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.

Section 1: Library and Data Imports (Q1)

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

In [206…

```
use_cols = ["MachineIdentifier", "SmartScreen", "AVProductsInstalled", "AppVersi
            "EngineVersion", "AVProductStatesIdentifier", "Census_OSVersion", "Ce
            "RtpStateBitfield", "Census_ProcessorModelIdentifier", "Census_Primar
             "Census_InternalPrimaryDiagonalDisplaySizeInInches", "Wdft_RegionIde
            "AvSigVersion", "IeVerIdentifier", "IsProtected", "Census_InternalPri
             "Census_OSWUAutoUpdateOptionsName", "Census_OSEdition", "Census_Genu
            "Census_OEMNameIdentifier", "Census_MDC2FormFactor", "Census_Firmware
             "Census_OSBuildNumber", "Census_IsPenCapable", "Census_IsTouchEnable
             "Census_SystemVolumeTotalCapacity", "Census_PrimaryDiskTotalCapacity
            ]
dtypes = {
        'MachineIdentifier':                                    'category',
        'ProductName':                                          'category',
        'EngineVersion':                                        'category',
        'AppVersion':                                           'category',
        'AvSigVersion':                                         'category',
        'IsBeta':                                               'int8',
        'RtpStateBitfield':                                     'float64',
        'IsSxsPassiveMode':                                     'int8',
        'DefaultBrowsersIdentifier':                            'float64',
        'AVProductStatesIdentifier':                            'float64',
        'AVProductsInstalled':                                  'float64',
        'AVProductsEnabled':                                    'float16',
        'HasTpm':                                               'int8',
        'CountryIdentifier':                                    'int16',
        'CityIdentifier':                                       'float32',
        'OrganizationIdentifier':                               'float64',
        'GeoNameIdentifier':                                    'float16',
        'LocaleEnglishNameIdentifier':                          'int8',
        'Platform':                                             'category',
        'Processor':                                            'category',
        'OsVer':                                                'category',
        'OsBuild':                                              'int16',
        'OsSuite':                                              'int16',
        'OsPlatformSubRelease':                                 'category',
        'OsBuildLab':                                           'category',
        'SkuEdition':                                           'category',
        'IsProtected':                                          'float64',
```

```
        'AutoSampleOptIn':                                  'int8',
        'PuaMode':                                          'category',
        'SMode':                                            'float16',
        'IeVerIdentifier':                                  'float64',
        'SmartScreen':                                      'category',
        'Firewall':                                         'float16',
        'UacLuaenable':                                     'float32',
        'Census_MDC2FormFactor':                            'category',
        'Census_DeviceFamily':                              'category',
        'Census_OEMNameIdentifier':                         'float64',
        'Census_OEMModelIdentifier':                        'float64',
        'Census_ProcessorCoreCount':                        'float64',
        'Census_ProcessorManufacturerIdentifier':           'float64',
        'Census_ProcessorModelIdentifier':                  'float64',
        'Census_ProcessorClass':                            'category',
        'Census_PrimaryDiskTotalCapacity':                  'float64',
        'Census_PrimaryDiskTypeName':                       'category',
        'Census_SystemVolumeTotalCapacity':                 'float64',
        'Census_HasOpticalDiskDrive':                       'int8',
        'Census_TotalPhysicalRAM':                          'float64',
        'Census_ChassisTypeName':                           'category',
        'Census_InternalPrimaryDiagonalDisplaySizeInInches':'float64',
        'Census_InternalPrimaryDisplayResolutionHorizontal':'float16',
        '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':                       'float64',
        'Census_IsFlightsDisabled':                         'float64',
        'Census_FlightRing':                                'category',
        'Census_ThresholdOptIn':                            'float16',
        'Census_FirmwareManufacturerIdentifier':            'float64',
        'Census_FirmwareVersionIdentifier':                 'float64',
        'Census_IsSecureBootEnabled':                       'int8',
        'Census_IsWIMBootEnabled':                          'float64',
        'Census_IsVirtualDevice':                           'float64',
        'Census_IsTouchEnabled':                            'int8',
        'Census_IsPenCapable':                              'int8',
        'Census_IsAlwaysOnAlwaysConnectedCapable':          'float64',
        'Wdft_IsGamer':                                     'float64',
        'Wdft_RegionIdentifier':                            'float64'
        }
```

In [207…

```
len(use_cols),len(dtypes)
```

Out[207… (40, 82)

In [208…
```
import pandas as pd
import os
```

In [95]:
```
fileName ='/Users/sbvaranasi/Documents/Fall21/DataScienceFundamentals/microsoft-
df=pd.read_csv(fileName, usecols=use_cols, dtype=dtypes)
```

In [96]:
```
df.head()
```

Out[96]:

	MachineIdentifier	EngineVersion	AppVersion	AvSigVersion	RtpStateBit
0	0000028988387b115f69f31a3bf04f09	1.1.15100.1	4.18.1807.18075	1.273.1735.0	
1	000007535c3f730efa9ea0b7ef1bd645	1.1.14600.4	4.13.17134.1	1.263.48.0	
2	000007905a28d863f6d0d597892cd692	1.1.15100.1	4.18.1807.18075	1.273.1341.0	
3	00000b11598a75ea8ba1beea8459149f	1.1.15100.1	4.18.1807.18075	1.273.1527.0	
4	000014a5f00daa18e76b81417eeb99fc	1.1.15100.1	4.18.1807.18075	1.273.1379.0	

5 rows × 39 columns

In [97]:
```
df.shape
```

Out[97]: (8921483, 39)

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

In [98]:
```
df[['Census_ProcessorCoreCount', 'Census_TotalPhysicalRAM', 'Wdft_IsGamer', 'Has
```

Out[98]:

	Census_ProcessorCoreCount	Census_TotalPhysicalRAM	Wdft_IsGamer	HasDetections
count	8.880177e+06	8.840950e+06	8.618032e+06	8.921483e+06
mean	3.989696e+00	6.115261e+03	2.835785e-01	4.997927e-01
std	2.082553e+00	5.115821e+03	4.507347e-01	5.000000e-01
min	1.000000e+00	2.550000e+02	0.000000e+00	0.000000e+00
25%	2.000000e+00	4.096000e+03	0.000000e+00	0.000000e+00
50%	4.000000e+00	4.096000e+03	0.000000e+00	0.000000e+00
75%	4.000000e+00	8.192000e+03	1.000000e+00	1.000000e+00
max	1.920000e+02	1.572864e+06	1.000000e+00	1.000000e+00

In [28]:
```
#Replacing NaN values with mode/mean
df['Wdft_IsGamer'].fillna(value = df['Wdft_IsGamer'].mode()[0], inplace = True)
df['Census_ProcessorCoreCount'].fillna(value = df['Census_ProcessorCoreCount'].m
```

In [29]:
```
#Normalizing the data
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
df['Census_ProcessorCoreCount']=min_max_scaler.fit_transform(df[['Census_Process
df['Census_TotalPhysicalRAM']=min_max_scaler.fit_transform(df[['Census_TotalPhys
```

In [30]:
```
df[['Census_ProcessorCoreCount', 'Census_TotalPhysicalRAM', 'Wdft_IsGamer', 'Has
```

Out[30]:

	Census_ProcessorCoreCount	Census_TotalPhysicalRAM	Wdft_IsGamer	HasDetections
count	8.921483e+06	8.921483e+06	8.921483e+06	8.921483e+06
mean	1.565286e-02	3.726458e-03	2.739330e-01	4.997927e-01
std	1.087815e-02	3.238363e-03	4.459751e-01	5.000000e-01
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	5.235602e-03	2.442438e-03	0.000000e+00	0.000000e+00
50%	1.570681e-02	2.442438e-03	0.000000e+00	0.000000e+00
75%	1.570681e-02	5.047027e-03	1.000000e+00	1.000000e+00
max	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

Assigning weights to the selected features : ['Census_ProcessorCoreCount', 'Census_TotalPhysicalRAM', 'Wdft_IsGamer'] as [0.4, 0.35, 0.25]

Defining Computing Power function as , computing_power = 0.4 *processor_core_count + 0.35* physical_ram + 0.25 * is_gamer

In [144…
```
df['ComputingPower'] = (0.4*df['Census_ProcessorCoreCount'] + 0.35*df['Census_To
```

In [145…
```
df[['ComputingPower','HasDetections']].describe()
```

Out[145…

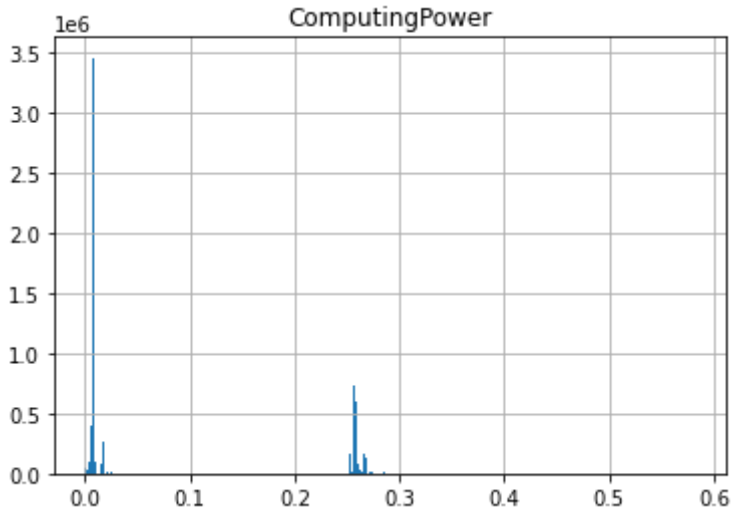
	ComputingPower	HasDetections
count	8.921483e+06	8.921483e+06
mean	7.604866e-02	4.997927e-01
std	1.123050e-01	5.000000e-01
min	0.000000e+00	0.000000e+00
25%	6.909674e-03	0.000000e+00
50%	7.593379e-03	0.000000e+00
75%	2.529491e-01	1.000000e+00
max	5.824607e-01	1.000000e+00

In [152…

```
import matplotlib.pyplot as plt

df.hist(column='ComputingPower', bins=500)
```

Out[152…

```
array([[<AxesSubplot:title={'center':'ComputingPower'}>]], dtype=object)
```

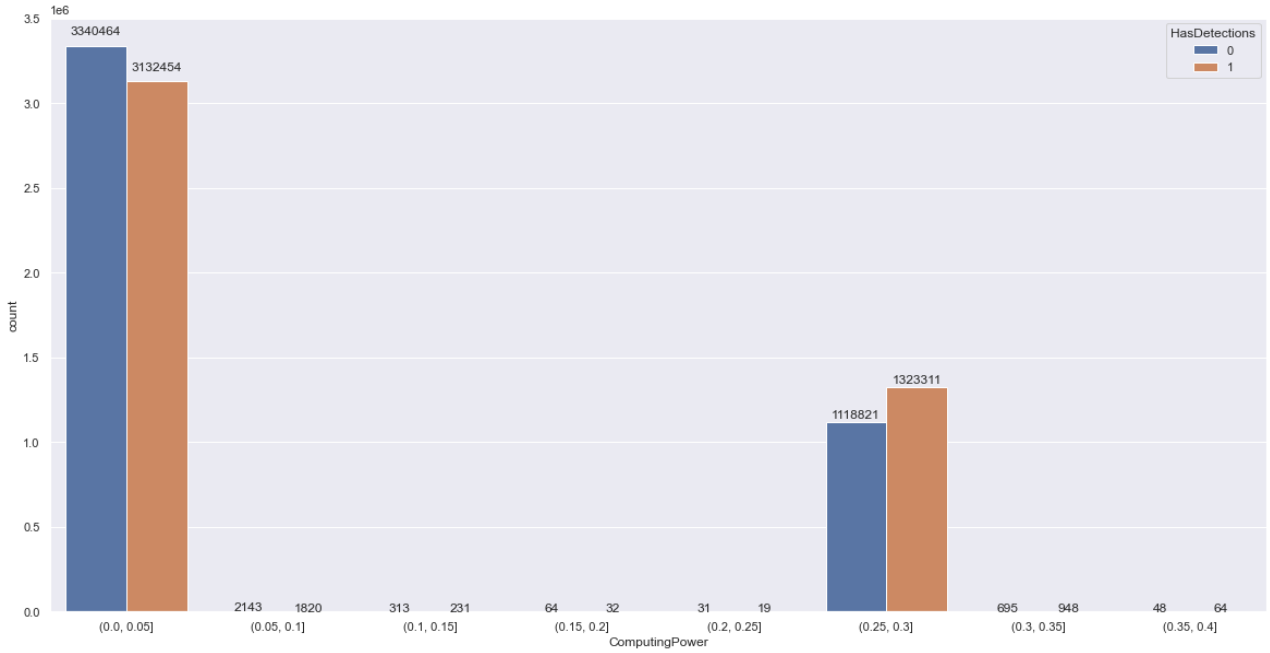


Computing power as a weighted function of Processor core count, RAM and if it's a gaming laptop. We could see a bimodal distribution of the machines using this metric computing power.

Intuition : Machines might be classified into high-power gaming laptops with high RAM/Processor cores and low-power non-gaming laptops.

In [154…

```
ranges = [0,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4]
comp_power_bin_data = df[["ComputingPower","HasDetections"]]
row_info = pd.cut(comp_power_bin_data["ComputingPower"], ranges);

sns.set(rc={'figure.figsize':(20,10)})
gph = sns.countplot(x=row_info, hue="HasDetections",data=comp_power_bin_data)
for patch in gph.patches:
    wd = patch.get_width()
    ht = patch.get_height()
    x, y = patch.get_xy()
    gph.annotate(f'{ht}', (x + wd/2, y + ht*1.02), ha='center')
```



Based on the above countplot, nothing solid can be said about computing power vs malware vulnerability.

But if we compare the ratio of malware detections, it's increasing as the computing power increases gradually. We can see the spike in ratio after 0.3 computing power.

Section 3: OS version vs Malware detected (Q3)

In [155…
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [110…
```
df[['Census_OSBuildNumber', 'Census_OSBuildRevision', 'HasDetections']].sample(1
```

Out[110…

	Census_OSBuildNumber	Census_OSBuildRevision	HasDetections
251517	17134	112	0
5985966	15063	608	0
971971	17134	167	1
8339794	15063	726	1
1510578	17134	228	1
5465846	17134	112	0
7507814	17134	228	1
618552	16299	547	1
2874451	17134	254	1
5501761	17134	285	1
253715	17134	228	0

	Census_OSBuildNumber	Census_OSBuildRevision	HasDetections
7127570	17134	165	1
3968000	16299	492	0
2617590	15063	850	1
8407067	14393	2189	1

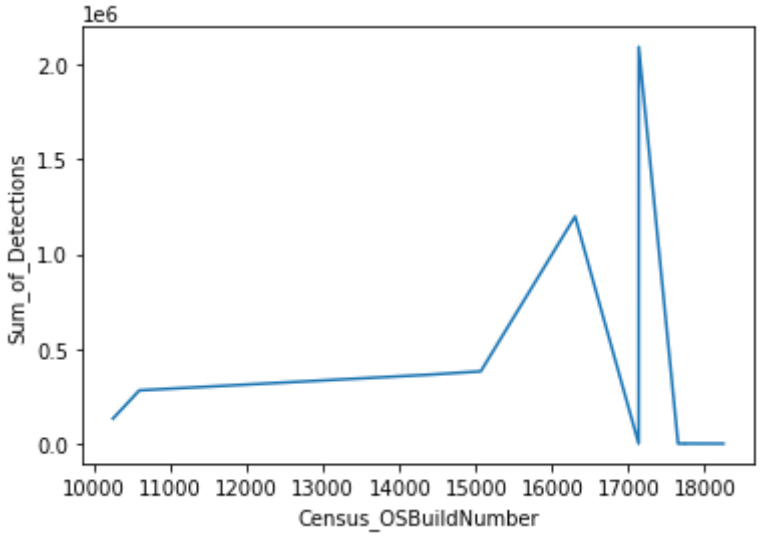
In [111…

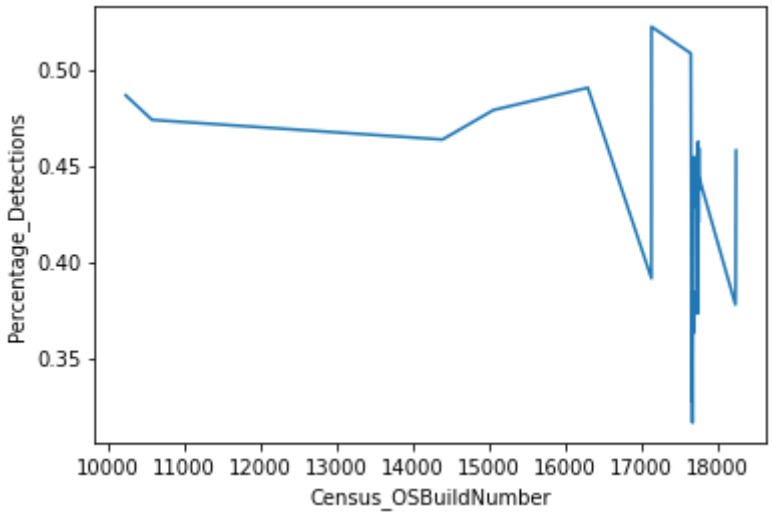
```
set(df.HasDetections)
```

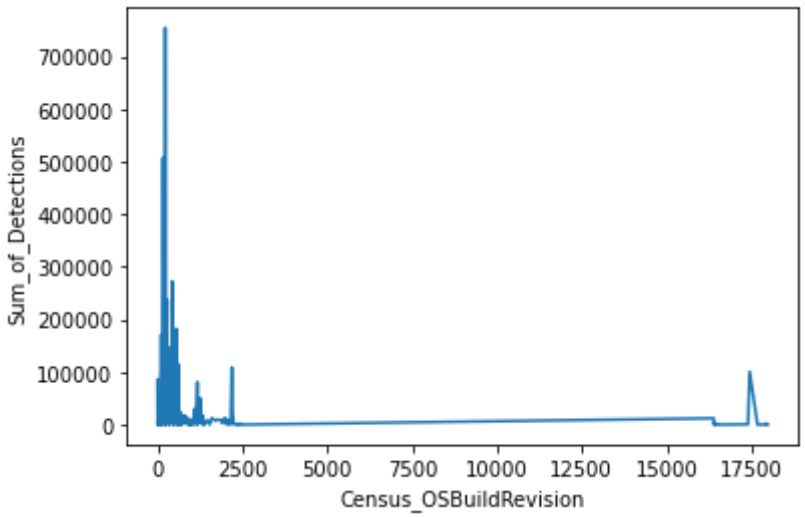
Out[111… {0, 1}

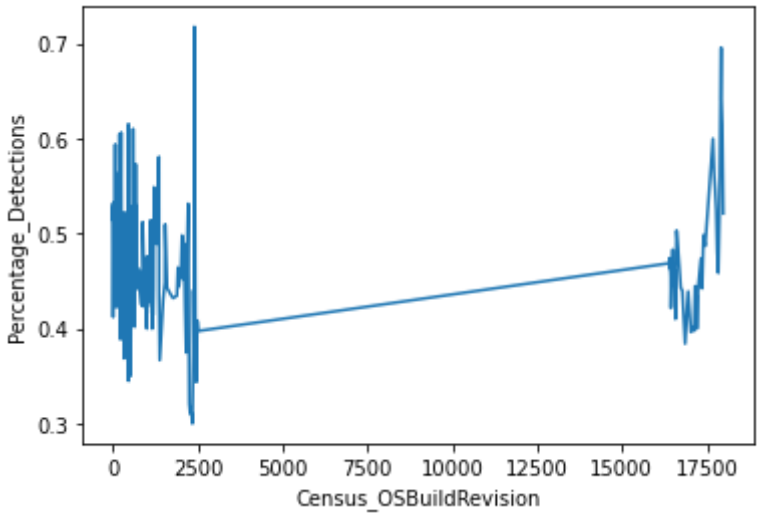
In [121…

```
group_df = df.groupby(['Census_OSBuildNumber']).agg({'HasDetections': ['sum', 'm
group_df.columns = ['detections_sum', 'detections_mean', 'detections_count']
group_df = group_df.reset_index()

#Ignoring the points with occurrences less than 100
group_df = group_df.loc[group_df['detections_count'] > 50]

plt.plot(group_df['Census_OSBuildNumber'], group_df['detections_sum'])
plt.xlabel('Census_OSBuildNumber')
plt.ylabel('Sum_of_Detections')
plt.show()

plt.plot(group_df['Census_OSBuildNumber'], group_df['detections_mean'])
plt.xlabel('Census_OSBuildNumber')
plt.ylabel('Percentage_Detections')
plt.show()

group_df = df.groupby(['Census_OSBuildRevision']).agg({'HasDetections': ['sum',
group_df.columns = ['detections_sum', 'detections_mean', 'detections_count']
group_df = group_df.reset_index()


#Ignoring the points with occurrences less than 100
group_df = group_df.loc[group_df['detections_count'] > 50]

plt.plot(group_df['Census_OSBuildRevision'], group_df['detections_sum'])
plt.xlabel('Census_OSBuildRevision')
plt.ylabel('Sum_of_Detections')
plt.show()

plt.plot(group_df['Census_OSBuildRevision'], group_df['detections_mean'])
plt.xlabel('Census_OSBuildRevision')
plt.ylabel('Percentage_Detections')
plt.show()
```


1e6
2.0
1.5
1.0
0.5
0.0
Sum_of_Detections
10000 11000 12000 13000 14000 15000 16000 17000 18000
Census_OSBuildNumber


0.50
0.45
0.40
0.35
Percentage_Detections
10000 11000 12000 13000 14000 15000 16000 17000 18000
Census_OSBuildNumber


700000
600000
500000
400000
300000
200000
100000
0
Sum_of_Detections
0 2500 5000 7500 10000 12500 15000 17500
Census_OSBuildRevision



Census_OSBuildNumber vs HasDetections:

1. Number of malware detections increased with varying slopes till 17000 OS_BuildNumber but we see a very iregular trend leading upto 17000 and till 18000 which is evident even in the % malware detections plot.
2. Probably a major release went very bad and they had to undergo frequent minor version releases where few are stable and few are not.

Census_OSBuildRevision vs HasDetections:

1. We see huge activity(high malware detections) in the initial revisions till 2500 and again we see a spike in activity > 16000 which might account to the behaviour we saw in the Census_OSBuildNumber above.

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

In [76]:
```
df.AVProductsInstalled.describe()
```

Out[76]:
```
count    8.921483e+06
mean     1.895398e-01
std      7.455208e-02
min      0.000000e+00
25%      1.428571e-01
50%      1.428571e-01
75%      2.857143e-01
max      1.000000e+00
Name: AVProductsInstalled, dtype: float64
```

In [120…
```
group_df = df.groupby(['AVProductsInstalled']).agg({'HasDetections': ['sum', 'me
group_df.columns = ['detections_sum', 'detections_mean', 'detections_count']
group_df = group_df.reset_index()
print(group_df)
```

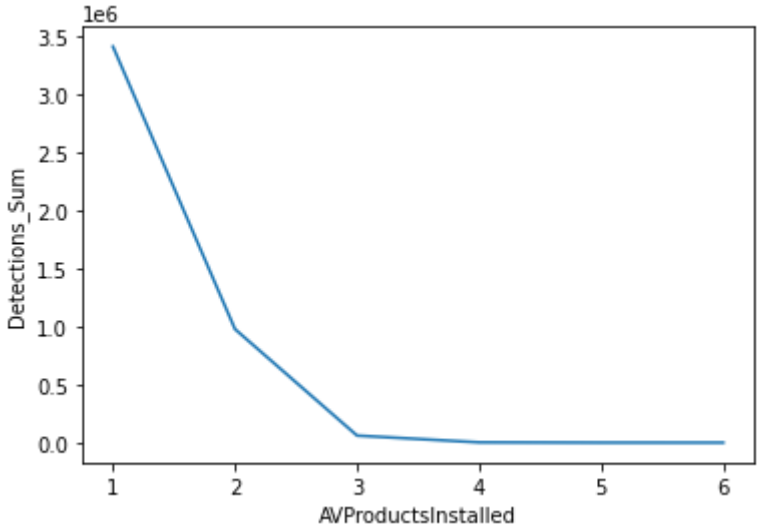
```
   AVProductsInstalled  detections_sum  detections_mean  detections_count
0                  0.0               0         0.000000                 1
1                  1.0         3406078         0.548581           6208893
```

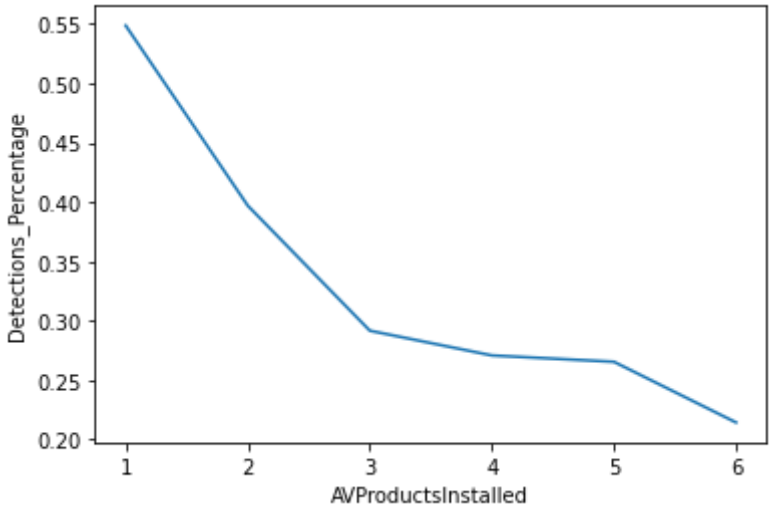
2	2.0	975996	0.396906	2459008
3	3.0	60682	0.291596	208103
4	4.0	2371	0.270755	8757
5	5.0	125	0.265393	471
6	6.0	6	0.214286	28
7	7.0	1	1.000000	1

Dropping the rows at indices 0,7 as they have too low detection_count

In [115…
```
groupdf=group_df.drop(index=[0,7])
```

In [118…
```
plt.plot(groupdf['AVProductsInstalled'], groupdf['detections_sum'])
plt.xlabel('AVProductsInstalled')
plt.ylabel('Detections_Sum')
plt.show()

plt.plot(groupdf['AVProductsInstalled'], groupdf['detections_mean'])
plt.xlabel('AVProductsInstalled')
plt.ylabel('Detections_Percentage')
plt.show()
```





As we see from the above two graphs, as the number of Antivirus products Installed increases, total malware detections and detection percentage decreases gradually.

So yes as evident from the above graphs, the number of antivirus products matter and the more products the lesser the system is vulnerable to malwares.

Section 5: Interesting findings (Q5)

In [89]:
```
df['HasDetections'].describe()
```

Out[89]:
```
count    8.921483e+06
mean     4.997927e-01
std      5.000000e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      1.000000e+00
max      1.000000e+00
Name: HasDetections, dtype: float64
```

In [176…
```
new_df = pd.read_csv(fileName, usecols=['Census_OSArchitecture','HasDetections',
                                        'Census_MDC2FormFactor', 'Platform', 'Co
```

In [177…
```
df_OSArchitecture = pd.DataFrame(new_df)
```

In [178…
```
df_OSArchitecture.head()
```

Out[178…

	CountryIdentifier	Platform	Census_MDC2FormFactor	Census_OSArchitecture	Census_IsVirtu
0	29	windows10	Desktop	amd64	
1	93	windows10	Notebook	amd64	
2	86	windows10	Desktop	amd64	
3	88	windows10	Desktop	amd64	
4	18	windows10	Notebook	amd64	

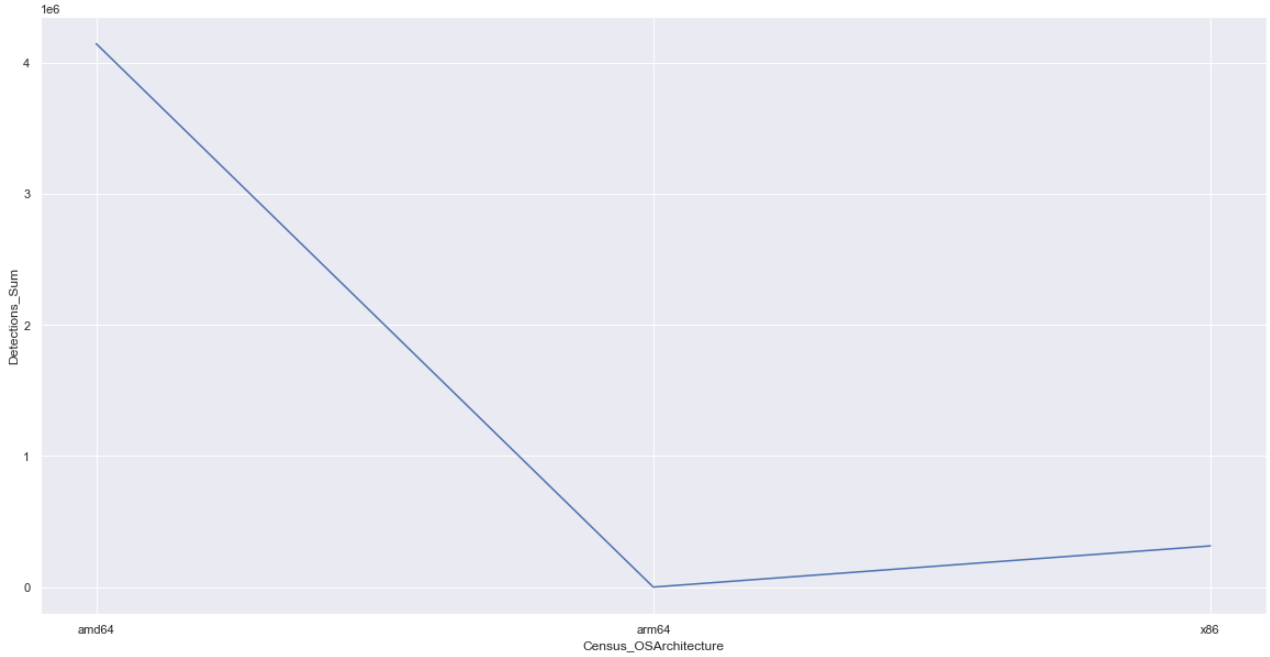
In [179…
```
df_OSArchitecture.Census_IsVirtualDevice.describe()
```

Out[179…
```
count    8.905530e+06
mean     7.039446e-03
std      8.360558e-02
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.000000e+00
Name: Census_IsVirtualDevice, dtype: float64
```

In [181…
```
arch=df_OSArchitecture.groupby(['Census_OSArchitecture']).agg({'HasDetections' :
arch.columns=['detections_sum','detections_mean']
arch=arch.reset_index()
print(arch), print(type(arch))
```

```
plt.plot(arch['Census_OSArchitecture'], arch['detections_sum'])
plt.xlabel('Census_OSArchitecture')
plt.ylabel('Detections_Sum')
plt.show()
```

```
  Census_OSArchitecture  detections_sum  detections_mean
0                 amd64         4144868         0.511341
1                 arm64               5         0.014451
2                   x86          314019         0.385180
<class 'pandas.core.frame.DataFrame'>
```



FINDING 1 : AMD64 devices are comparatively more vulnerable to malware while arm64 are the least.

In [183…

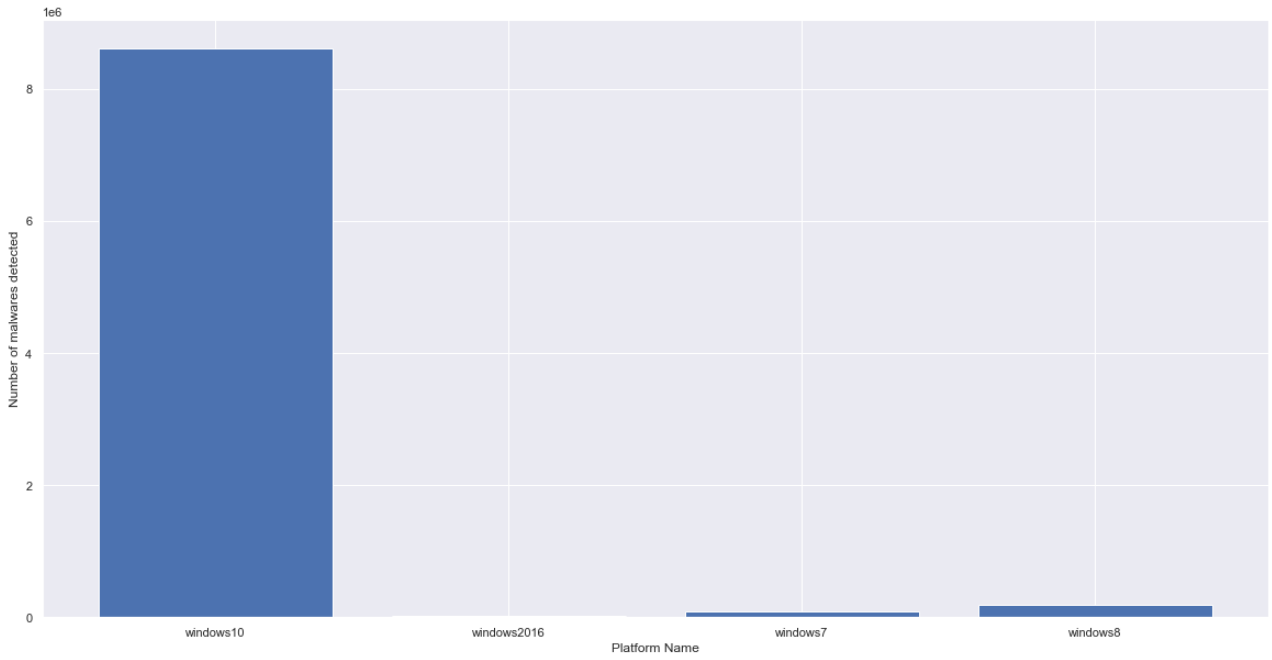
```
group_df = df_OSArchitecture.groupby(['Platform']).agg({'HasDetections': ['count
group_df.columns = ['detections_count']
group_df = group_df.reset_index()
group_df
```

Out[183…

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

In [171…

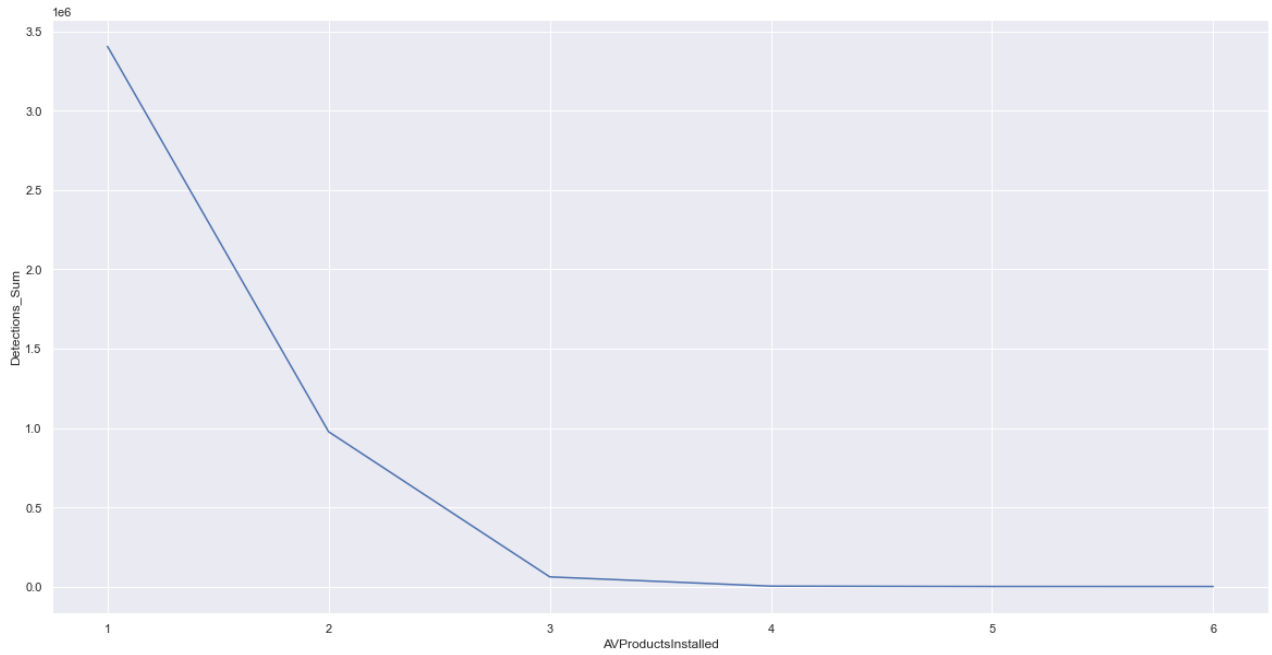
```
fig = plt.gcf()
x = np.arange(len(group_df['Platform']))
ax = plt.bar(x, group_df['detections_count'])

plt.xlabel('Platform Name')
plt.ylabel('Number of malwares detected')

plt.xticks(x, group_df['Platform'])
plt.show()
```



FINDING 2 : Surprisingly most of the malware detected is found in windows 10 computers.

In [186…

```
group_df = df.groupby(['AVProductsInstalled']).agg({'HasDetections': ['sum', 'me
group_df.columns = ['detections_sum', 'detections_mean', 'detections_count']
group_df = group_df.reset_index()

plt.plot(groupdf['AVProductsInstalled'], groupdf['detections_sum'])
plt.xlabel('AVProductsInstalled')
plt.ylabel('Detections_Sum')
plt.show()
```



FINDING 3 : The more the number of antivirus products installed on a machine, the lesser is the number of malware detected.

Section 6: Baseline modelling (Q6)

In [187…
```
import seaborn as sns
from matplotlib import pyplot as plt
```

In [188…
```
df.columns
df.isnull().sum()
```

Out[188…
```
MachineIdentifier                                       0
EngineVersion                                           0
AppVersion                                              0
AvSigVersion                                            0
RtpStateBitfield                                        0
AVProductStatesIdentifier                               0
AVProductsInstalled                                     0
CountryIdentifier                                       0
LocaleEnglishNameIdentifier                             0
OsBuildLab                                             21
IsProtected                                             0
IeVerIdentifier                                         0
SmartScreen                                       3177011
Census_MDC2FormFactor                                   0
Census_OEMNameIdentifier                                0
Census_ProcessorCoreCount                               0
Census_ProcessorModelIdentifier                         0
Census_PrimaryDiskTotalCapacity                         0
Census_PrimaryDiskTypeName                          12844
Census_SystemVolumeTotalCapacity                        0
Census_TotalPhysicalRAM                                 0
Census_InternalPrimaryDiagonalDisplaySizeInInches   47134
Census_InternalPrimaryDisplayResolutionVertical     46986
Census_OSVersion                                        0
Census_OSBuildNumber                                    0
Census_OSBuildRevision                                  0
Census_OSEdition                                        0
Census_OSInstallTypeName                                0
Census_OSWUAutoUpdateOptionsName                        0
Census_GenuineStateName                                 0
Census_ActivationChannel                                0
Census_FirmwareManufacturerIdentifier                   0
Census_IsSecureBootEnabled                              0
Census_IsTouchEnabled                                   0
Census_IsPenCapable                                     0
Census_IsAlwaysOnAlwaysConnectedCapable                 0
Wdft_IsGamer                                            0
Wdft_RegionIdentifier                                   0
HasDetections                                           0
ComputingPower                                          0
dtype: int64
```

In [123…
```
df_droppedna = df.dropna(how='any', inplace=False)
print(df_droppedna.shape)
```

```
(5430439, 39)
```

In [190…
```
columns_list = list(df_droppedna.columns)

#Columns which are IsTouchCapable, IsProtected are named as Boolean Columns
BooleanColumns = [x for x in columns_list if 'Is' in x]
```

```
#For Baseline model Model0 I chose Numerical columns by describing the dataframe
NumericalColumns = ['AVProductsInstalled','Census_ProcessorCoreCount','Census_Pr
                    'Census_SystemVolumeTotalCapacity','Census_TotalPhysicalRAM','Ce

#Ignoring categorical features for baseline model as they are not encoded yet.

Final_Features = BooleanColumns + NumericalColumns
```

In [189…

```
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder


#Running logistic regression model=> Model0 without any preprocessing of the dat
```

In [195…

```
X=df_droppedna[Final_Features]
y=df_droppedna['HasDetections']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
Model0 = LogisticRegression(max_iter=800)
Model0.fit(X_train, y_train)

y_pred = Model0.predict(X_test)
# print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(

print('Accuracy : {:.2f}'.format(metrics.accuracy_score(y_test, y_pred)))
print('AUC score : {:.2f}'.format(metrics.roc_auc_score(y_test, Model0.predict_p
```

```
Accuracy : 0.51
AUC score : 0.53
```

Error rate : 0.49 (Computed as 1-Accuracy = 1-0.51 = 0.49)

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

Plan for feature cleaning:

1. Instead of dropping null values as I did for logistic regression, I will try to compute mean, mode and accordingly replace for numerical, categorical data respectively.
2. Convert categorical features(_Identifiers, _Version) from string -> int by doing some preprocessing. (Later realized this is what label encoding does).
3. Normalize numerical values using min-max scaling or standard scaling.
4. Using groupby->counts for each categorical variable, I will try to introduce few one-hot encoding columns and compare the model's performance. (Couldn't execute this plan because introducing one-hot encoding involved an explosion of features).

In [137…

```
#Computing correlation matrix
corr = df_droppedna.corr()
```

```
plt.figure(figsize=(20,15))
sns.heatmap(corr, cmap="Greens",annot=True)

c1 = corr[(corr < 0.3) & (corr > -0.3)].abs().unstack().transpose().sort_values(
#Picking values that lie between -0.3 and 0.3 from correlation matrix
print(c1[1:25])

#Used these values from correlation matrix to check if there's any correlation b
```

```
Census_IsSecureBootEnabled                          Census_PrimaryDiskTotalCapaci
ty                    0.000015
Census_IsPenCapable                                 Census_FirmwareManufacturerId
entifier              0.000043
Census_TotalPhysicalRAM                             Census_PrimaryDiskTotalCapaci
ty                    0.000050
Census_InternalPrimaryDiagonalDisplaySizeInInches   Census_PrimaryDiskTotalCapaci
ty                    0.000079
RtpStateBitfield                                    Census_PrimaryDiskTotalCapaci
ty                    0.000094
Census_IsPenCapable                                 Census_PrimaryDiskTotalCapaci
ty                    0.000124
AVProductsInstalled                                 Census_PrimaryDiskTotalCapaci
ty                    0.000138
IsProtected                                         Census_PrimaryDiskTotalCapaci
ty                    0.000159
Census_PrimaryDiskTotalCapacity                     Census_OSBuildRevision
0.000161
Census_IsAlwaysOnAlwaysConnectedCapable             Census_PrimaryDiskTotalCapaci
ty                    0.000176
RtpStateBitfield                                    LocaleEnglishNameIdentifier
0.000233
Census_PrimaryDiskTotalCapacity                     Census_IsTouchEnabled
0.000253
                                                    Census_ProcessorCoreCount
0.000258
                                                    Census_InternalPrimaryDisplay
ResolutionVertical    0.000334
Wdft_IsGamer                                        Census_PrimaryDiskTotalCapaci
ty                    0.000348
Census_PrimaryDiskTotalCapacity                     Census_OEMNameIdentifier
0.000379
                                                    Census_FirmwareManufacturerId
entifier              0.000388
                                                    LocaleEnglishNameIdentifier
0.000440
                                                    Census_OSBuildNumber
0.000451
                                                    Wdft_RegionIdentifier
0.000614
AVProductStatesIdentifier                           Census_PrimaryDiskTotalCapaci
ty                    0.000666
Census_PrimaryDiskTotalCapacity                     CountryIdentifier
0.000772
                                                    Census_ProcessorModelIdentifi
er                    0.000800
                                                    Census_SystemVolumeTotalCapac
ity                   0.000887
dtype: float64
```

In [133…

```
# Adding categorical columns in Model1 model.
IdentifierColumns = [x for x in columns_list if 'Identifier' in x]
IdentifierColumns.remove('MachineIdentifier')

CategoricalColumns = IdentifierColumns + [x for x in columns_list if 'Version' i

# df_droppedna.isnull().sum()
```

In [134…

```
print(CategoricalColumns)
print(BooleanColumns)
print(NumericalColumns)

# print(set(CategoricalColumns).intersection(BooleanColumns))
```

```
['AVProductStatesIdentifier', 'CountryIdentifier', 'LocaleEnglishNameIdentifie
r', 'IeVerIdentifier', 'Census_OEMNameIdentifier', 'Census_ProcessorModelIdentif
ier', 'Census_FirmwareManufacturerIdentifier', 'Wdft_RegionIdentifier', 'EngineV
ersion', 'AppVersion', 'AvSigVersion', 'Census_OSVersion']
['IsProtected', 'Census_IsSecureBootEnabled', 'Census_IsTouchEnabled', 'Census_I
sPenCapable', 'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer']
['AVProductsInstalled', 'Census_ProcessorCoreCount', 'Census_PrimaryDiskTotalCap
acity', 'Census_SystemVolumeTotalCapacity', 'Census_TotalPhysicalRAM', 'Census_O
SBuildNumber', 'Census_OSBuildRevision', 'RtpStateBitfield']
```

df.astype({'IeVerIdentifier': 'float64', 'Census_OEMNameIdentifier': 'float64', 'Census_ProcessorModelIdentifier': 'float64', 'Census_FirmwareManufacturerIdentifier': 'float64', 'Wdft_RegionIdentifier': 'float64', ...}).dtypes

Changing datatypes of the above columns to float64 because of overflow while computing mean => Made this change while loading the dataset itself as a part of use_cols.

In [135…
```
df.isnull().sum()
```

Out[135…
```
MachineIdentifier                                     0
EngineVersion                                         0
AppVersion                                            0
AvSigVersion                                          0
RtpStateBitfield                                      0
AVProductStatesIdentifier                             0
AVProductsInstalled                                   0
CountryIdentifier                                     0
LocaleEnglishNameIdentifier                           0
OsBuildLab                                           21
IsProtected                                           0
IeVerIdentifier                                       0
SmartScreen                                     3177011
Census_MDC2FormFactor                                 0
Census_OEMNameIdentifier                              0
Census_ProcessorCoreCount                             0
Census_ProcessorModelIdentifier                       0
Census_PrimaryDiskTotalCapacity                       0
Census_PrimaryDiskTypeName                        12844
Census_SystemVolumeTotalCapacity                      0
Census_TotalPhysicalRAM                               0
Census_InternalPrimaryDiagonalDisplaySizeInInches     47134
Census_InternalPrimaryDisplayResolutionVertical       46986
Census_OSVersion                                      0
Census_OSBuildNumber                                  0
Census_OSBuildRevision                                0
Census_OSEdition                                      0
Census_OSInstallTypeName                              0
Census_OSWUAutoUpdateOptionsName                      0
Census_GenuineStateName                               0
Census_ActivationChannel                              0
Census_FirmwareManufacturerIdentifier                 0
Census_IsSecureBootEnabled                            0
Census_IsTouchEnabled                                 0
Census_IsPenCapable                                   0
Census_IsAlwaysOnAlwaysConnectedCapable               0
Wdft_IsGamer                                          0
Wdft_RegionIdentifier                                 0
HasDetections                                         0
dtype: int64
```

In [193…
```
# Filling nan values in the original dataframe df with mean for numerical featur
# and mode for categorical and boolean features.

def replaceNan(df):
    for item in NumericalColumns:
        df[item].fillna(value = df[item].mean(), inplace = True)

    for item in CategoricalColumns:
```

```
        df[item].fillna(value = df[item].mode()[0], inplace = True)

    for item in BooleanColumns:
        df[item].fillna(value = df[item].mode()[0], inplace = True)
    print('Replace Nan execution successful')

def labelEncoding(df):
    #Label encoding for categorical features
    labelencoder = LabelEncoder()
    for item in CategoricalColumns:
        df[item] = labelencoder.fit_transform(df[item])
    print('Label encoding execution successful')

#Min-max scaling for numerical features
def min_max_scaling(df):
    for column in NumericalColumns:
        df[column] = (df[column] - df[column].min())/(df[column].max() - df[colu
    print('Min-max scaling execution successful')

# TransformDataFrame function which does the dataframe transformation by replaci
# and Min_Max_Scaling.
def transformDataFrame(df):
    replaceNan(df)
    labelEncoding(df)
    min_max_scaling(df)
```

In [130…

```
transformDataFrame(df)
```

```
Replace Nan execution successful
Label encoding execution successful
Min-max scaling execution successful
```

In [19]:

```
df[NumericalColumns].describe()
```

Out[19]:

	AVProductsInstalled	Census_ProcessorCoreCount	Census_PrimaryDiskTotalCapacity	Censu
count	8.921483e+06	8.921483e+06	8.921483e+06	
mean	1.895398e-01	1.565286e-02	3.785402e-07	
std	7.455208e-02	1.087815e-02	5.438909e-04	
min	0.000000e+00	0.000000e+00	0.000000e+00	
25%	1.428571e-01	5.235602e-03	2.992438e-08	
50%	1.428571e-01	1.570681e-02	5.844540e-08	
75%	2.857143e-01	1.570681e-02	1.168895e-07	
max	1.000000e+00	1.000000e+00	1.000000e+00	

In [20]:

```
Final_Features_1 = Final_Features + CategoricalColumns
print(Final_Features_1,  len(Final_Features_1))
```

```
['IsProtected', 'Census_IsSecureBootEnabled', 'Census_IsTouchEnabled', 'Census_I
sPenCapable', 'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer', 'AVProd
uctsInstalled', 'Census_ProcessorCoreCount', 'Census_PrimaryDiskTotalCapacity',
'Census_SystemVolumeTotalCapacity', 'Census_TotalPhysicalRAM', 'Census_OSBuildNu
```

```
mber', 'Census_OSBuildRevision', 'RtpStateBitfield', 'AVProductStatesIdentifie
r', 'CountryIdentifier', 'LocaleEnglishNameIdentifier', 'IeVerIdentifier', 'Cens
us_OEMNameIdentifier', 'Census_ProcessorModelIdentifier', 'Census_FirmwareManufa
cturerIdentifier', 'Wdft_RegionIdentifier', 'EngineVersion', 'AppVersion', 'AvSi
gVersion', 'Census_OSVersion'] 26
```

In [64]:
```
df[Final_Features_1].describe()
```

Out[64]:

	IsProtected	Census_IsSecureBootEnabled	Census_IsTouchEnabled	Census_IsPenCapable
count	8.921483e+06	8.921483e+06	8.921483e+06	8.921483e+06
mean	9.458434e-01	4.860229e-01	1.255431e-01	3.807091e-02
std	2.263264e-01	4.998046e-01	3.313338e-01	1.913675e-01
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00
max	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

8 rows × 26 columns

In [26]:
```
X=df[Final_Features_1]
y=df['HasDetections']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
# print(X_train.shape), print(X_test.shape), print(y_train.shape)
Model1 = LogisticRegression(max_iter=1600)
Model1.fit(X_train, y_train)

y_pred = Model1.predict(X_test)

# print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(
print('Accuracy score : {:.2f}'.format(metrics.accuracy_score(y_test, y_pred)))
print('AUC score : {:.2f}'.format(metrics.roc_auc_score(y_test, Model1.predict_p
```

```
Accuracy score : 0.55
AUC score : 0.57
```

Error_Rate : 0.45 (Computed as 1- Accuracy score = 1 - 0.55 = 0.45)

In [21]:
```
# Using RandomForestClassifier for Model2
from sklearn.ensemble import RandomForestClassifier

X=df[Final_Features_1]
y=df['HasDetections']

# Instantiate model with 100 decision trees
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
Model2 = RandomForestClassifier(n_estimators=50)
Model2.fit(X_train, y_train)                                    # Train the mode

y_predict = Model2.predict(X_test)
# print('Accuracy of random forest regressor on test set: {:.2f}'.format(rf.scor
```

```
print('Accuracy : {:.2f}'.format(metrics.accuracy_score(y_test, y_predict)))
print('AUC score : {:.2f}'.format(metrics.roc_auc_score(y_test, Model2.predict_p
```

```
Accuracy : 0.62
AUC score : 0.66
```

Error_Rate : 0.38 (Computed as 1 - Accuracy score = 1 - 0.62 = 0.38)

Comparing Models

Model	Error_Rate
Model0	0.49
Model1	0.45
Model2	0.38

Comparing Models:

1. Model0 : LogisticRegression model without any preprocessing.
2. Model1 : LogisticRegression model
   a. Features selected as numerical and categorical columns by describing the data and understanding the data.
   b. Compared the selected with correlation matrix table and none of them are highly-correlated (filterd the pearson correlation matrix between -0.3 <-> 0.3 for making sure to choose features which are not highly-correlated), hence went ahead with the chosen 26 features list.
   c. Handling Nan values with mean/mode of the respective Pandas Series(based on if it's numerical/categorical/boolean).
   d. Label encoding for categorical variables.
   e. Min-max scaling for normalizing the numerical features.
3. Model2 : Random forest classifier model
   a. Preprocessing similar to Model0.
   b. Conducted exploratory data analysis with different estimators and the model seems to converge starting n_estimators=50. (Also tested with n_jobs = -1 for parallel processing).

Testing models on test.csv

In [22]:
```
use_cols_test=use_cols.remove('HasDetections')
```

In [23]:
```
testFile ='/Users/sbvaranasi/Documents/Fall21/DataScienceFundamentals/microsoft-
test_df=pd.read_csv(testFile, usecols=use_cols_test, dtype=dtypes)
```

In [53]:
```
transformDataFrame(test_df)
X_test=test_df[Final_Features_1]
# Machine_ID=test_df['MachineIdentifier']
```

```
my_submission=pd.DataFrame({'MachineIdentifier': Machine_ID, 'HasDetections':Mod
my_submission
```

In [54]:

```
my_submission_model1=pd.DataFrame({'MachineIdentifier': Machine_ID, 'HasDetectio
my_submission_model1
```

Out[54]:

	MachineIdentifier	HasDetections
0	0000010489e3af074adeac69c53e555e	0.558093
1	00000176ac758d54827acd545b6315a5	0.557698
2	0000019dcefc128c2d4387c1273dae1d	0.518351
3	0000055553dc51b1295785415f1a224d	0.544380
4	00000574cefffeca83ec8adf9285b2bf	0.607559
...	...	...
7853248	fffff8c0e065c468a2373f7afd5e7674	0.582906
7853249	fffff90b27a1248b6fffc7a535bd736c	0.592425
7853250	fffffa6a956c17ddbabca53d4ab708ae	0.655775
7853251	fffffad7b6c8196ec5cae634406c0d4f	0.463636
7853252	fffffbd305a90eb0f93ee4f30a39c736	0.570924

7853253 rows × 2 columns

In [39]:

```
# my_submission.to_csv('model2_submission.csv', index=False)
my_submission_model1.to_csv('model1_submission.csv', index=False)
```

Section 8: Screenshots (Q8)

Public Score: 0.56812

Private Score: 0.55044

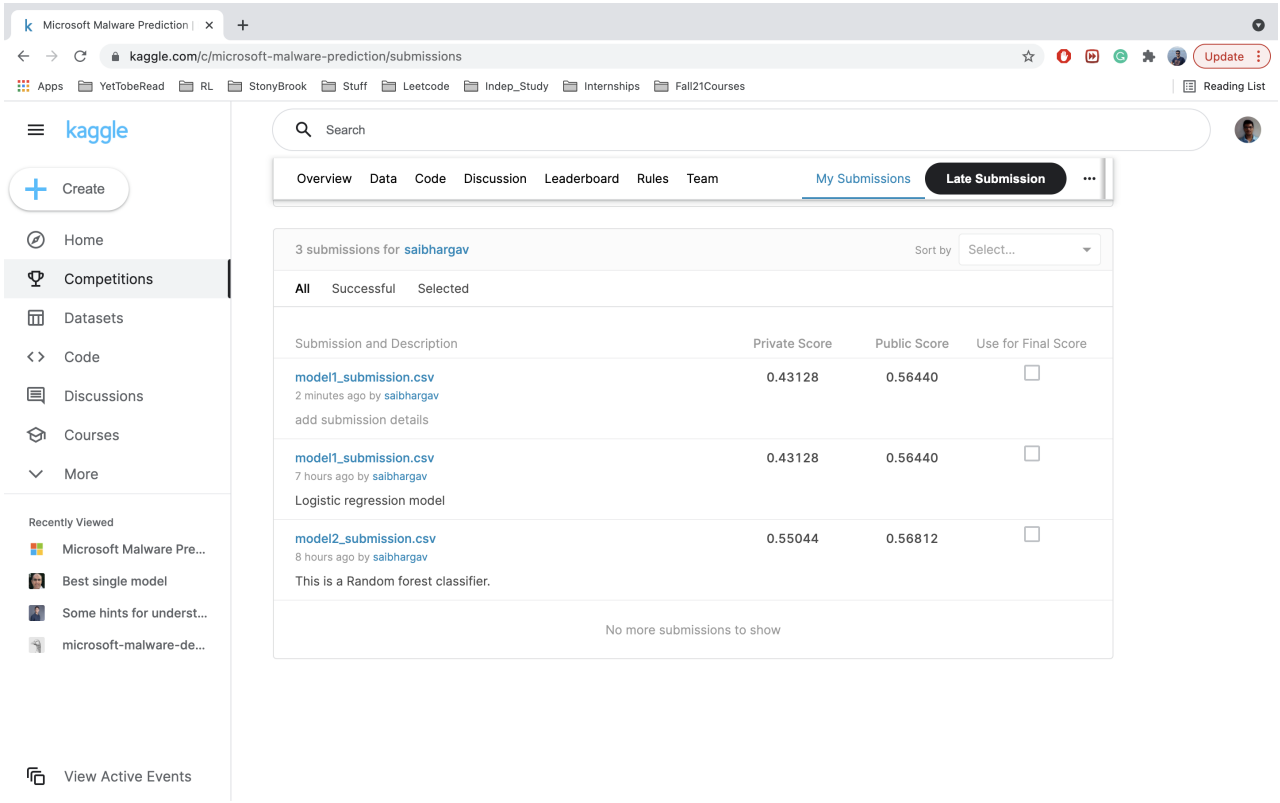
Kaggle profile link: https://www.kaggle.com/saibvara/account

Screenshot(s):

In [205…

```
from IPython.display import Image
Image("Screen Shot 2021-09-23 at 9.10.28 AM.png")
```

Out[205…


Microsoft Malware Prediction
kaggle.com/c/microsoft-malware-prediction/submissions
Update
Apps
YetTobeRead
RL
StonyBrook
Stuff
Leetcode
Indep_Study
Internships
Fall21Courses
Reading List
kaggle
Search
Create
Home
Competitions
Datasets
Code
Discussions
Courses
More
Recently Viewed
Microsoft Malware Pre...
Best single model
Some hints for underst...
microsoft-malware-de...
View Active Events
Overview
Data
Code
Discussion
Leaderboard
Rules
Team
My Submissions
Late Submission
3 submissions for saibhargav
Sort by
Select...
All
Successful
Selected
Submission and Description
Private Score
Public Score
Use for Final Score
model1_submission.csv
2 minutes ago by saibhargav
add submission details
0.43128
0.56440
model1_submission.csv
7 hours ago by saibhargav
Logistic regression model
0.43128
0.56440
model2_submission.csv
8 hours ago by saibhargav
This is a Random forest classifier.
0.55044
0.56812
No more submissions to show

In [ ]: