Use the "Text" blocks to provide explanations wherever you find them necessary. Highlight your answers inside these text fields to ensure that we don't miss it while grading your HW.

Setup

- Code to download the data directly from the colab notebook.
- If you find it easier to download the data from the kaggle website (and uploading it to your drive), you can skip this section.

In [ ]:
```
!pip install -q kaggle
```

In [ ]:
```
from google.colab import files
# Create a new API token under "Account" in the kaggle webpage and download the json fi
# Upload the file by clicking on the browse
files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[ ]:
```
{'kaggle.json': b'{"username":"harshaldaftar","key":"d278c633f99968d2abc03df76956cca
6"}'}
```

In [ ]:
```
mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json
```

In [ ]:
```
!kaggle competitions download -c microsoft-malware-prediction
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (serv
er 1.5.12 / client 1.5.4)
Downloading test.csv.zip to /content
100% 670M/672M [00:04<00:00, 172MB/s]
100% 672M/672M [00:04<00:00, 171MB/s]
Downloading sample_submission.csv.zip to /content
 97% 130M/134M [00:00<00:00, 184MB/s]
100% 134M/134M [00:00<00:00, 183MB/s]
Downloading train.csv.zip to /content
 98% 750M/768M [00:04<00:00, 190MB/s]
100% 768M/768M [00:04<00:00, 165MB/s]
```

Section 1: Library and Data Imports (Q1)

- Import your libraries and read the data into a dataframe. Print the head of the dataframe.

In [ ]:
```
use_cols = ["MachineIdentifier", "SmartScreen", "AVProductsInstalled", "AppVersion", "C
            "EngineVersion", "AVProductStatesIdentifier", "Census_OSVersion", "Census_To
            "RtpStateBitfield", "Census_ProcessorModelIdentifier", "Census_PrimaryDiskTo
             "Census_InternalPrimaryDiagonalDisplaySizeInInches", "Wdft_RegionIdentifier
            "AvSigVersion", "IeVerIdentifier", "IsProtected", "Census_InternalPrimaryDis
```

```
            "Census_OSWUAutoUpdateOptionsName", "Census_OSEdition", "Census_GenuineStat
           "Census_OEMNameIdentifier", "Census_MDC2FormFactor", "Census_FirmwareManufac
            "Census_OSBuildNumber", "Census_IsPenCapable", "Census_IsTouchEnabled", "Ce
            "Census_SystemVolumeTotalCapacity", "Census_PrimaryDiskTotalCapacity", "Has
           ]
dtypes = {
        'MachineIdentifier':                                    'category',
        'ProductName':                                          'category',
        'EngineVersion':                                        'category',
        'AppVersion':                                           'category',
        'AvSigVersion':                                         'category',
        'IsBeta':                                               'int8',
        'RtpStateBitfield':                                     'float16',
        'IsSxsPassiveMode':                                     'int8',
        'DefaultBrowsersIdentifier':                            'float16',
        'AVProductStatesIdentifier':                            'float32',
        'AVProductsInstalled':                                  'float16',
        'AVProductsEnabled':                                    'float16',
        'HasTpm':                                               'int8',
        'CountryIdentifier':                                    'int16',
        'CityIdentifier':                                       'float32',
        'OrganizationIdentifier':                               'float16',
        'GeoNameIdentifier':                                    'float16',
        'LocaleEnglishNameIdentifier':                          'int8',
        'Platform':                                             'category',
        'Processor':                                            'category',
        'OsVer':                                                'category',
        'OsBuild':                                              'int16',
        'OsSuite':                                              'int16',
        'OsPlatformSubRelease':                                 'category',
        'OsBuildLab':                                           'category',
        'SkuEdition':                                           'category',
        'IsProtected':                                          'float16',
        'AutoSampleOptIn':                                      'int8',
        'PuaMode':                                              'category',
        'SMode':                                                'float16',
        'IeVerIdentifier':                                      'float16',
        'SmartScreen':                                          'category',
        'Firewall':                                             'float16',
        'UacLuaenable':                                         'float32',
        'Census_MDC2FormFactor':                                'category',
        'Census_DeviceFamily':                                  'category',
        'Census_OEMNameIdentifier':                             'float16',
        'Census_OEMModelIdentifier':                            'float32',
        'Census_ProcessorCoreCount':                            'float16',
        'Census_ProcessorManufacturerIdentifier':               'float16',
        'Census_ProcessorModelIdentifier':                      'float16',
        'Census_ProcessorClass':                                'category',
        'Census_PrimaryDiskTotalCapacity':                      'float32',
        'Census_PrimaryDiskTypeName':                           'category',
        'Census_SystemVolumeTotalCapacity':                     'float32',
        'Census_HasOpticalDiskDrive':                           'int8',
        'Census_TotalPhysicalRAM':                              'float32',
        'Census_ChassisTypeName':                               'category',
        'Census_InternalPrimaryDiagonalDisplaySizeInInches':    'float64',
        'Census_InternalPrimaryDisplayResolutionHorizontal':    'float64',
        'Census_InternalPrimaryDisplayResolutionVertical':      'float64',
        'Census_PowerPlatformRoleName':                         'category',
        'Census_InternalBatteryType':                           'category',
        'Census_InternalBatteryNumberOfCharges':                'float32',
        'Census_OSVersion':                                     'category',
```

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

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
sns.set(rc = {'figure.figsize':(15,8)})
```

In [ ]:

```
!unzip train.csv.zip
# !unzip test.csv.zip
```

```
Archive:  train.csv.zip
  inflating: train.csv
```

In [ ]:

```
df = pd.read_csv('train.csv', usecols=use_cols, dtype=dtypes)
```

In [ ]:

```
df.head()
```

Out[ ]:

	MachineIdentifier	EngineVersion	AppVersion	AvSigVersion	RtpStateBitfield	AV
0	0000028988387b115f69f31a3bf04f09	1.1.15100.1	4.18.1807.18075	1.273.1735.0	7.0	
1	000007535c3f730efa9ea0b7ef1bd645	1.1.14600.4	4.13.17134.1	1.263.48.0	7.0	
2	000007905a28d863f6d0d597892cd692	1.1.15100.1	4.18.1807.18075	1.273.1341.0	7.0	

	MachineIdentifier	EngineVersion	AppVersion	AvSigVersion	RtpStateBitfield	AV
3	00000b11598a75ea8ba1beea8459149f	1.1.15100.1	4.18.1807.18075	1.273.1527.0	7.0	
4	000014a5f00daa18e76b81417eeb99fc	1.1.15100.1	4.18.1807.18075	1.273.1379.0	7.0	

In [ ]:
```
df.corr()['HasDetections']
```

Out[ ]:
```
RtpStateBitfield                                 0.041486
AVProductStatesIdentifier                        0.117404
AVProductsInstalled                             -0.149626
CountryIdentifier                                0.007099
LocaleEnglishNameIdentifier                     -0.009981
IsProtected                                      0.057045
IeVerIdentifier                                  0.015907
Census_OEMNameIdentifier                        -0.015541
Census_ProcessorCoreCount                        0.054299
Census_ProcessorModelIdentifier                  0.022711
Census_PrimaryDiskTotalCapacity                  0.000170
Census_SystemVolumeTotalCapacity                 0.014481
Census_TotalPhysicalRAM                          0.057069
Census_InternalPrimaryDiagonalDisplaySizeInInches    0.034243
Census_InternalPrimaryDisplayResolutionVertical      0.013927
Census_OSBuildNumber                             0.029486
Census_OSBuildRevision                          -0.009342
Census_FirmwareManufacturerIdentifier           -0.025924
Census_IsSecureBootEnabled                      -0.001711
Census_IsTouchEnabled                           -0.040410
Census_IsPenCapable                             -0.017177
Census_IsAlwaysOnAlwaysConnectedCapable         -0.062780
Wdft_IsGamer                                     0.053891
Wdft_RegionIdentifier                           -0.022855
HasDetections                                    1.000000
Name: HasDetections, dtype: float64
```

Thus, the baseline model I will use shall have a total of 6 numeric features, which are most likely to have an effect on 'HasDetections'.

1. AVProductsInstalled (number of antivirus products installed)
2. AVProductStatesIdentifier (which antivirus is installed)
3. Census_IsAlwaysOnAlwaysConnectedCapable (whether the battery enables the device to be AlwaysOnAlwaysConnected)
4. IsProtected (returns: a. TRUE if there is at least one active and up-to-date antivirus product running on this machine. b. FALSE if there is no active AV product on this machine, or if the AV is active, but is not receiving the latest updates. c. null if there are no Anti Virus Products in the report. Returns: Whether a machine is protected
5. Census_TotalPhysicalRAM (Retrieves the physical RAM in MB)
6. Census_ProcessorCoreCount (Number of cores in machine)
7. Census_OSBuildNumber (extracted from the OsVersionFull which OS used should have an effect on accuracy).

In [ ]:

```
baseline_cols = ['AVProductsInstalled', 'AVProductStatesIdentifier', 'Census_IsAlwaysOn
```

Section 2: Measure of Power (Q2a & 2b)

Columns to be considered when deciding the power of the PC

1. Census_ProcessorCoreCount (more number of processors means more powerful)
2. Census_TotalPhysicalRAM (more is better) 3.Census_PrimaryDiskTypeName (SSD is more powerful than HDD)
3. Wdft_IsGamer (Gaming machines are more powerful)

In [ ]:
```
for feature in ['Census_ProcessorCoreCount', 'Census_TotalPhysicalRAM', 'Census_Primary
  print(df[feature].value_counts())
  print("\n")
```

```
4.0      5430193
2.0      2311969
8.0       865004
12.0       92702
1.0        70390
6.0        69910
16.0       18551
3.0        13580
32.0        2136
24.0        1847
20.0        1781
40.0         506
36.0         287
28.0         271
48.0         235
5.0          216
56.0         132
10.0          98
64.0          93
7.0           92
72.0          39
88.0          23
14.0          22
80.0          20
44.0          16
30.0          10
9.0            7
112.0          6
96.0           6
18.0           5
22.0           4
11.0           4
52.0           3
46.0           3
128.0          3
104.0          2
15.0           2
26.0           2
144.0          1
54.0           1
192.0          1
50.0           1
120.0          1
13.0           1
25.0           1
```

```
Name: Census_ProcessorCoreCount, dtype: int64


4096.0     4094512
8192.0     2196505
2048.0     1097474
16384.0     531558
6144.0      398671
               ...
6231.0           1
6228.0           1
6225.0           1
6222.0           1
255.0            1
Name: Census_TotalPhysicalRAM, Length: 3446, dtype: int64


HDD            5806804
SSD            2466808
UNKNOWN         358251
Unspecified     276776
Name: Census_PrimaryDiskTypeName, dtype: int64


0.0    6174143
1.0    2443889
Name: Wdft_IsGamer, dtype: int64
```

For processors and RAM, I have taken the log to base of a very low value in each category to get a relative number instead of absolute number. And I added 1 to power if the computer has SSD and 1 if it is a gaming PC. I thought to make code more clean and easy to comprehend but storing each value in unnecessary variables was causing colab to crash

In [ ]:
```
df['Power'] = np.log(df['Census_ProcessorCoreCount']) / np.log(2) + np.log(df['Census_T
```

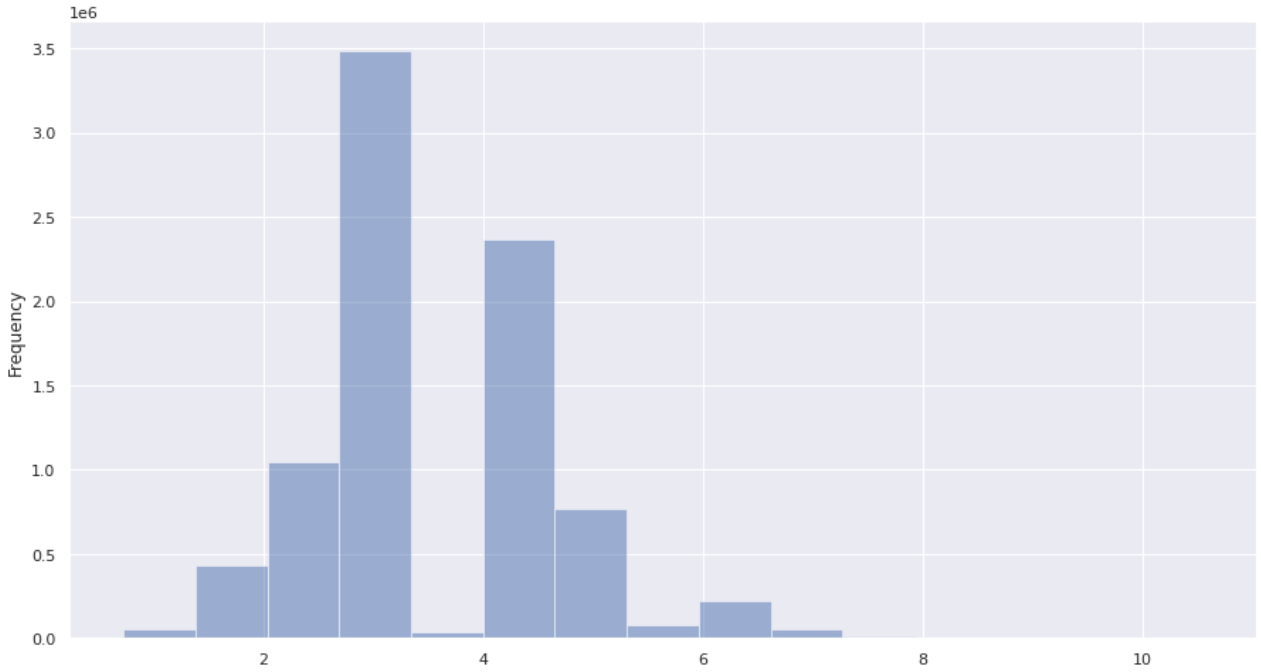
In [ ]:
```
df['Power']
```

Out[ ]:
```
0          3.090909
1          3.090909
2          4.090909
3          3.090909
4          3.144088
              ...
8921478    3.090909
8921479    2.000000
8921480    5.181818
8921481    2.090909
8921482    3.144088
Name: Power, Length: 8921483, dtype: float64
```

Distribution of powerful machines

In [ ]:
```
df['Power'].plot.hist(bins=15, alpha=0.5)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8445f10090>
```



In [ ]:
```
# Checking how many powerful machines have malware. Here, I have considered Power > 4 t
condition_malware_powerful = (df['Power'] >= 4) & (df['HasDetections'] == 1)
np.count_nonzero(condition_malware_powerful) / np.count_nonzero(df['Power'] >= 4)
```

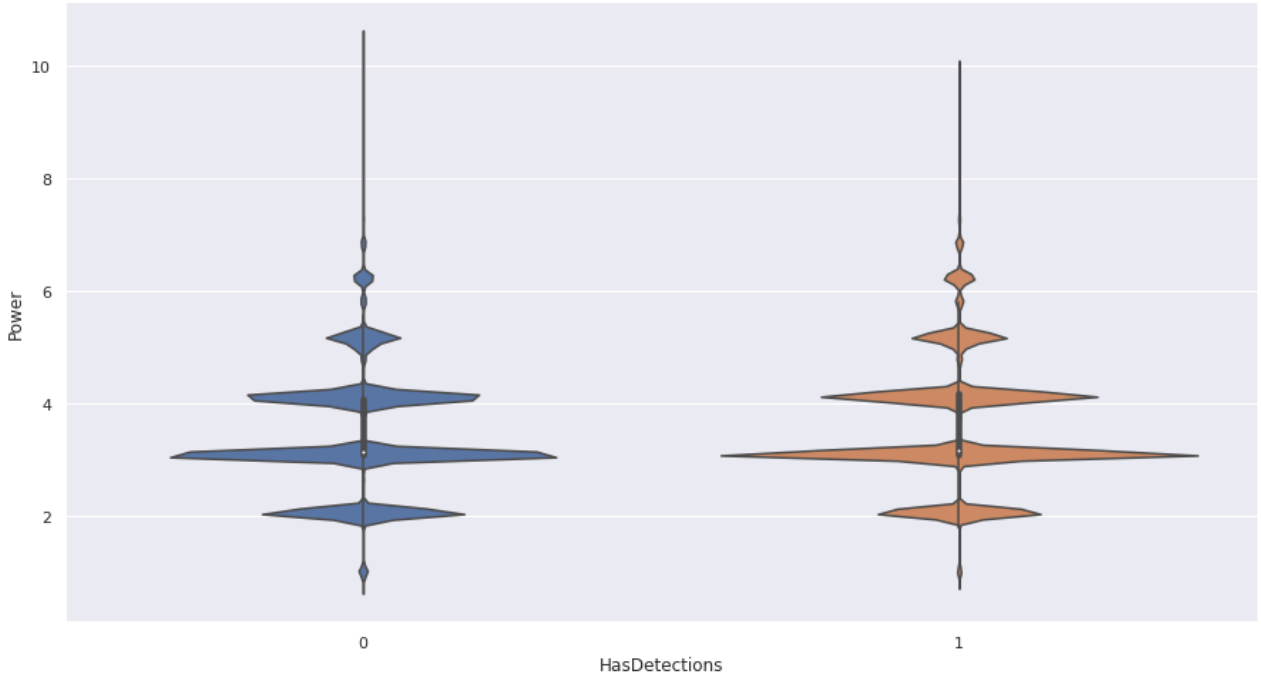
Out[ ]:
```
0.5242807875122147
```

Based on above value, it seems that power of a computer does not determine if a computer is more or less likely to be infected

Plotting

In [ ]:
```
sns.violinplot(x="HasDetections", y="Power", data=df)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8463b5bd90>
```

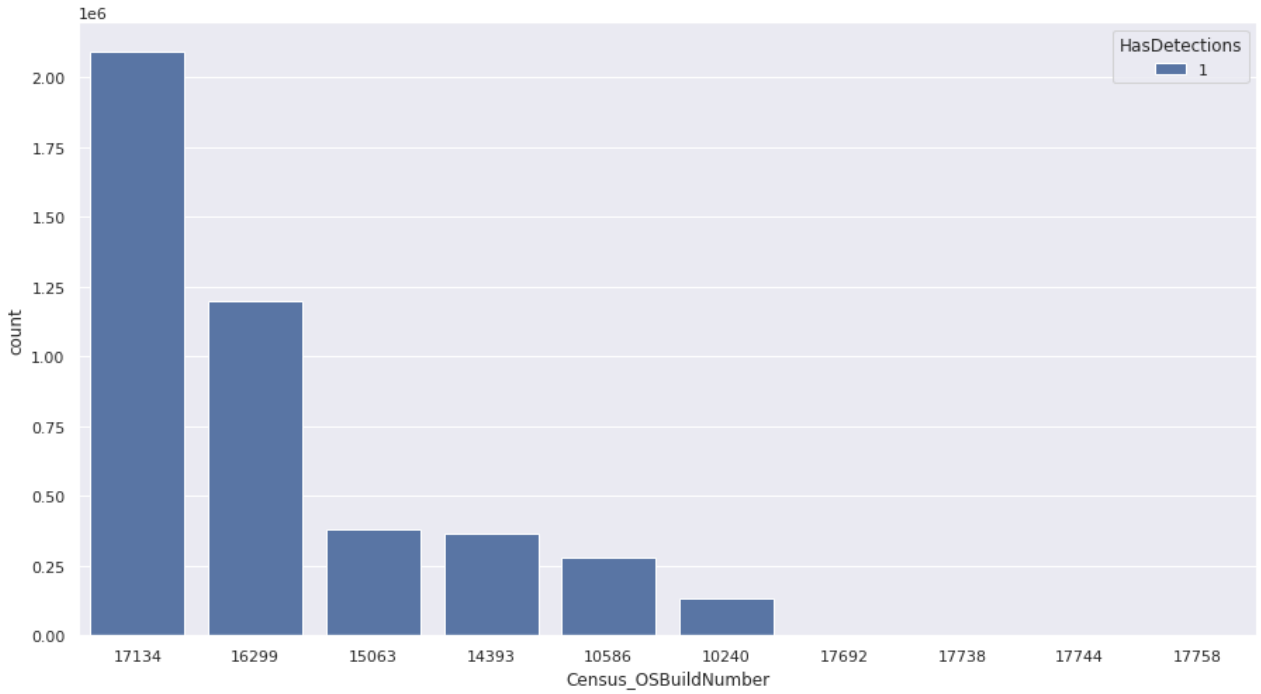


Section 3: OS version vs Malware detected (Q3)

We are not plotting those with value_count very small because it will distort the plot and we cant gain any insights from them

In [ ]:
```
sns.countplot(x="Census_OSBuildNumber", hue="HasDetections", data=df[df['HasDetections'
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8481e34b90>
```



In [ ]:
```
malware_infected_build_number = df.groupby('Census_OSBuildNumber')['HasDetections'].val
malware_infected_build_number[malware_infected_build_number['HasDetections'] == 1]
```

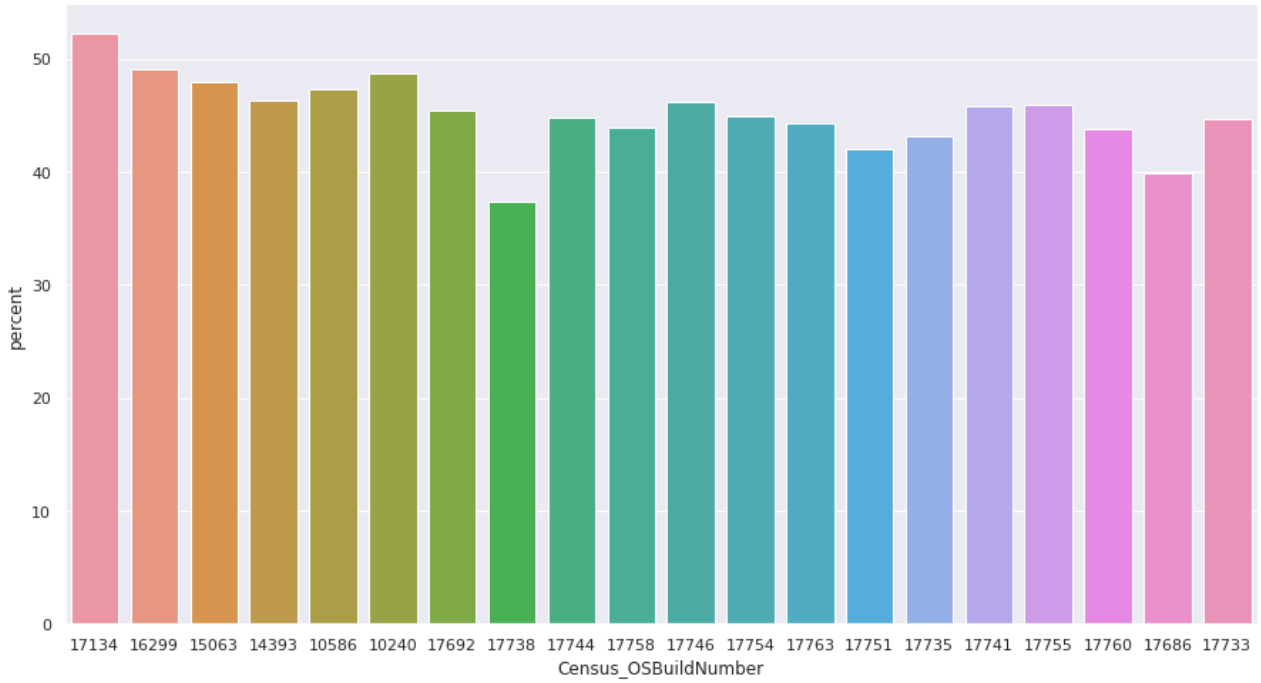
Out[ ]:

	Census_OSBuildNumber	HasDetections	percent
1	7601	1	57.142857
4	9200	1	50.000000
6	9600	1	25.000000
8	10240	1	48.638091
9	10565	1	80.000000
...	...	...	...
245	18219	1	50.000000
246	18224	1	100.000000
252	18234	1	37.768240
255	18237	1	38.728324
260	18242	1	45.774648

125 rows × 3 columns

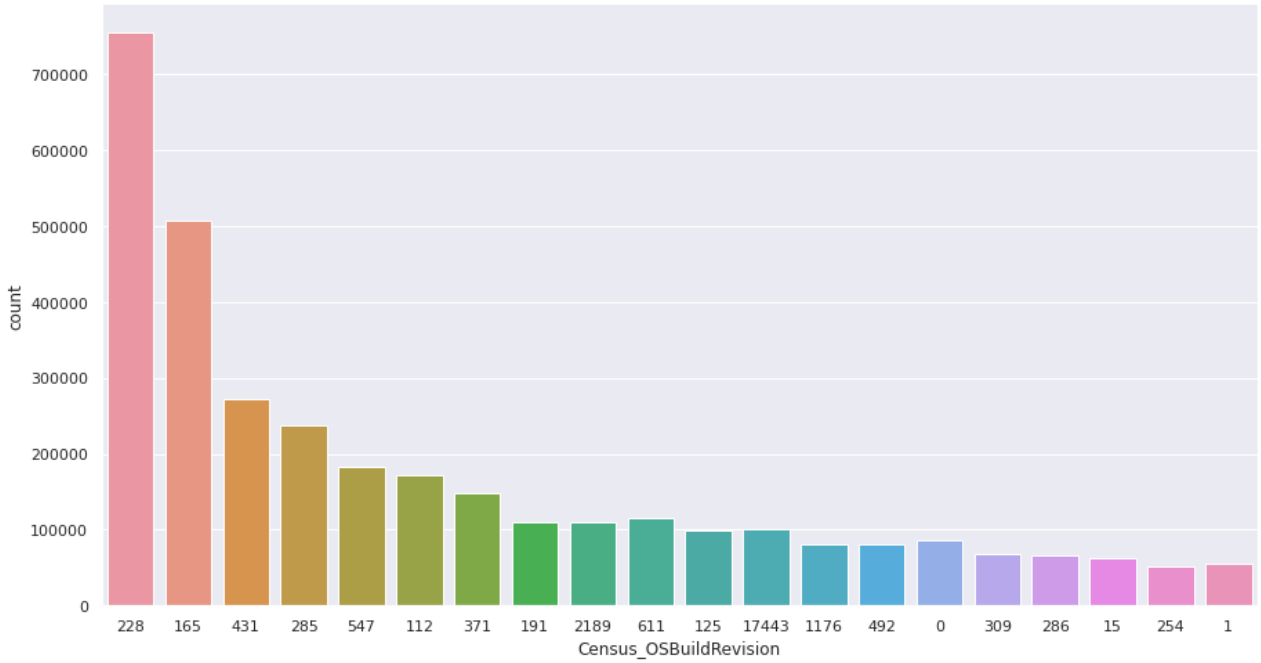
In [ ]:
```
sns.barplot(x="Census_OSBuildNumber", y="percent", data=malware_infected_build_number[m
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f84635e02d0>
```



In [ ]:
```
sns.countplot(x="Census_OSBuildRevision", data=df[df['HasDetections'] == 1], order=df.C
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f848243ce10>
```



In [ ]:
```
malware_infected_build_revision = df.groupby('Census_OSBuildRevision')['HasDetections']
malware_infected_build_revision[malware_infected_build_revision['HasDetections'] == 1]
```

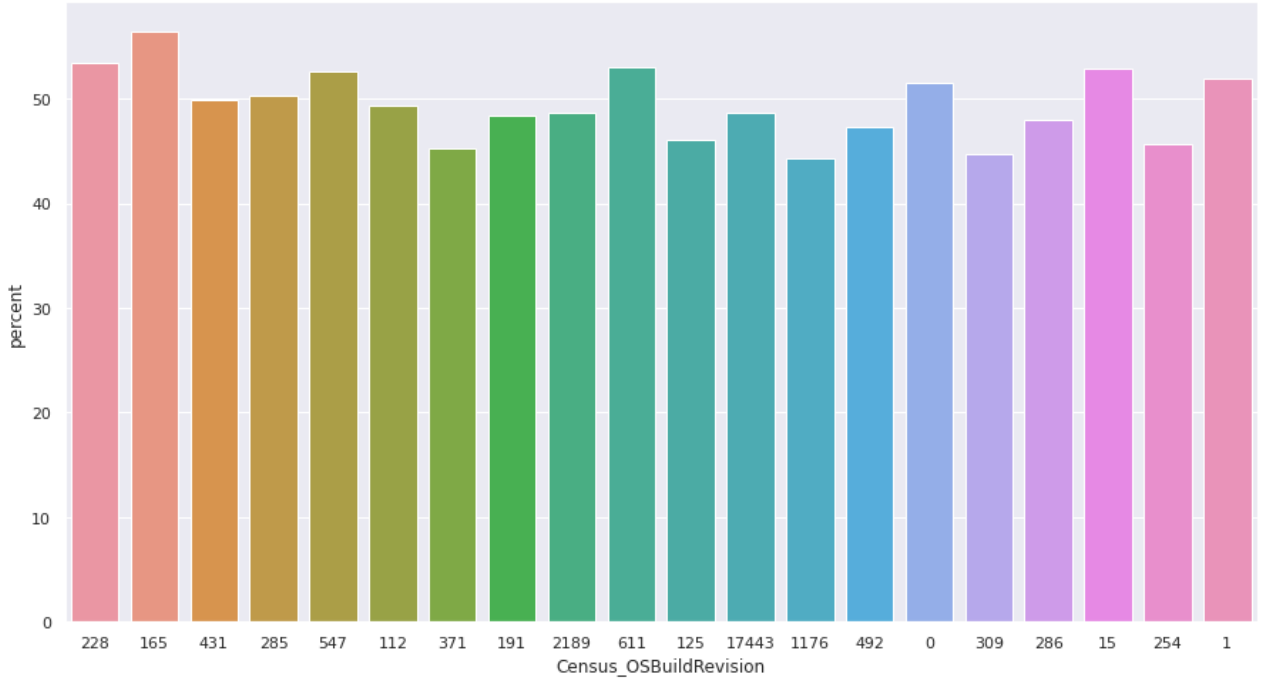
Out[ ]:

	Census_OSBuildRevision	HasDetections	percent
0	0	1	51.481947
2	1	1	51.887262
4	3	1	53.226524
7	4	1	45.785441
9	5	1	41.226913
...	...	...	...
523	19069	1	50.000000
525	21703	1	100.000000
528	24214	1	100.000000
529	24241	1	100.000000
530	41736	1	80.000000

255 rows × 3 columns

In [ ]:
```
sns.barplot(x="Census_OSBuildRevision", y="percent", data=malware_infected_build_revisi
```

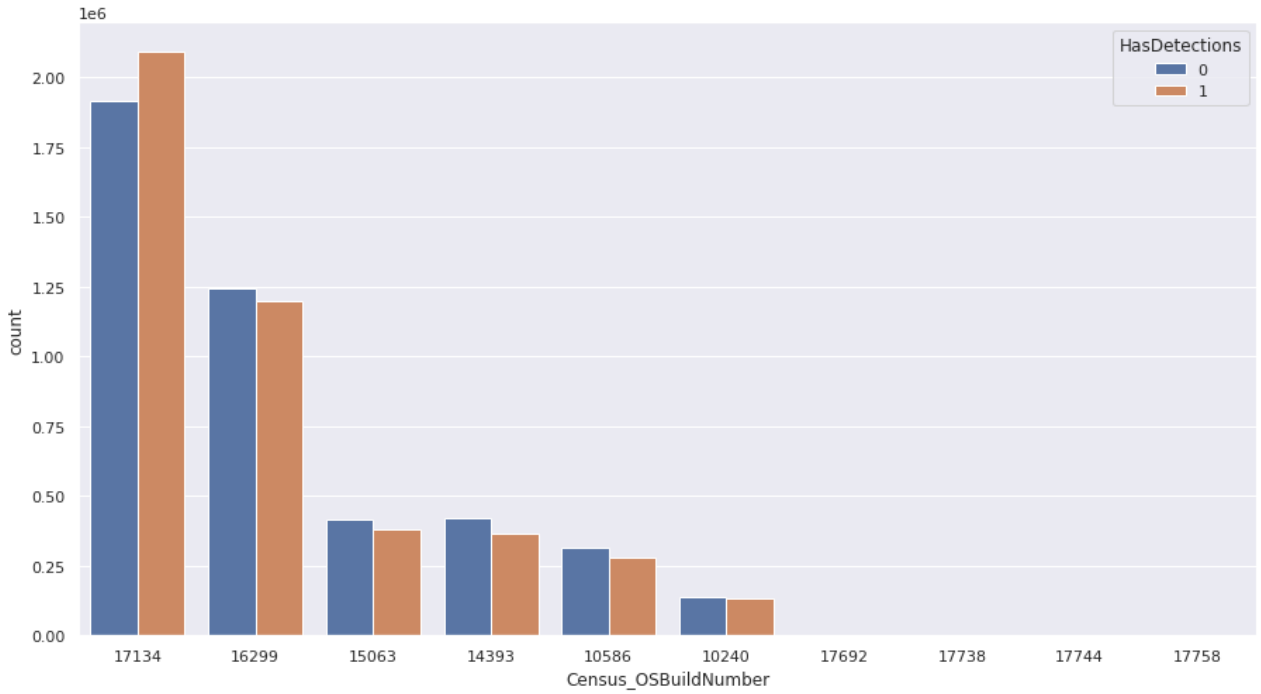
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8464753550>
```



Additional: Comparison between y=0 and y=1 is shown below

In [ ]:
```
# 17134    50
# 16299    23
# 15063     9
# 10586     8
# 14393     7
# 10240     3
# Name: Census_OSBuildNumber, dtype: int64
sns.countplot(x="Census_OSBuildNumber", hue="HasDetections", data=df, order=df.Census_O
```

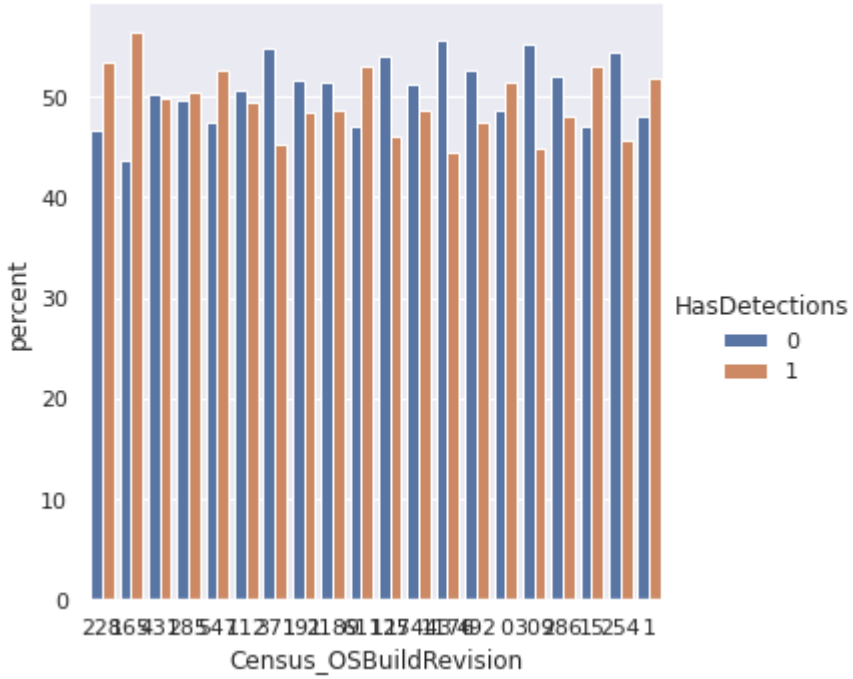
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f847edd1390>



In [ ]:
```
(df
```

```
.groupby('Census_OSBuildRevision')['HasDetections']
.value_counts(normalize=True)
.mul(100)
.rename('percent')
.reset_index()
.pipe((sns.catplot,'data'), x='Census_OSBuildRevision',y='percent',hue='HasDetections',
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7f847f2b5990>



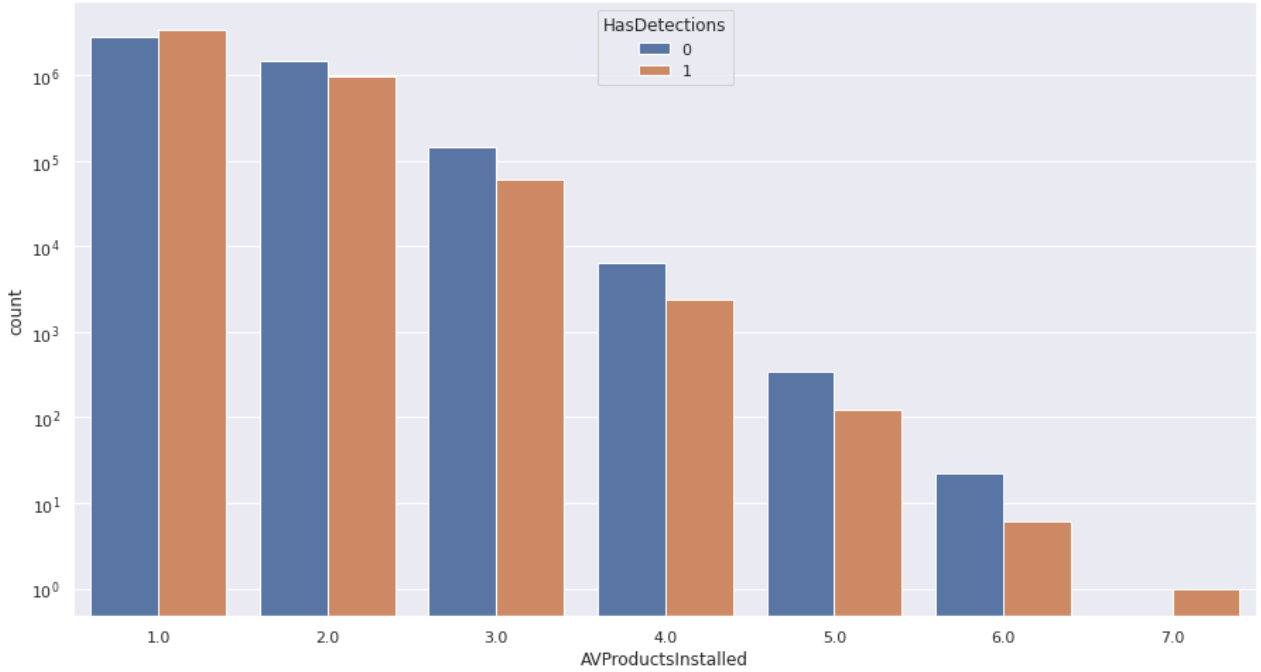
Section 4: Effect of Number of AV Products Installed (Q4)

In [ ]:
```
print("Machines which are not protected and not infected with malware =", np.count_nonz
print("Machines which are not protected and infected with malware =", np.count_nonzero(
print("Machines which are protected and not infected with malware =", np.count_nonzero(
print("Machines which are protected and infected with malware =", np.count_nonzero((df[
```

```
Machines which are not protected and not infected with malware = 298904
Machines which are not protected and infected with malware = 184253
Machines which are protected and not infected with malware = 4141184
Machines which are protected and infected with malware = 4261098
```

Thus, based on above statistics, it seems that there is no effect of having a antivirus in chances of the machine getting infected with malware

In [ ]:
```
sns.countplot(x='AVProductsInstalled', hue='HasDetections',data=df[df['AVProductsInstal
```



From above graph, it seems that if the machine has 1 anti virus, then chances if it getting infected is more but as number of antivirus increases, malware detections in machine decreases (except for when AVProductsInstalled = 7, which is an outlier I believe since there is only 1 machine in the entire dataset. Thus number of AV products matter.

Section 5: Interesting findings (Q5)

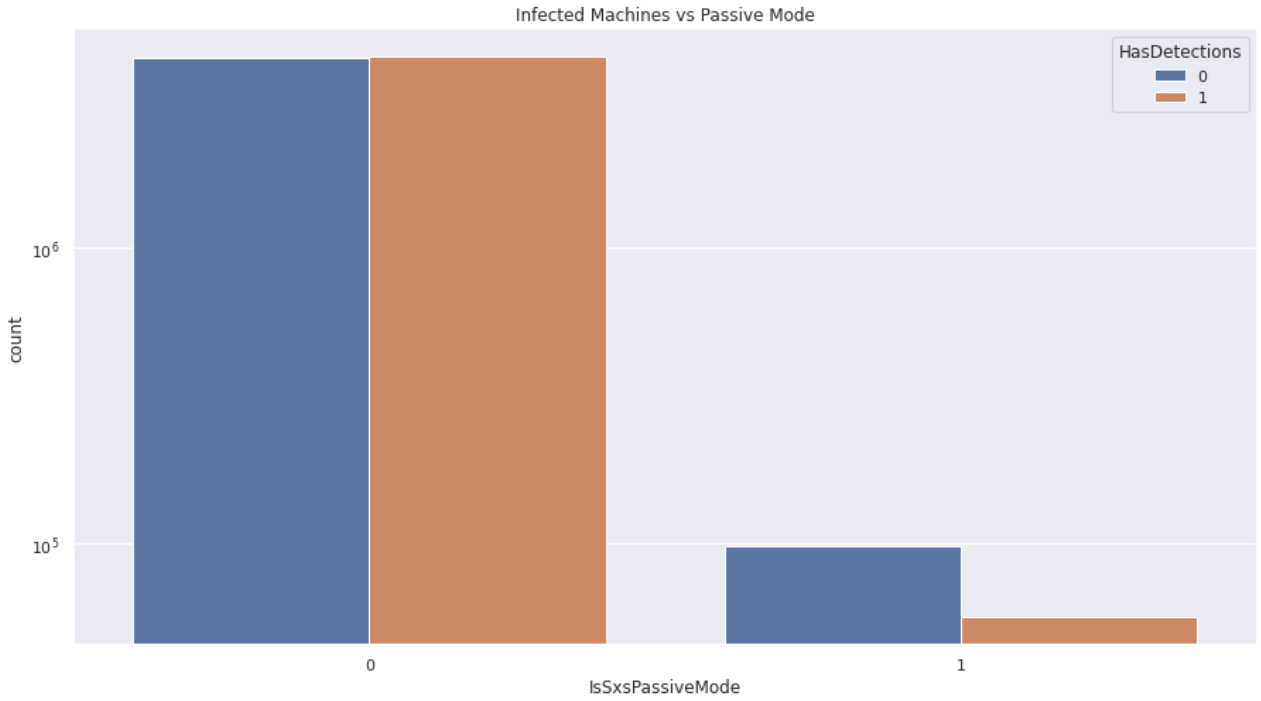
In [ ]:
```
del df
gc.collect()
```

Out[ ]: 9675

In [ ]:
```
df = pd.read_csv('train.csv', usecols=["IsSxsPassiveMode", "PuaMode", "SMode", "Platfor
```

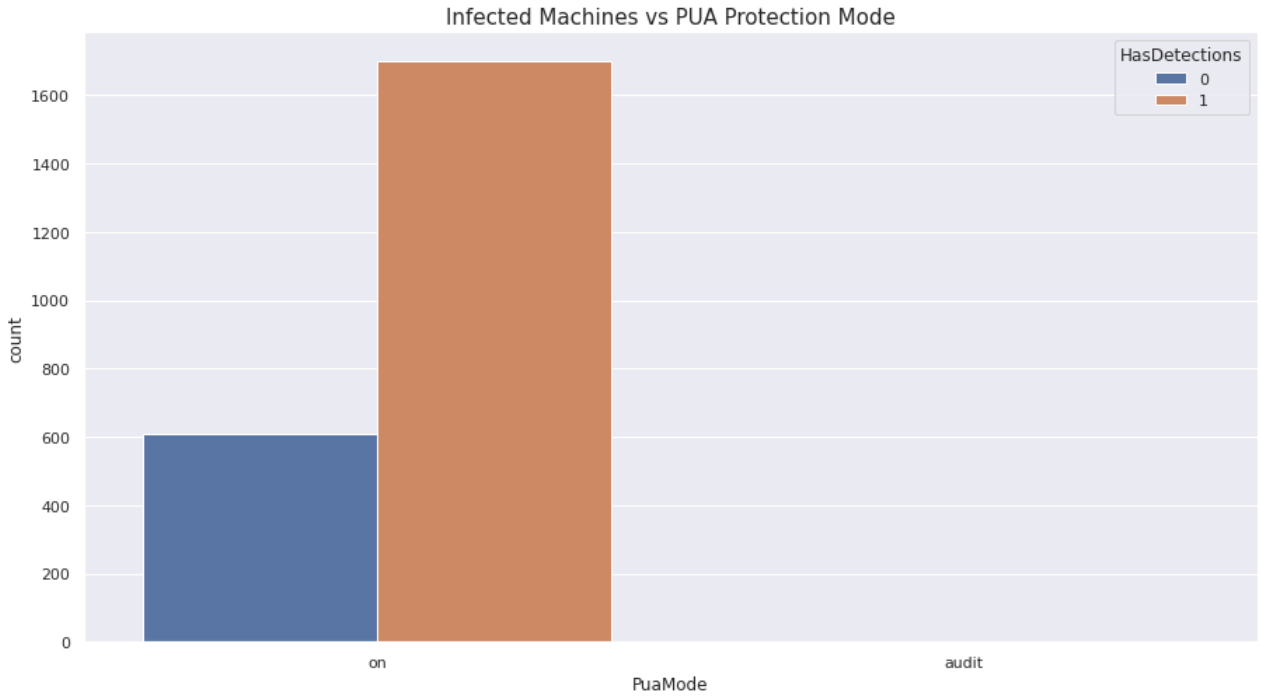
1. SxS Passive Mode means Microsoft Defender Antivirus is running alongside another antivirus/antimalware product. Thus, when a machine is having one third party antivirus installed alongside Windows Defender, it has very less chances of getting infected with malware.

In [ ]:
```
sns.countplot(x='IsSxsPassiveMode', hue='HasDetections', data=df)
plt.yscale('log')
plt.title('Infected Machines vs Passive Mode')
plt.show()
```



1. Should we enable PUA mode as recommended by windows? From their website "The Potentially Unwanted Applications (PUA) protection feature in Microsoft Defender Antivirus can identify and block PUAs from downloading and installing on endpoints in your network." The plots seem to show something else that if PUA is on, then relatively more machines are infected with malware.

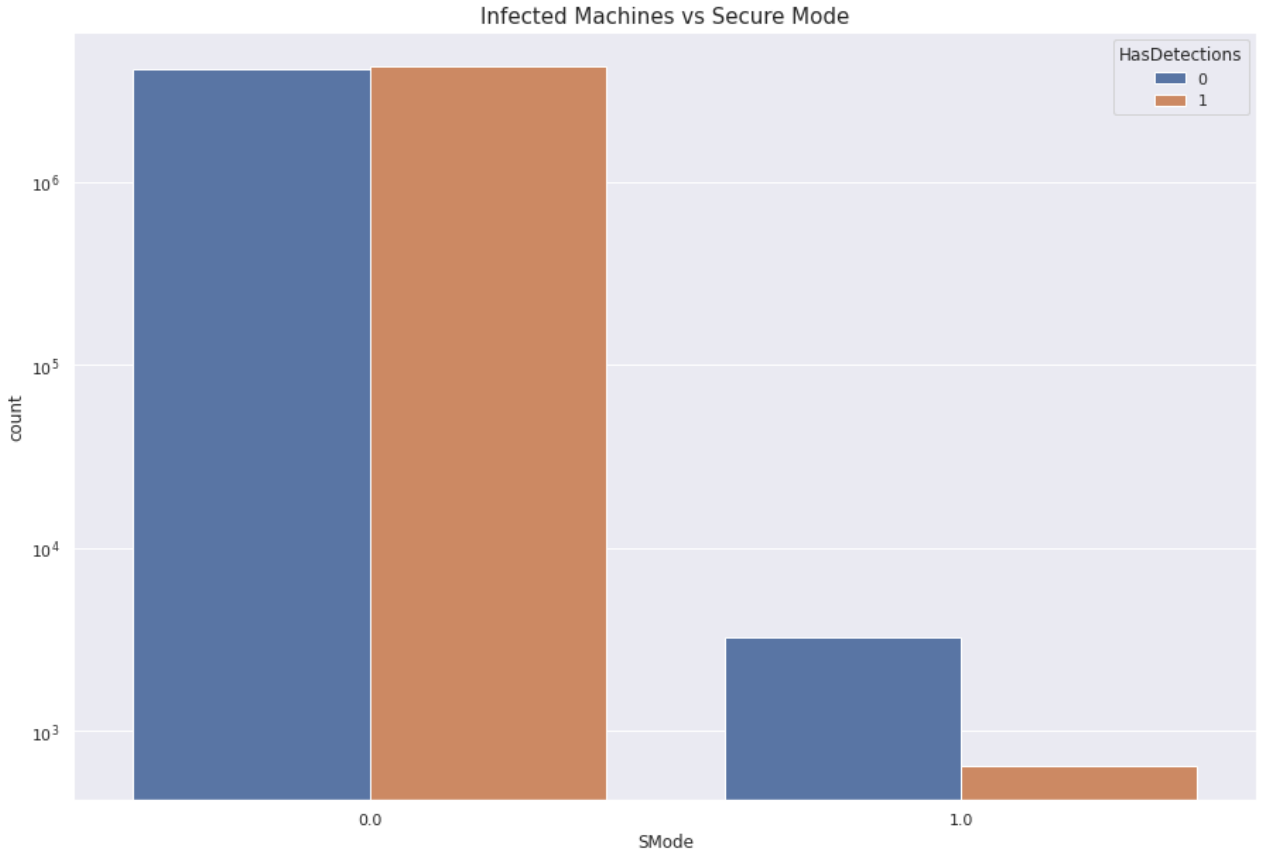
In [ ]:
```
sns.countplot(x='PuaMode', hue='HasDetections', data=df)
plt.title('Infected Machines vs PUA Protection Mode', size=15)
plt.show()
```



1. It is very difficult to be infected with malware with S mode enabled. Thus, Microsoft's claim that

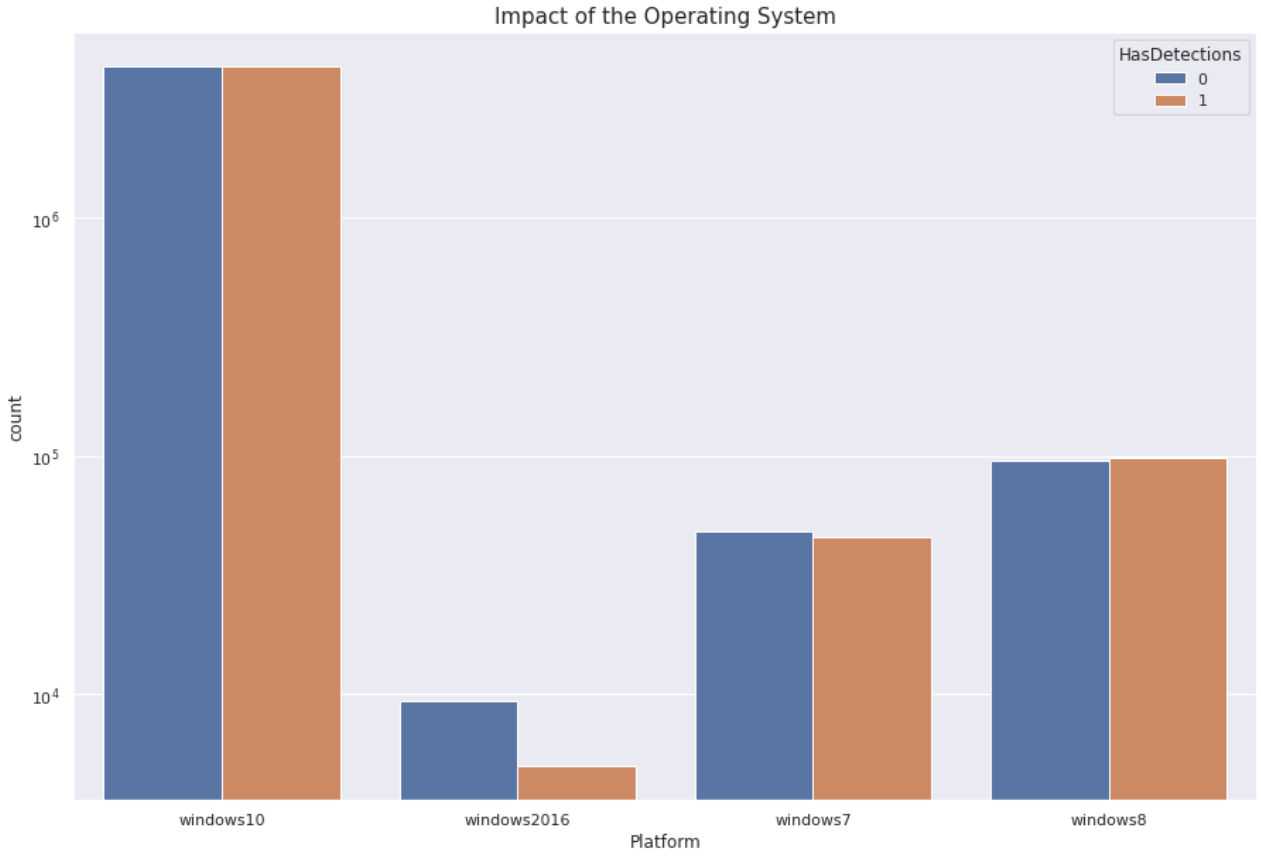
"Windows 10 in S mode is a version of Windows 10 that's streamlined for security and performance" is correct.

In [ ]:
```
plt.figure(figsize=(15,10))
sns.countplot(x='SMode', hue='HasDetections', data=df)
plt.title('Infected Machines vs Secure Mode', size=15)
plt.yscale('log')
plt.show()
```



1. It seems that windows 2016 has very less computers infected with malware as compared to other platforms.

In [ ]:
```
plt.figure(figsize=(15,10))
sns.countplot(x='Platform', hue='HasDetections', data=df)
plt.title('Impact of the Operating System', size=15)
plt.yscale('log')
plt.show()
```



Section 6: Baseline modelling (Q6)

In [ ]:
```
del df
gc.collect()
```

Out[ ]: 7405

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import joblib
```

Features I feel are useful, and thus used in baseline model are mentioned in variable baseline_cols

In [ ]:
```
baseline_cols
```

Out[ ]:
```
['AVProductsInstalled',
 'AVProductStatesIdentifier',
 'Census_IsAlwaysOnAlwaysConnectedCapable',
 'IsProtected',
 'Census_TotalPhysicalRAM',
 'Census_ProcessorCoreCount',
 'Census_OSBuildNumber']
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
df = pd.read_csv('train.csv', usecols=baseline_cols + ['HasDetections'], dtype=dtypes)
df.dropna(inplace=True)
```

```
X = df[baseline_cols]
y = df[['HasDetections']]
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
print("Accuracy of baseline model is ", clf.score(X_test, y_test))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change the shap
e of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy of baseline model is  0.5418740279716378
```

In [ ]:
```
print("Error rate is 1 - accuracy = ", 1 - clf.score(X_test, y_test))
print("AUC score is ", roc_auc_score(y_test, clf.decision_function(X_test)))
```

```
Error rate is 1 - accuracy =  0.45812597202836225
AUC score is  0.5576255444943179
```

In [ ]:
```
joblib.dump(clf, 'logistic_base.pkl')
```

Out[ ]:
```
['logistic_base.pkl']
```

Section 7: Feature Cleaning and Additional models (Q7a & 7b)

In [ ]:
```
advance_models = ['AVProductsInstalled',
 'AVProductStatesIdentifier',
 'Census_IsAlwaysOnAlwaysConnectedCapable',
 'IsProtected',
 'Census_TotalPhysicalRAM',
 'Census_ProcessorCoreCount',
 'Census_OSBuildNumber',
 'Census_PrimaryDiskTypeName',
 'Census_InternalPrimaryDiagonalDisplaySizeInInches',
 'Census_InternalPrimaryDisplayResolutionVertical']
```

In [ ]:
```
del df
gc.collect()
```

Out[ ]:
```
150
```

In [ ]:
```
df = pd.read_csv('train.csv', usecols=advance_models + ['HasDetections'], dtype=dtypes)
med = df.median()
mde = df.mode()
df.fillna(med, inplace=True) #Replace missing numerical values with median, this does n
df.fillna(mde, inplace=True) #Replace missing categorical values with mode
```

In [ ]:
```
#Census_OSBuildNumber should be converted to one-hot encoding for further processing
#We will only consider those which appear more than 10000 times, else number of columns
```

```
less_common_build_number = df.Census_OSBuildNumber.value_counts()
list_more_common_build_number = less_common_build_number[less_common_build_number > 100
for i, v in df['Census_OSBuildNumber'].items():
  if v not in list_more_common_build_number:
    df.at[i, 'Census_OSBuildNumber'] = 0

os_build_number = pd.get_dummies(df.Census_OSBuildNumber, prefix='OSBuildNumber')
df = pd.concat([df, os_build_number], axis=1)
del df['OSBuildNumber_0']
del df['Census_OSBuildNumber']

gc.collect()
```

Out[ ]: 0

In [ ]:
```
#Similarly, AVProductStatesIdentifier should be converted to one-hot encoding for furth
#We will only consider those which appear more than 80000 times, else number of columns

less_common_build_number = df.AVProductStatesIdentifier.value_counts()
list_more_common_build_number = less_common_build_number[less_common_build_number > 800
for i, v in df['AVProductStatesIdentifier'].items():
  if v not in list_more_common_build_number:
    df.at[i, 'AVProductStatesIdentifier'] = 0

os_build_number = pd.get_dummies(df.AVProductStatesIdentifier, prefix='AVProductID')
df = pd.concat([df, os_build_number], axis=1)
del df['AVProductID_0.0']
del df['AVProductStatesIdentifier']

gc.collect()
```

Out[ ]: 0

In [ ]:
```
storage_type = pd.get_dummies(df.Census_PrimaryDiskTypeName, prefix='Storage')
df = pd.concat([df, storage_type], axis=1)
del df['Census_PrimaryDiskTypeName']
gc.collect()
```

Out[ ]: 50

Census_InternalPrimaryDiagonalDisplaySizeInInches and Census_InternalPrimaryDisplayResolutionVertical can be an important feature too because sometimes https://www.makeuseof.com/tag/malware-uses-screen-resolution-avoid-detection/

In [ ]:
```
print("Missing values in Census_InternalPrimaryDisplayResolutionVertical", df.Census_In
print("Missing values in Census_InternalPrimaryDiagonalDisplaySizeInInches", df.Census_
```

```
Missing values in Census_InternalPrimaryDisplayResolutionVertical 0
Missing values in Census_InternalPrimaryDiagonalDisplaySizeInInches 0
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler #Not used as none of the distributions
from sklearn.preprocessing import MinMaxScaler #MinMaxScaler may be used when the upper
from sklearn.preprocessing import RobustScaler # For Processor and RAM as df.col_name.h
```

In [ ]:
```
print("Replacing erratic value of having -1 resolution by median ",df.Census_InternalPr
for i, v in df['Census_InternalPrimaryDisplayResolutionVertical'].items():
  if v == -1.0:
    df.at[i, 'Census_InternalPrimaryDisplayResolutionVertical'] = 768.0
```

```
Replacing erratic value of having -1 resolution by median  768.0
```

In [ ]:
```
scalar1 = MinMaxScaler()
scalar2 = MinMaxScaler()
df.Census_InternalPrimaryDisplayResolutionVertical = scalar1.fit_transform(df.Census_In
df.Census_InternalPrimaryDiagonalDisplaySizeInInches = scalar2.fit_transform(df.Census_
```

In [ ]:
```
scalar3 = RobustScaler()
scalar4 = RobustScaler()
df.Census_ProcessorCoreCount = scalar3.fit_transform(df.Census_ProcessorCoreCount.value
df.Census_TotalPhysicalRAM = scalar4.fit_transform(df.Census_TotalPhysicalRAM.values.re
```

In [ ]:
```
print("Missing values in RAM", df.Census_TotalPhysicalRAM.isna().sum())
print("Missing values in ProcessorCount", df.Census_ProcessorCoreCount.isna().sum())
```

```
Missing values in RAM 0
Missing values in ProcessorCount 0
```

In [ ]:
```
X = df.drop('HasDetections', axis=1)
y = df[['HasDetections']]
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
print("Accuracy of advance logistic regression is ", clf.score(X_test, y_test))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change the shap
e of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
Accuracy of advance logistic regression is  0.5871903612459136
```

In [ ]:
```
print("Error rate is 1 - accuracy = ", 1 - clf.score(X_test, y_test))
print("AUC score is ", roc_auc_score(y_test, clf.decision_function(X_test)))
```

```
Error rate is 1 - accuracy =  0.4128096387540864
AUC score is  0.6128417848111801
```

In [ ]:
```
joblib.dump(clf, 'logistic_advance.pkl')
```

Out[ ]:
```
['logistic_advance.pkl']
```

Advance Model

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0).fit(X_train, y_train)
print("Accuracy of advance random forest classifier is ", rfc.score(X_test, y_test))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

Accuracy of advance random forest classifier is  0.5936113774780768
```

In [ ]:
```
print("Error rate is 1 - accuracy = ", 1 - rfc.score(X_test, y_test))
print("AUC score is ", roc_auc_score(y_test, rfc.predict_proba(X_test)[:,1]))
```

```
Error rate is 1 - accuracy =  0.4063886225219232
AUC score is  0.6195889399745863
```

In [ ]:
```
joblib.dump(rfc, 'random_forest_classifier.pkl')
```

Out[ ]:
```
['random_forest_classifier.pkl']
```

Random Forest Classifier performs better than advance logistic regression because it uses ensemble learning (bagging) to reduce variance in the model, which in turn performs better than baseling model because we have done data pre-processing, handling of missing values, etc.

Error rate in a dataframe table

In [95]:
```
dataerr = [['baseline_logistic_regression', 0.45812597202836225], ['advance_logistic_re
pd.DataFrame(dataerr, columns = ['Model', 'Error Rate'])
```

Out[95]:

	Model	Error Rate
0	baseline_logistic_regression	0.458126
1	advance_logistic_regression	0.412810
2	random_forest_classifier	0.406389

Section 8: Screenshots (Q8)

In [ ]:
```
del df
gc.collect()
```

Out[ ]:
```
77
```

In [ ]:
```
!unzip test.csv.zip
```

```
Archive:  test.csv.zip
  inflating: test.csv
```

In [ ]:
```
df_test = pd.read_csv('test.csv',usecols=advance_models,dtype=dtypes)
```

In [ ]:

```
df_test.fillna(med, inplace=True) #Replace missing numerical values with median of TRAI
df_test.fillna(mde, inplace=True) #Replace missing categorical values with mode of TRAI


less_common_build_number = df_test.Census_OSBuildNumber.value_counts()
list_more_common_build_number = less_common_build_number[less_common_build_number > 300
for i, v in df_test['Census_OSBuildNumber'].items():
  if v not in list_more_common_build_number:
    df_test.at[i, 'Census_OSBuildNumber'] = 0

os_build_number = pd.get_dummies(df_test.Census_OSBuildNumber, prefix='OSBuildNumber')
df_test = pd.concat([df_test, os_build_number], axis=1)
del df_test['OSBuildNumber_0']
del df_test['Census_OSBuildNumber']

gc.collect()


#Similarly, AVProductStatesIdentifier should be converted to one-hot encoding for furth
#We will only consider those which appear more than 80000 times, else number of columns

less_common_build_number = df_test.AVProductStatesIdentifier.value_counts()
list_more_common_build_number = less_common_build_number[less_common_build_number > 800
for i, v in df_test['AVProductStatesIdentifier'].items():
  if v not in list_more_common_build_number:
    df_test.at[i, 'AVProductStatesIdentifier'] = 0

os_build_number = pd.get_dummies(df_test.AVProductStatesIdentifier, prefix='AVProductID
df_test = pd.concat([df_test, os_build_number], axis=1)
del df_test['AVProductID_0.0']
del df_test['AVProductStatesIdentifier']

gc.collect()


storage_type = pd.get_dummies(df_test.Census_PrimaryDiskTypeName, prefix='Storage')
df_test = pd.concat([df_test, storage_type], axis=1)
del df_test['Census_PrimaryDiskTypeName']
gc.collect()



print("Replacing erratic value of having -1 resolution by median from TRAIN data")
for i, v in df_test['Census_InternalPrimaryDisplayResolutionVertical'].items():
  if v == -1.0:
    df_test.at[i, 'Census_InternalPrimaryDisplayResolutionVertical'] = 768.0


df_test.Census_InternalPrimaryDisplayResolutionVertical = scalar1.transform(df_test.Cen
df_test.Census_InternalPrimaryDiagonalDisplaySizeInInches = scalar2.transform(df_test.C

df_test.Census_ProcessorCoreCount = scalar3.transform(df_test.Census_ProcessorCoreCount
df_test.Census_TotalPhysicalRAM = scalar4.transform(df_test.Census_TotalPhysicalRAM.val
```

```
Replacing erratic value of having -1 resolution by median from TRAIN data
```

Note: On analysing the columns of df_test, and comparing with columns of df, we can see OSBuildNumber_17763 but that is not there in our training data. So, the best way is to drop the

column

In [ ]:
```
df_test.drop('OSBuildNumber_17763',axis=1,inplace=True)
```

Also, the test data does not have some of our one-hot encoded features. Thus, since the model is already trained, I am creating a new feature with value = 0, so that it does not affect training

In [ ]:
```
for av in ['AVProductID_23657.0', 'AVProductID_41571.0', 'AVProductID_46413.0']:
  df_test[av] = 0
```

In [ ]:
```
gc.collect()
```

In [ ]:
```
print(df_test.columns)
X_train.columns
```

```
Index(['AVProductsInstalled', 'IsProtected', 'Census_ProcessorCoreCount',
       'Census_TotalPhysicalRAM',
       'Census_InternalPrimaryDiagonalDisplaySizeInInches',
       'Census_InternalPrimaryDisplayResolutionVertical',
       'Census_IsAlwaysOnAlwaysConnectedCapable', 'OSBuildNumber_10240',
       'OSBuildNumber_10586', 'OSBuildNumber_14393', 'OSBuildNumber_15063',
       'OSBuildNumber_16299', 'OSBuildNumber_17134', 'AVProductID_7945.0',
       'AVProductID_47238.0', 'AVProductID_49480.0', 'AVProductID_53447.0',
       'AVProductID_62773.0', 'Storage_HDD', 'Storage_SSD', 'Storage_UNKNOWN',
       'Storage_Unspecified'],
      dtype='object')
```

Out[ ]:
```
Index(['AVProductsInstalled', 'IsProtected', 'Census_ProcessorCoreCount',
       'Census_TotalPhysicalRAM',
       'Census_InternalPrimaryDiagonalDisplaySizeInInches',
       'Census_InternalPrimaryDisplayResolutionVertical',
       'Census_IsAlwaysOnAlwaysConnectedCapable', 'OSBuildNumber_10240',
       'OSBuildNumber_10586', 'OSBuildNumber_14393', 'OSBuildNumber_15063',
       'OSBuildNumber_16299', 'OSBuildNumber_17134', 'AVProductID_7945.0',
       'AVProductID_23657.0', 'AVProductID_41571.0', 'AVProductID_46413.0',
       'AVProductID_47238.0', 'AVProductID_49480.0', 'AVProductID_53447.0',
       'AVProductID_62773.0', 'Storage_HDD', 'Storage_SSD', 'Storage_UNKNOWN',
       'Storage_Unspecified'],
      dtype='object')
```

In [ ]:
```
!unzip sample_submission.csv.zip
```

```
Archive:  sample_submission.csv.zip
  inflating: sample_submission.csv
```

In [68]:
```
submission = pd.read_csv('sample_submission.csv')
submission['HasDetections'] = rfc.predict_proba(df_test)[:,1]
submission.to_csv('randomforest_submission.csv', index=False)
```

In [70]:
```
gc.collect()
submission['HasDetections'] = clf.predict_proba(df_test)[:,1]
submission.to_csv('logistic_advance_submission.csv', index=False)
```

In [72]:
```
clf = joblib.load('logistic_base.pkl')
```

In [74]:
```
del df_test
gc.collect()
```

Out[74]: 482

In [78]:
```
del X
del y
del X_train
del y_train
gc.collect()
```

Out[78]: 397

In [82]:
```
del df_test
gc.collect()
```

Out[82]: 291

In [89]:
```
#df_test = pd.read_csv('test.csv', usecols=baseline_cols, dtype=dtypes)
df_test.fillna(df_test.median(), inplace=True) #Replace missing values with median
submission['HasDetections'] = clf.predict_proba(df_test)[:,1]
submission.to_csv('logistic_base_submission.csv', index=False)
```

logistic_base_submission.csv 0.54580 Private 0.53650 Public

logistic_advance_submission.csv 0.53629 Private 0.59341 Public

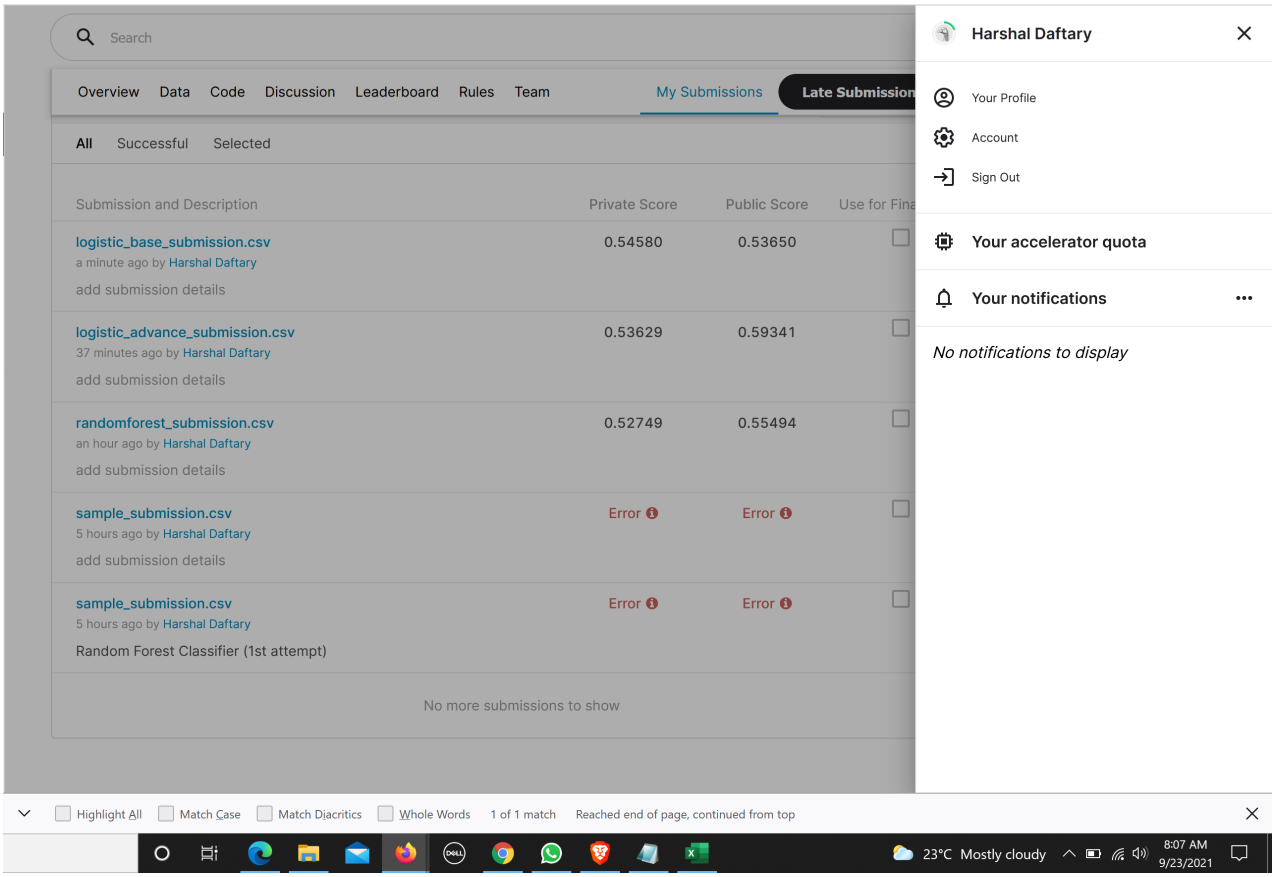
randomforest_submission.csv 0.52749 Private 0.55494 Public

Kaggle profile link: https://www.kaggle.com/harshaldaftar

Screenshot(s): !picture

In [97]:
```
from IPython.display import Image
Image("kaggle.PNG", width=1200)
```

Out[97]:


Search
Overview Data Code Discussion Leaderboard Rules Team My Submissions Late Submission
All Successful Selected
Submission and Description Private Score Public Score Use for Fina
logistic_base_submission.csv
a minute ago by Harshal Daftary
add submission details
0.54580 0.53650
logistic_advance_submission.csv
37 minutes ago by Harshal Daftary
add submission details
0.53629 0.59341
randomforest_submission.csv
an hour ago by Harshal Daftary
add submission details
0.52749 0.55494
sample_submission.csv
5 hours ago by Harshal Daftary
add submission details
Error Error
sample_submission.csv
5 hours ago by Harshal Daftary
Random Forest Classifier (1st attempt)
Error Error
No more submissions to show
Harshal Daftary
Your Profile
Account
Sign Out
Your accelerator quota
Your notifications
No notifications to display
Highlight All Match Case Match Diacritics Whole Words 1 of 1 match Reached end of page, continued from top
23°C Mostly cloudy
8:07 AM
9/23/2021