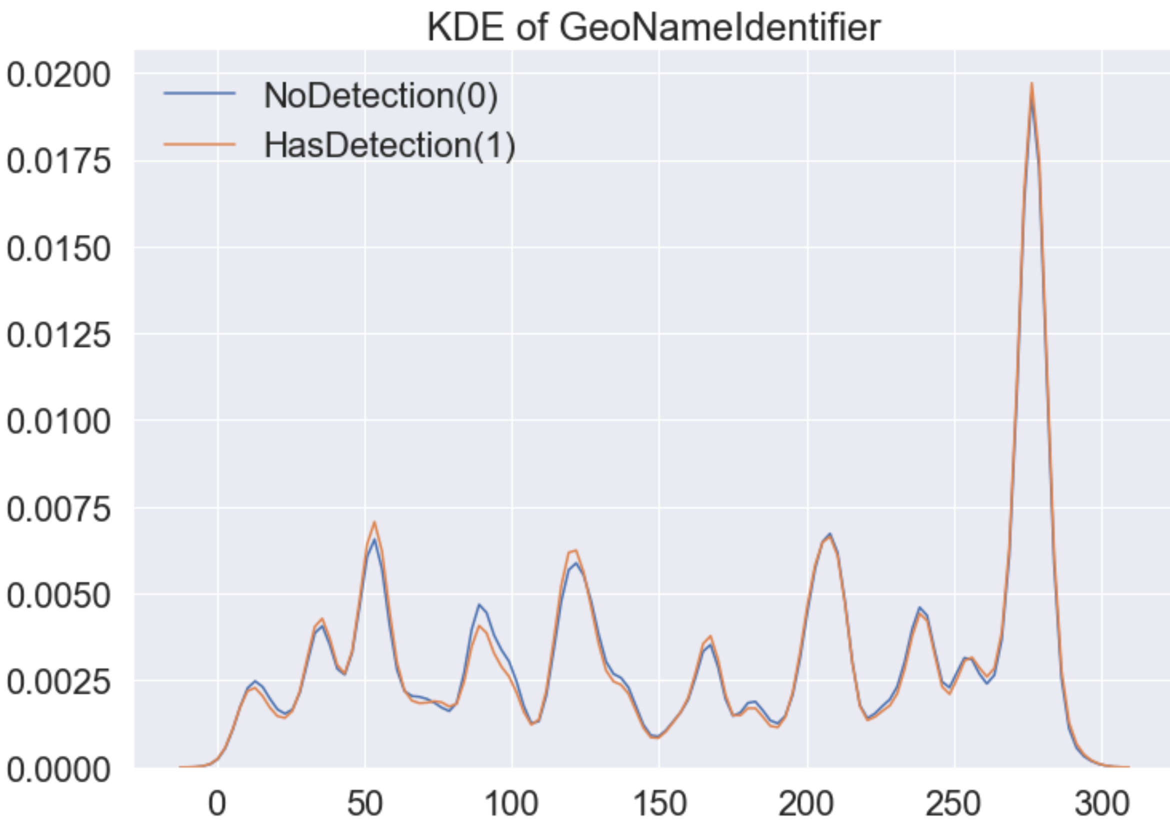
In [75]:
```
fig, ax = plt.subplots(figsize=(11.7, 8.27))
sns.kdeplot(train_5.loc[train_5['HasDetections'] == 0, 'CountryIdentifie
r'], label='NoDetection(0)')
sns.kdeplot(train_5.loc[train_5['HasDetections'] == 1, 'CountryIdentifie
r'], label='HasDetection(1)')

plt.title('KDE of {}'.format('CountryIdentifier'))
```

Out[75]:
```
Text(0.5,1,'KDE of CountryIdentifier')
```



In [78]:
```
fig, ax = plt.subplots(figsize=(11.7, 8.27))
sns.kdeplot(train_5.loc[train_5['HasDetections'] == 0, 'GeoNameIdentifie
r'], label='NoDetection(0)')
sns.kdeplot(train_5.loc[train_5['HasDetections'] == 1, 'GeoNameIdentifie
r'], label='HasDetection(1)')
plt.title('KDE of {}'.format('GeoNameIdentifier'))
```

Out[78]: Text(0.5,1,'KDE of GeoNameIdentifier')



In [22]:

In [ ]:

In [ ]:

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier, GradientBoostingCla
ssifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import LinearSVC

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
COLS = [
      'HasDetections',
     'AVProductStatesIdentifier',
     'AVProductsInstalled',
     'GeoNameIdentifier',
     'CountryIdentifier',
     'OsBuild',
     'Census_ProcessorCoreCount',
     'Census_PrimaryDiskTotalCapacity',
     'Processor'
]
```

In [3]:
```
train = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS)
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                     , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [5]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'])
```

In [6]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                            remainder='drop')
    clf = RandomForestClassifier(n_estimators=7, max_depth=60)
    pl = Pipeline(steps=[('preprocessor', preproc),
                        ('clf', clf)
                        ])
    return pl
```

In [ ]:
```
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['LinearSVC', 'Observation accuracy'] = pred_score
output.loc['LinearSVC', 'Random_Data accuracy'] = rand_score
```

In [ ]:
```
output
```

In [ ]:
```

```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier, GradientBoostingCla
ssifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import LinearSVC

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
COLS = [
    'HasDetections',
    'AVProductStatesIdentifier',
    'AVProductsInstalled',
    'GeoNameIdentifier',
    'CountryIdentifier',
    'OsBuild',
    'Census_ProcessorCoreCount',
    'Census_PrimaryDiskTotalCapacity',
    'Processor'
]
```

In [3]:
```
train = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS)
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                    , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [5]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'])
```

In [6]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                          remainder='drop')
    clf = SGDClassifier()
    pl = Pipeline(steps=[('preprocessor', preproc),
                    ('clf', clf)
                    ])
    return pl
```

In [ ]:
```
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['SGDClassifier', 'Observation accuracy'] = pred_score
output.loc['SGDClassifier', 'Random_Data accuracy'] = rand_score
```

In [ ]:
```
output
```

In [ ]:
```

```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier, GradientBoostingCla
ssifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import LinearSVC

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [2]:
```
COLS = [
    'HasDetections',
    'AVProductStatesIdentifier',
    'AVProductsInstalled',
    'GeoNameIdentifier',
    'CountryIdentifier',
    'OsBuild',
    'Census_ProcessorCoreCount',
    'Census_PrimaryDiskTotalCapacity',
    'Processor'
]
```

In [3]:
```
train = pd.read_csv("train.csv", sep=',', engine='c', usecols=COLS)
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                    , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [5]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'])
```

In [6]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                          remainder='drop')
    clf = SGDClassifier()
    pl = Pipeline(steps=[('preprocessor', preproc),
                    ('clf', clf)
                    ])
    return pl
```

In [ ]:
```
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['SGDClassifier', 'Observation accuracy'] = pred_score
output.loc['SGDClassifier', 'Random_Data accuracy'] = rand_score
```

In [ ]:
```
output
```

In [ ]:
```
```

In [ ]:
```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
#from sklearn.impute import SimpleImputer
import pandas as pd
import numpy as np
import lightgbm as lgb
```

In [ ]:
```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import lightgbm as lgb
import time
import datetime

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, KFold, TimeSeriesSp
lit
from sklearn.metrics import mean_squared_error, roc_auc_score
from sklearn.linear_model import LogisticRegression, LogisticRegressionC
V
import gc
from tqdm import tqdm_notebook


import warnings
warnings.filterwarnings("ignore")

import logging
```

In [ ]:
```
#selecting columns we chosde, and ranking them in feature selection mode
l via random forest
```

In [ ]:
```
train = pd.read_csv("train.csv", sep=',', engine='c', keep_default_na =
False)
```

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

In [ ]:
```
clf = RandomForestClassifier(n_estimators=7, max_depth=60)
```

In [ ]:
```
selector = RFE(clf, n_features_to_select=20)
```

In [ ]:
```
y = train['HasDetections']
train = train.drop(['HasDetections', 'MachineIdentifier'], axis=1)
test = test.drop(['MachineIdentifier'], axis=1)
gc.collect()
train.sort_values('AvSigVersion')
train1 = train[:4000000]
train = train[4000000:8000000]

y1 = y[:4000000]
y = y[4000000:8000000]
```

In [ ]:
```
n_fold = 5
folds = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=15)
```

In [ ]:
```
#imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequen
t')
```

In [ ]:
```
onehot = LabelEncoder()
```

In [ ]:
```
X = X.astype(str).apply(LabelEncoder().fit_transform)
```

In [ ]:
```
selector.fit(X, y)
```

In [ ]:
```
selector.verbose
```

In [ ]:
```

```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import lightgbm as lgb
from sklearn.model_selection import KFold
import warnings
import gc
import time
import sys
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
from sklearn import metrics
import scipy.stats as stats

from sklearn.model_selection import permutation_test_score
from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, ClassifierMixin

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier, GradientBoostingCla
ssifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import LinearSVC

plt.style.use('seaborn')
sns.set(font_scale=2)
pd.set_option('display.max_columns', 500)
```

In [ ]:
```
# we selecting top 20 columns from the feature selection model of Recurs
ive feature elimination
COLS = [
    'HasDetections',
    'CountryIdentifier',
    'Census_OSVersion',
    'GeoNameIdentifier',
    'Census_OSBuildRevision',
    'OsBuildLab',
    'LocaleEnglishNameIdentifier',
    'Census_FirmwareManufacturerIdentifier',
    'AppVersion',
    'AVProductStatesIdentifier',
    'SmartScreen',
    'AvSigVersion',
    'Census_OEMModelIdentifier',
    'Census_FirmwareVersionIdentifier',
    'Census_SystemVolumeTotalCapacity',
    'CityIdentifier',
    'Census_OSVersion',
    'EngineVersion',
    'Census_OEMNameIdentifier',
    'Census_ProcessorModelIdentifier',
    'Census_OSInstallTypeName'
]
```

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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(train.dropna().drop(
'HasDetections',axis = 1)\
                                                        , train.dropna()['Ha
sDetections'], test_size=0.25)
N = len(y_test)
y_random = y_test.sample(replace=False, frac = 1)
```

In [ ]:
```
output = pd.DataFrame(columns = ['Observation accuracy', 'Random_Data ac
curacy'])
```

In [ ]:
```
def skl(col):
    nominal_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[('onehot', nominal_transfo
rmer, col)],\
                                          remainder='drop')
    clf = RandomForestClassifier()
    pl = Pipeline(steps=[('preprocessor', preproc),
                    ('clf', clf)
                    ])
    return pl
```

In [ ]:
```
pl = skl(COLS[1:])
pl.fit(X_train, y_train)
pred_score = pl.score(X_test, y_test)
rand_score = pl.score(X_test, y_random)
output.loc['random_forest', 'Observation accuracy'] = pred_score
output.loc['random_forest', 'Random_Data accuracy'] = rand_score
```

In [ ]:
```
output
```

In [ ]:
```
```