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.

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call d
```

```
# First mount your drive before running these cells.
# Create a folder for the this HW and change to that dir
%cd /content/gdrive/MyDrive/CSE519Fall2021/HW2/
```

```
/content/gdrive/MyDrive/CSE519Fall2021/HW2
```

```
!pip install -q kaggle
```

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

```
Choose Files kaggle.json
• kaggle.json(application/json) - 75 bytes, last modified: 9/16/2021 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"vamsikrishnapalleni","key":"a37f315ea1681787
```

```
!kaggle competitions download -c microsoft-malware-prediction
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix t
Warning: Looks like you're using an outdated API Version, please consider updati
Downloading test.csv.zip to /root/.kaggle
 99% 665M/672M [00:13<00:00, 52.7MB/s]
100% 672M/672M [00:13<00:00, 51.3MB/s]
Downloading sample_submission.csv.zip to /root/.kaggle
 98% 131M/134M [00:01<00:00, 101MB/s]
100% 134M/134M [00:01<00:00, 95.2MB/s]
```

```
Downloading train.csv.zip to /root/.kaggle
 99% 764M/768M [00:08<00:00, 128MB/s]
100% 768M/768M [00:08<00:00, 91.2MB/s]
```

Unzipping Train Data

```
from zipfile import ZipFile
train_file="train.csv.zip"
with  ZipFile(train_file,'r') as zip_file :
  zip_file.extractall();
  print('Reached unzip stage!')
```

```
Reached unzip stage!
```

Unzipping Test Data

```
from zipfile import ZipFile
test_file="test.csv.zip"
with  ZipFile(test_file,'r') as zip_file_test :
  zip_file_test.extractall();
  print('Reached unzip stage!')
```

```
Reached unzip stage!
```

Unzipping Sample Data

```
from zipfile import ZipFile
sample_file="sample_submission.csv.zip"
with  ZipFile(sample_file,'r') as zip_file_sample :
  zip_file_sample.extractall();
  print('Reached unzip stage!')
```

```
Reached unzip stage!
```

```
 # This is formatted as code
```

Section 1: Library and Data Imports (Q1)

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

```
use_cols·=·["MachineIdentifier",·"SmartScreen",·"AVProductsInstalled",·"AppVersion",·
···········"EngineVersion",·"AVProductStatesIdentifier",·"Census_OSVersion",·"Census_
···········"RtpStateBitfield",·"Census_ProcessorModelIdentifier",·"Census_PrimaryDisk
···········"Census_InternalPrimaryDiagonalDisplaySizeInInches",·"Wdft_RegionIdentifi
```

```
    ···········"AvSigVersion",·"IeVerIdentifier",·"IsProtected",·"Census_InternalPrimaryD
    ···········"Census_OSWUAutoUpdateOptionsName",·"Census_OSEdition",·"Census_GenuineSt
    ···········"Census_OEMNameIdentifier",·"Census_MDC2FormFactor",·"Census_FirmwareManuf
    ···········"Census_OSBuildNumber",·"Census_IsPenCapable",·"Census_IsTouchEnabled",·"
    ···········"Census_SystemVolumeTotalCapacity",·"Census_PrimaryDiskTotalCapacity","Ha
    ···········]
    dtypes = {
            'MachineIdentifier':                                    'category',
            'ProductName':                                          'category',
            'EngineVersion':                                        'category',
            'AppVersion':                                           'category',
            'AvSigVersion':                                         'category',
            'IsBeta':                                               'int8',
            'RtpStateBitfield':                                     'float16',
            'IsSxsPassiveMode':                                     'int8',
            'DefaultBrowsersIdentifier':                            'float16',
            'AVProductStatesIdentifier':                            'float32',
            'AVProductsInstalled':                                  'float16',
            'AVProductsEnabled':                                    'float16',
            'HasTpm':                                               'int8',
            'CountryIdentifier':                                    'int16',
            'CityIdentifier':                                       'float32',
            'OrganizationIdentifier':                               'float16',
            'GeoNameIdentifier':                                    'float16',
            'LocaleEnglishNameIdentifier':                          'int8',
            'Platform':                                             'category',
            'Processor':                                            'category',
            'OsVer':                                                'category',
            'OsBuild':                                              'int16',
            'OsSuite':                                              'int16',
            'OsPlatformSubRelease':                                 'category',
            'OsBuildLab':                                           'category',
            'SkuEdition':                                           'category',
            'IsProtected':                                          'float16',
            'AutoSampleOptIn':                                      'int8',
            'PuaMode':                                              'category',
            'SMode':                                                'float16',
            'IeVerIdentifier':                                      'float16',
            'SmartScreen':                                          'category',
            'Firewall':                                             'float16',
            'UacLuaenable':                                         'float32',
            'Census_MDC2FormFactor':                                'category',
            'Census_DeviceFamily':                                  'category',
            'Census_OEMNameIdentifier':                             'float16',
            'Census_OEMModelIdentifier':                            'float32',
            'Census_ProcessorCoreCount':                            'float16',
            'Census_ProcessorManufacturerIdentifier':               'float16',
            'Census_ProcessorModelIdentifier':                      'float16',
            'Census_ProcessorClass':                                'category',
            'Census_PrimaryDiskTotalCapacity':                      'float32',
            'Census_PrimaryDiskTypeName':                           'category',
            'Census_SystemVolumeTotalCapacity':                     'float32',
```

```
 'Census_HasOpticalDiskDrive':                              'int8',
 'Census_TotalPhysicalRAM':                                 'float32',
 'Census_ChassisTypeName':                                  'category',
 'Census_InternalPrimaryDiagonalDisplaySizeInInches':       'float16',
 'Census_InternalPrimaryDisplayResolutionHorizontal':       'float16',
 'Census_InternalPrimaryDisplayResolutionVertical':         'float16',
 'Census_PowerPlatformRoleName':                            'category',
 'Census_InternalBatteryType':                              'category',
 'Census_InternalBatteryNumberOfCharges':                   'float32',
 'Census_OSVersion':                                        'category',
 'Census_OSArchitecture':                                   'category',
 'Census_OSBranch':                                         'category',
 'Census_OSBuildNumber':                                    'int16',
 'Census_OSBuildRevision':                                  'int32',
 'Census_OSEdition':                                        'category',
 'Census_OSSkuName':                                        'category',
 'Census_OSInstallTypeName':                                'category',
 'Census_OSInstallLanguageIdentifier':                      'float16',
 'Census_OSUILocaleIdentifier':                             'int16',
 'Census_OSWUAutoUpdateOptionsName':                        'category',
 'Census_IsPortableOperatingSystem':                        'int8',
 'Census_GenuineStateName':                                 'category',
 'Census_ActivationChannel':                                'category',
 'Census_IsFlightingInternal':                              'float16',
 'Census_IsFlightsDisabled':                                'float16',
 'Census_FlightRing':                                       'category',
 'Census_ThresholdOptIn':                                   'float16',
 'Census_FirmwareManufacturerIdentifier':                   'float16',
 'Census_FirmwareVersionIdentifier':                        'float32',
 'Census_IsSecureBootEnabled':                              'int8',
 'Census_IsWIMBootEnabled':                                 'float16',
 'Census_IsVirtualDevice':                                  'float16',
 'Census_IsTouchEnabled':                                   'int8',
 'Census_IsPenCapable':                                     'int8',
 'Census_IsAlwaysOnAlwaysConnectedCapable':                 'float16',
 'Wdft_IsGamer':                                            'float16',
 'Wdft_RegionIdentifier':                                   'float16'
}
```

Loaded the CSV into a dataframe with the features as listed in the use_cols variable.

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

df=pd.read_csv('train.csv',usecols=use_cols,dtype=dtypes)
#Loaded the CSV into a dataframe with the features as listed in the use_cols variable
```

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

2a

Measure of computer power as a function of RAM, processor core count,OSBuildNumber,ProcessorModelIdentifier.

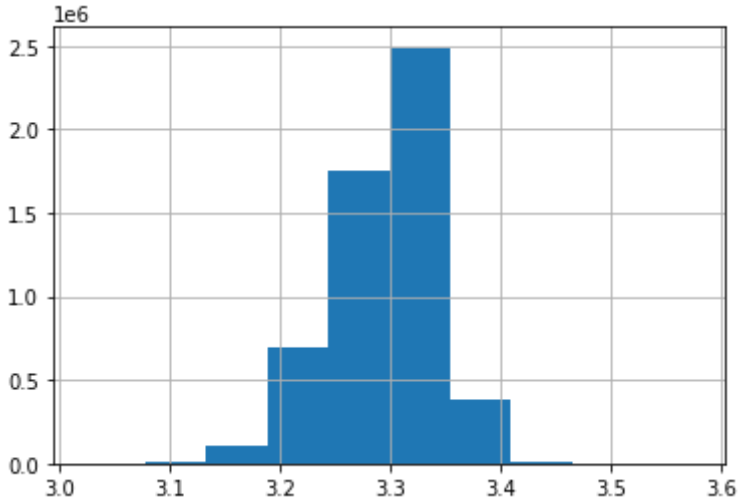
Plotted histogram for the same

```
df.dropna(axis='index',how='any',inplace=True)
#removing na values and placing it into the same dataframe


import numpy as np
#scalling the column's using log operation provided by numpy library and copying the
df['Log_Census_TotalPhysicalRAM']=np.log( df['Census_TotalPhysicalRAM'])
df['Log_Census_ProcessorCoreCount']=np.log(df['Census_ProcessorCoreCount'])
df['Log_Census_OSBuildNumber']=np.log(df['Census_OSBuildNumber'])
df['Log_Census_ProcessorModelIdentifier']=np.log(df['Census_ProcessorModelIdentifier'
#assigning all the requried columns names into one single variable
powerall =['Log_Census_TotalPhysicalRAM','Log_Census_ProcessorCoreCount','Log_Census_
#obtaining the system power by adding the above logged columns and again scalling int
df['power_map']=np.log((df['Log_Census_TotalPhysicalRAM']+df['Log_Census_ProcessorCor

df['power_map'].hist() # Histogram visualizing the system power which was obtained fr
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83b4b5efd0>
```

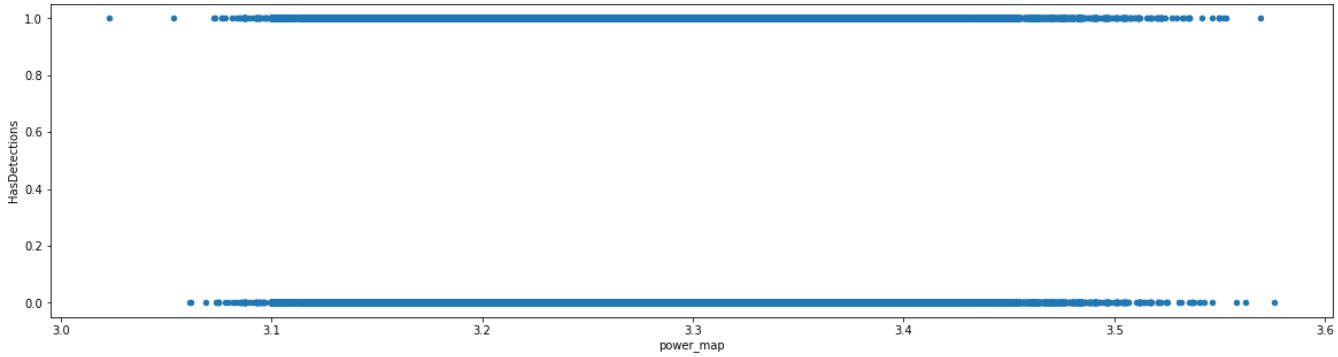


2b

Q:Powerful computers more or less likely to have malware than underpowered machines? Ans: In the below scatter plot it is evident that there is no correlation between powerful computers more or less likely to have malware

```
df[["power_map", "HasDetections"]].plot(x="power_map", y="HasDetections", kind="scatt
#Power vs malware
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83c2166fd0>
```



Section 3: OS version vs Malware detected (Q3)

Number-(Malware detections against Census_OSBuildNumber)

```
#Number of Malware vs Census_OSBuildNumber Plot
total_buildnumber1=df[['Census_OSBuildNumber','HasDetections']].groupby('Census_OSBui
total_buildnumber1.plot()
```

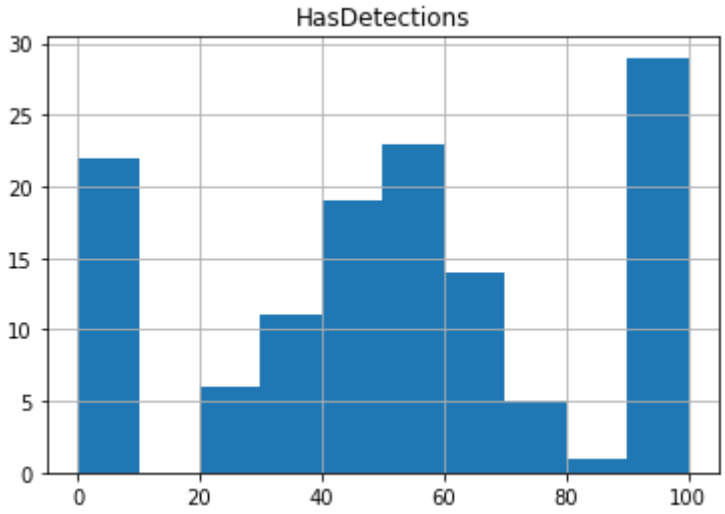
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83bfa1bb10>
```

From the above graph in betwwen (16000-18000)OSBuildNumber has heighest number of malware attacks when compared to other OSBuildNumber.

Percentage-(Malware detections against Census_OSBuildNumber)

```
#Percent of Malware vs Census_OSBuildNumber Plot
total_buildnumber2=df[['Census_OSBuildNumber','HasDetections']].groupby('Census_OSBui
total_buildnumber=total_buildnumber1/total_buildnumber2
total_buildnumber=total_buildnumber*100
total_buildnumber.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f839545f990>]],
      dtype=object)
```

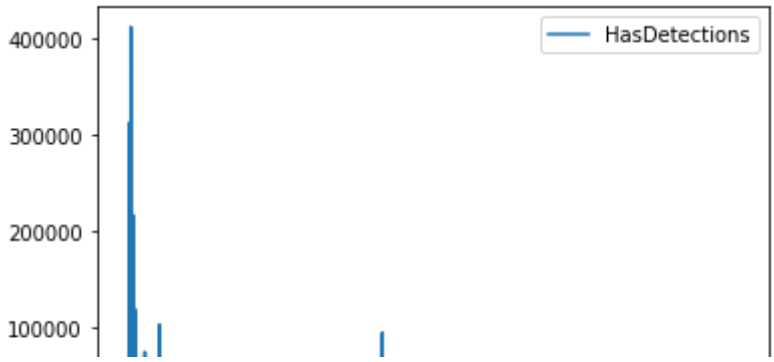


From the above histogram (0-10)OSBuildNumber has highest percentage of malware detection and from (30-60)OSBuildNumber has consistency percentage level and lowest range was maintained between (70-90)OSBuildNumber.

Number-(Malware detections against Census_OSBuildRevision)

```
#Number of Malware vs Census_OSBuildRevision Plot
total_buildnumber3=df[['Census_OSBuildRevision','HasDetections']].groupby('Census_OSB
total_buildnumber3.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83bfa1b8d0>
```



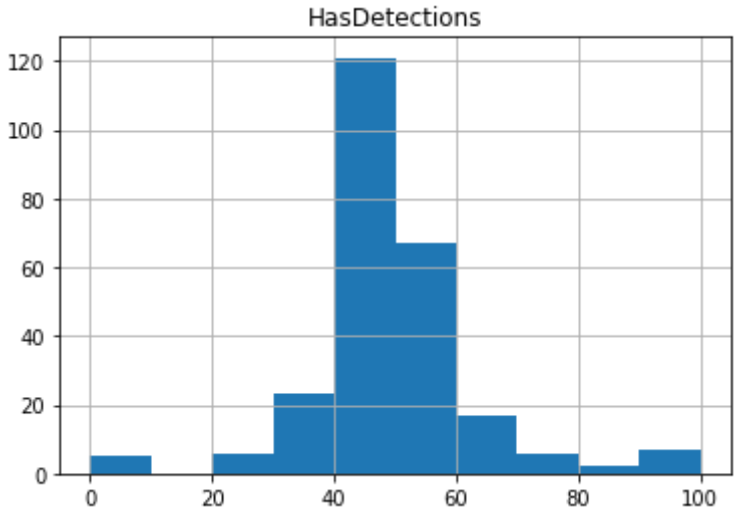
From the above graph in betwwen (0-10000)OSBuildRevision has heighest number of malware attacks when compared to other OSBuildRevision.

Census_OSBuildRevision

Percentage-(Malware detections against Census_OSBuildRevision)

```
#Percentage of Malware vs Census_OSBuildRevision Plot
total_buildnumber4=df[['Census_OSBuildRevision','HasDetections']].groupby('Census_OSB
total_buildnumber=total_buildnumber3/total_buildnumber4
total_buildnumber=total_buildnumber*100
total_buildnumber.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f83b33fcd90>]],
      dtype=object)
```



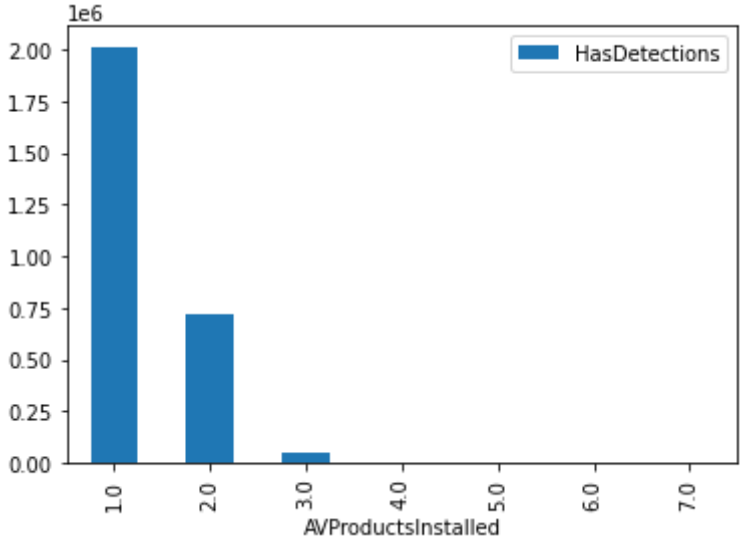
From the above histogram in between(40-60)percentage has highest number of malware attacks

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

```
av_mal=df[['AVProductsInstalled','HasDetections']].groupby('AVProductsInstalled').sum
```

```
av_mal.plot.bar(y='HasDetections')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83ac2373d0>
```



AntiVirus software reduces the attack of malware as per the above histogram plot because the more the antivirus software installed the more the protection from malware attack

Correlation identification

```
df[['AVProductsInstalled','HasDetections']].corr()
```

	AVProductsInstalled	HasDetections
AVProductsInstalled	1.000000	-0.167522
HasDetections	-0.167522	1.000000

Double-click (or enter) to edit

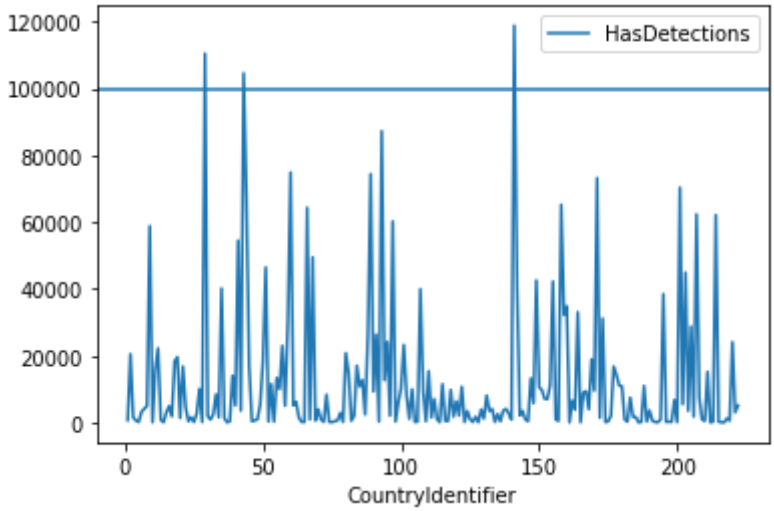
Section 5: Interesting findings (Q5)

FACT 1:

Top

```
#df[['CountryIdentifier','HasDetections']].plot(x='CountryIdentifier', y='HasDetectio
X = df[['CountryIdentifier', 'HasDetections']].groupby('CountryIdentifier').sum()
X.plot()
plt.axhline(100000)
```

```
<matplotlib.lines.Line2D at 0x7f83a1dd0350>
```

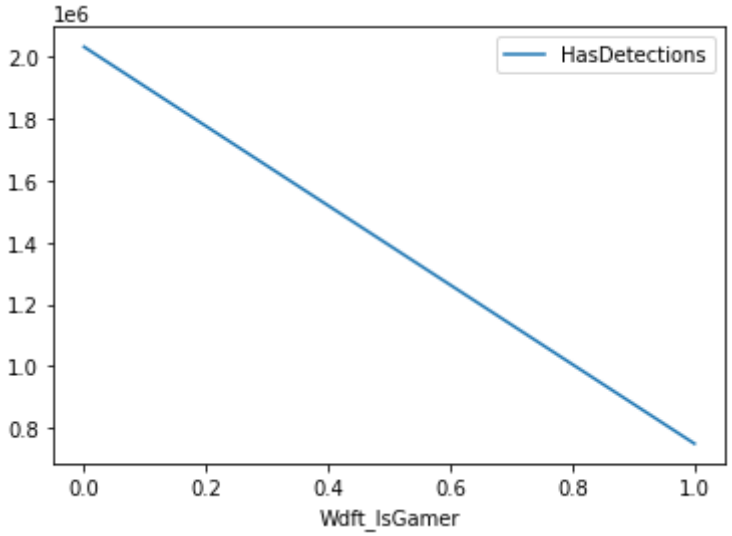


Answer:- There are 3 countries with highest malware detection

Fact-2

```
x=df[['Wdft_IsGamer','HasDetections']].groupby('Wdft_IsGamer').sum()
x.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83c12b3ad0>
```

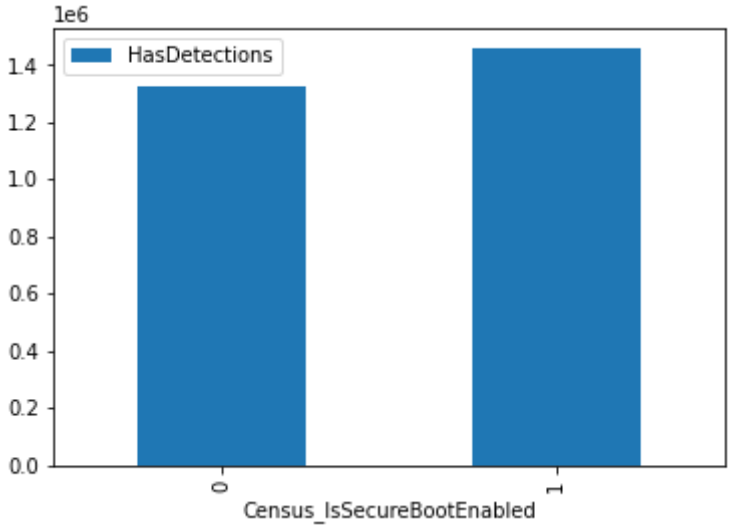


Answer: Non Gamers are having high number of malware attacks than the gamer's

FACT 3

```
sec_dete=df[['Census_IsSecureBootEnabled','HasDetections']].groupby('Census_IsSecureB
sec_dete.plot.bar(y='HasDetections')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83bdd42a10>
```



SecureBootEnabled(1) systems has more number of malware attacks than the disabled SecureBoot(0)

Section 6: Baseline modelling (Q6)

```
df.dropna(axis='index',how='any',inplace=True)
#dropping the NaN values
```

```
from sklearn.linear_model import LogisticRegression
feature_cols =['AVProductStatesIdentifier','AVProductsInstalled','IsProtected','Censu
#Feature columns
x=df[feature_cols] #dataframe with the featured columns
y=df.HasDetections #Target Label HasDetections
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=1) #Sp
Model0_logreg=LogisticRegression()
Model0_logreg.fit(X_train,y_train) #Fitting the data set
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_pred_0=Model0_logreg.predict(X_test) #Normal Prediction
y_pred0=Model0_logreg.predict_proba(X_test) #Probability of detection of 20% data
```

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test,y_pred_0) #Accuracy Score determination
```

```
error=1-score #Obtaining Error Rate
print("Accuracy Score:",score)
print("Error",error)
```

```
Accuracy Score: 0.5221519803183535
Error 0.4778480196816465
```

The Error Rate is 47.78 for the base model, which depicts the correlation in the data set is very low for the features I have considered and may be because of the dropped values.

AUC

```
from sklearn import metrics
auc = metrics.roc_auc_score(y_test, y_pred_0) #Obtaining AUC score
print(auc)
```

```
0.5104740741961882
```

```
#y_pred_proba = Model0_logreg.predict_proba(X_test)[::,1]
#auc1 = metrics.roc_auc_score(y_test, y_pred_proba)
#print(auc1)
#plotting
#fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
#plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
```

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

Cleaning & Preprocessing

```
import pandas as pd
df_new= pd.read_csv('train.csv',usecols=use_cols,dtype=dtypes) #Loading the data
feature_cols =['AVProductStatesIdentifier','AVProductsInstalled','IsProtected','Censu
# filling null places with highest occured value in that column
df_new[feature_cols] = df_new[feature_cols].apply(lambda x: x.fillna(x.value_counts()
```

Splitting the data

```
from sklearn.linear_model import LogisticRegression
x=df_new[feature_cols]
y=df_new.HasDetections
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=1)
```

Model 1 Logistic Regression

```
from sklearn.model_selection import train_test_split
Model1_logreg=LogisticRegression()
Model1_logreg.fit(X_train,y_train) #Logistic·Regression·Model·Training
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
y_pred_1=Model1_logreg.predict(X_test)
y_pred1=Model1_logreg.predict_proba(X_test) #Logistic Regression Model Prediction
```

AUC Score

```
from sklearn import metrics
auc = metrics.roc_auc_score(y_test, y_pred_1)
print(auc)
```

```
0.5317802893412631
```

```
from sklearn.metrics import accuracy_score
score=accuracy_score(y_test,y_pred_1)

print("Accuracy·Score:",score)
Error=1-score
print("Error:",Error)
```

```
Score: 0.5317085664550241
Error: 0.46829143354497593
```

Model 2- RandomForest

```
from sklearn.ensemble import RandomForestClassifier
Model2_RF = RandomForestClassifier(n_estimators=10)
Model2_RF.fit(X_train, y_train) #Fitting the Model with the data set of 80% iof train
y_pred_2=Model2_RF.predict(X_test)
y_pred2 = Model2_RF.predict_proba(X_test) #Predicting the value
#y_pred_score2 = Model2.predict_proba(X_test)
```

```
from sklearn import metrics
auc = metrics.roc_auc_score(y_test, y_pred_2)
print("AUC:",auc)
```

```
print('AUC: ',auc)
```

```
AUC: 0.5953663841155108
```

```
from sklearn.metrics import accuracy_score
score=accuracy_score(y_test,y_pred_2)
print("Accuracy Score:",score)
Error=1-score
print("Error:",Error)
```

```
Accuracy Score: 0.5952865470266441
Error: 0.4047134529733559
```

```
AUC_Score.      Error.              Accurancy.
```

model0. 0.51047 0.47784 0.52216 model1. 0.53178 0.468 0.53170 model2 0.59519 0.40471 0.59528

Section 8: Screenshots (Q8)

Double-click (or enter) to edit

```
#Use_cols without target variable for test.csv file prediction
use_cols = ["MachineIdentifier", "SmartScreen", "AVProductsInstalled", "AppVersion",
           "EngineVersion", "AVProductStatesIdentifier", "Census_OSVersion", "Census_
           "RtpStateBitfield", "Census_ProcessorModelIdentifier", "Census_PrimaryDisk
            "Census_InternalPrimaryDiagonalDisplaySizeInInches", "Wdft_RegionIdentifi
           "AvSigVersion", "IeVerIdentifier", "IsProtected", "Census_InternalPrimaryD
            "Census_OSWUAutoUpdateOptionsName", "Census_OSEdition", "Census_GenuineSt
           "Census_OEMNameIdentifier", "Census_MDC2FormFactor", "Census_FirmwareManuf
            "Census_OSBuildNumber", "Census_IsPenCapable", "Census_IsTouchEnabled", "
            "Census_SystemVolumeTotalCapacity", "Census_PrimaryDiskTotalCapacity"
           ]
```

final_model_o

```
import pickle
filename = 'final_model_1.sav'
pickle.dump(Model0_logreg, open(filename, 'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
df_test= pd.read_csv('/content/gdrive/MyDrive/test.csv',usecols=use_cols)
x_test=df_test[['AVProductStatesIdentifier','AVProductsInstalled','IsProtected','Cens
```

```
x_test=x_test.apply(lambda x: x.fillna(x.value_counts().index[0]))
x_pred=loaded_model.predict_proba(x_test)
df_test['HasDetections']=x_pred[:,1]
df_test[['MachineIdentifier','HasDetections']].to_csv('model1_log.csv',index=False)
```

final_model_1

```
import pickle
filename = 'final_model_1.sav'
pickle.dump(Model1_logreg, open(filename, 'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
df_test= pd.read_csv('/content/gdrive/MyDrive/test.csv',usecols=use_cols)
x_test=df_test[['AVProductStatesIdentifier','AVProductsInstalled','IsProtected','Cens
x_test=x_test.apply(lambda x: x.fillna(x.value_counts().index[0]))
x_pred=loaded_model.predict_proba(x_test)
df_test['HasDetections']=x_pred[:,1]
df_test[['MachineIdentifier','HasDetections']].to_csv('model1_log.csv',index=False)
```

```
0.5317085664550241
```

final_model_2

```
import pickle
filename = 'final_model_2.sav'
pickle.dump(Model2_RF, open(filename, 'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
df_test= pd.read_csv('/content/gdrive/MyDrive/test.csv',usecols=use_cols)
x_test=df_test[['AVProductStatesIdentifier','AVProductsInstalled','IsProtected','Cens
x_test=x_test.apply(lambda x: x.fillna(x.value_counts().index[0]))
x_pred=loaded_model.predict_proba(x_test)
df_test['HasDetections']=x_pred[:,1]
df_test[['MachineIdentifier','HasDetections']].to_csv('model2_RF.csv',index=False)
```

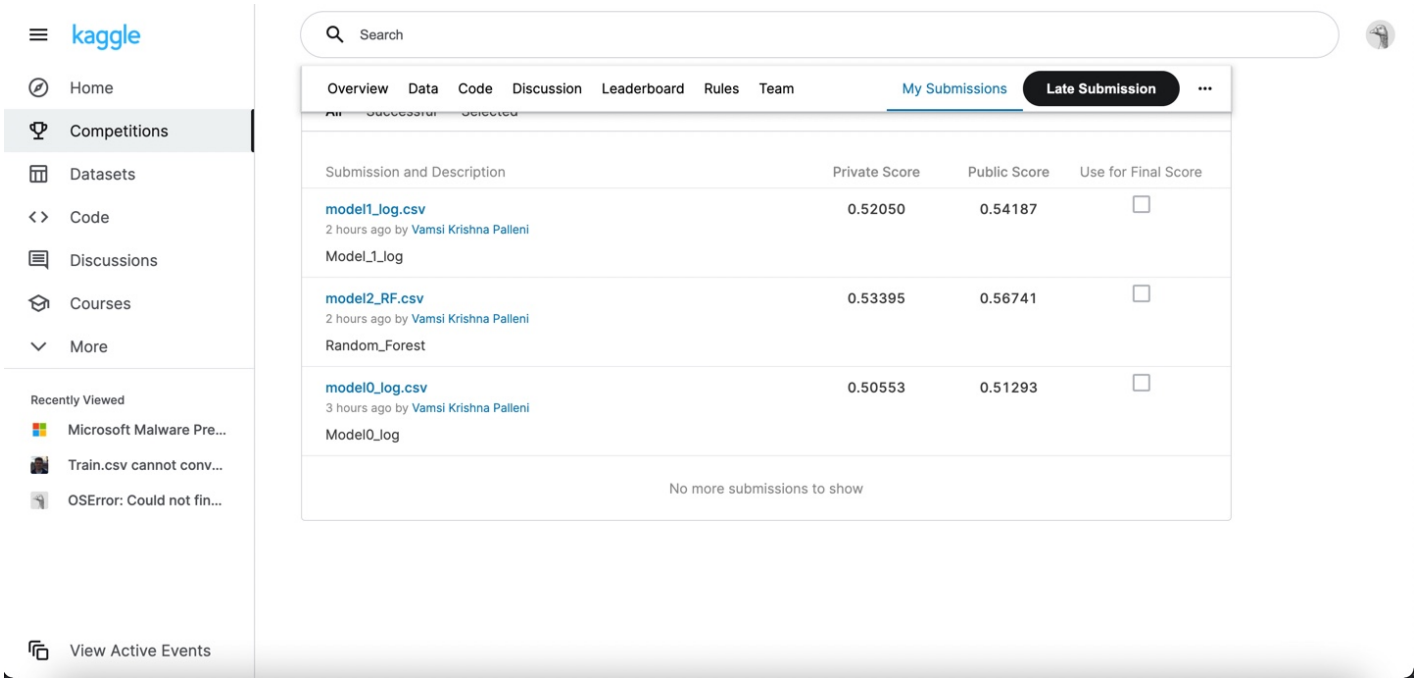
```
0.5951167322480506
```

Public Score: 0.56741

Private Score: 0.53395

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

Screenshot(s):



✓ 0s completed at 6:46 AM