cse519_hw2_Baraskar_Gauri_114395197

September 23, 2021

0.1 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.

0.2 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.

[133]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-133-bf1647b42bbd> in <module>
----> 1 from google.colab import drive
      2 drive.mount("/content/drive")

ModuleNotFoundError: No module named 'google'
```

[109]:
```
# First mount your drive before running these cells.
# Create a folder for the this HW and change to that dir
%cd drive/MyDrive/CSE\ 519\ fall\ 2021/HW2
```

```
[Errno 2] No such file or directory: 'drive/MyDrive/CSE 519 fall 2021/HW2'
/Users/gauribaraskar/Desktop/DSF/HW2
```

[4]:
```
!pip install -q kaggle
```

[5]:
```
from google.colab import files
# Create a new API token under "Account" in the kaggle webpage and download the␣
↪json file
# Upload the file by clicking on the browse
files.upload()
```

```
<IPython.core.display.HTML object>
```

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

[5]: {'kaggle.json':
b'{"username":"gauribaraskar","key":"44e7ac8ba4588bd2085916c112ccfadf"}'}

[11]:
```
!rm -rf /root/.kaggle.
!mkdir /root/.kaggle
!mv kaggle.json /root/.kaggle/kaggle.json
!ls /root/.kaggle/kaggle.json
```

```
mkdir: cannot create directory '/root/.kaggle': File exists
/root/.kaggle/kaggle.json
```

[15]:
```
!kaggle competitions download -c microsoft-malware-prediction
```

```
Warning: Looks like you're using an outdated API Version, please consider
updating (server 1.5.12 / client 1.5.4)
Downloading sample_submission.csv.zip to /content/drive/My Drive/CSE 519 fall
2021/HW2
 92% 123M/134M [00:00<00:00, 128MB/s]
100% 134M/134M [00:01<00:00, 131MB/s]
Downloading train.csv.zip to /content/drive/My Drive/CSE 519 fall 2021/HW2
100% 768M/768M [00:07<00:00, 102MB/s]
100% 768M/768M [00:07<00:00, 107MB/s]
Downloading test.csv.zip to /content/drive/My Drive/CSE 519 fall 2021/HW2
 99% 664M/672M [00:07<00:00, 61.6MB/s]
100% 672M/672M [00:07<00:00, 93.0MB/s]
```

0.3 Section 1: Library and Data Imports (Q1)

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

[133]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pyplot import figure
import pandas as pd
from scipy.stats import norm
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
import pickle
import tabulate

%matplotlib inline
```

[2]:
```
use_cols = ["MachineIdentifier", "SmartScreen", "AVProductsInstalled",␣
↪"AppVersion", "CountryIdentifier", "Census_OSInstallTypeName",␣
↪"Wdft_IsGamer",
```

```
           "EngineVersion", "AVProductStatesIdentifier", "Census_OSVersion",␣
↪"Census_TotalPhysicalRAM", "Census_ActivationChannel",
           "RtpStateBitfield", "Census_ProcessorModelIdentifier",␣
↪"Census_PrimaryDiskTotalCapacity",
            "Census_InternalPrimaryDiagonalDisplaySizeInInches",␣
↪"Wdft_RegionIdentifier", "LocaleEnglishNameIdentifier",
           "AvSigVersion", "IeVerIdentifier", "IsProtected",␣
↪"Census_InternalPrimaryDisplayResolutionVertical",␣
↪"Census_PrimaryDiskTypeName",
            "Census_OSWUAutoUpdateOptionsName", "Census_OSEdition",␣
↪"Census_GenuineStateName", "Census_ProcessorCoreCount",
           "Census_OEMNameIdentifier", "Census_MDC2FormFactor",␣
↪"Census_FirmwareManufacturerIdentifier", "OsBuildLab",␣
↪"Census_OSBuildRevision",
            "Census_OSBuildNumber", "Census_IsPenCapable",␣
↪"Census_IsTouchEnabled", "Census_IsAlwaysOnAlwaysConnectedCapable",␣
↪"Census_IsSecureBootEnabled",
            "Census_SystemVolumeTotalCapacity",␣
↪"Census_PrimaryDiskTotalCapacity", "HasDetections"
           ]
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'
        }
```

Loading data in the frame using read_csv function in pandas and then printing head of the dataframe.

[3]:
```
df = pd.read_csv('train.csv',usecols=use_cols)
df.head()
```

[3]:
```
                 MachineIdentifier EngineVersion       AppVersion  \
0  0000028988387b115f69f31a3bf04f09   1.1.15100.1  4.18.1807.18075
1  000007535c3f730efa9ea0b7ef1bd645   1.1.14600.4     4.13.17134.1
2  000007905a28d863f6d0d597892cd692   1.1.15100.1  4.18.1807.18075
3  00000b11598a75ea8ba1beea8459149f   1.1.15100.1  4.18.1807.18075
4  000014a5f00daa18e76b81417eeb99fc   1.1.15100.1  4.18.1807.18075

   AvSigVersion  RtpStateBitfield  AVProductStatesIdentifier  \
0  1.273.1735.0               7.0                    53447.0
1    1.263.48.0               7.0                    53447.0
2  1.273.1341.0               7.0                    53447.0
3  1.273.1527.0               7.0                    53447.0
4  1.273.1379.0               7.0                    53447.0

   AVProductsInstalled  CountryIdentifier  LocaleEnglishNameIdentifier  \
0                  1.0                 29                          171
1                  1.0                 93                           64
2                  1.0                 86                           49
3                  1.0                 88                          115
4                  1.0                 18                           75

                              OsBuildLab  …  Census_GenuineStateName  \
0  17134.1.amd64fre.rs4_release.180410-1804  …               IS_GENUINE
1  17134.1.amd64fre.rs4_release.180410-1804  …                  OFFLINE
2  17134.1.amd64fre.rs4_release.180410-1804  …               IS_GENUINE
3  17134.1.amd64fre.rs4_release.180410-1804  …               IS_GENUINE
4  17134.1.amd64fre.rs4_release.180410-1804  …               IS_GENUINE

   Census_ActivationChannel Census_FirmwareManufacturerIdentifier  \
0                    Retail                                 628.0
```

```
1                    Retail                                     628.0
2                OEM:NONSLP                                     142.0
3                OEM:NONSLP                                     355.0
4                    Retail                                     355.0

  Census_IsSecureBootEnabled  Census_IsTouchEnabled  Census_IsPenCapable  \
0                          0                      0                    0
1                          0                      0                    0
2                          0                      0                    0
3                          0                      0                    0
4                          0                      0                    0

   Census_IsAlwaysOnAlwaysConnectedCapable  Wdft_IsGamer  \
0                                      0.0           0.0
1                                      0.0           0.0
2                                      0.0           0.0
3                                      0.0           0.0
4                                      0.0           0.0

  Wdft_RegionIdentifier  HasDetections
0                  10.0              0
1                   8.0              0
2                   3.0              0
3                   3.0              1
4                   1.0              1

[5 rows x 39 columns]
```

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

(2a) I have selected columns for computer power calculation based on intuition. A computer's power is usually defined by its processsing power and memory capacity. Gaming laptops can also be considered high power because of memory and real-time processing they require to run games.Hence, I have selected the following columns from the dataset.

1. Census_ProcessorCoreCount
2. Census_TotalPhysicalRAM
3. Census_PrimaryDiskTotalCapacity
4. Census_SystemVolumeTotalCapacity
5. Wdft_IsGamer

[4]:
```
computer_power_cols = [
    'Census_ProcessorCoreCount',
    'Census_TotalPhysicalRAM',
    'Census_PrimaryDiskTotalCapacity',
    'Census_SystemVolumeTotalCapacity',
    'Wdft_IsGamer',
    'HasDetections'
```

```
]
```

[5]:
```
computer_power_df = df[computer_power_cols].copy()
computer_power_df.head()
```

[5]:
```
   Census_ProcessorCoreCount  Census_TotalPhysicalRAM  \
0                        4.0                   4096.0
1                        4.0                   4096.0
2                        4.0                   4096.0
3                        4.0                   4096.0
4                        4.0                   6144.0

   Census_PrimaryDiskTotalCapacity  Census_SystemVolumeTotalCapacity  \
0                         476940.0                          299451.0
1                         476940.0                          102385.0
2                         114473.0                          113907.0
3                         238475.0                          227116.0
4                         476940.0                          101900.0

   Wdft_IsGamer  HasDetections
0           0.0              0
1           0.0              0
2           0.0              0
3           0.0              1
4           0.0              1
```

Since some values are nan, those rows should be dropped. After checking unique values for each row I observed that only Census_ProcessorCoreCount has 0/nan values and hence in the next step I have dropped this row. This will also be helpful because I am taking log in the next step.

[6]:
```
computer_power_df.dropna(subset=['Census_ProcessorCoreCount'])
```

[6]:
```
         Census_ProcessorCoreCount  Census_TotalPhysicalRAM  \
0                              4.0                   4096.0
1                              4.0                   4096.0
2                              4.0                   4096.0
3                              4.0                   4096.0
4                              4.0                   6144.0
…                                …                        …
8921478                        4.0                   4096.0
8921479                        2.0                   2048.0
8921480                        8.0                   8192.0
8921481                        2.0                   4096.0
8921482                        4.0                   6144.0

         Census_PrimaryDiskTotalCapacity  Census_SystemVolumeTotalCapacity  \
0                               476940.0                          299451.0
1                               476940.0                          102385.0
```

```
2                                     114473.0                              113907.0
3                                     238475.0                              227116.0
4                                     476940.0                              101900.0
…                                        …                                     …
8921478                               953869.0                              936175.0
8921479                                76293.0                               75741.0
8921480                               244198.0                              242989.0
8921481                               476940.0                              463486.0
8921482                               953869.0                              637127.0

         Wdft_IsGamer  HasDetections
0                 0.0              0
1                 0.0              0
2                 0.0              0
3                 0.0              1
4                 0.0              1
…                  …              …
8921478           0.0              1
8921479           0.0              0
8921480           0.0              1
8921481           0.0              1
8921482           0.0              0

[8880177 rows x 6 columns]
```

Since some of the above values are too big, it makes sense to take log and reduce them to smaller values easier for calculation. The next steps reduce all columns except HasDetections and Wdft_IsGamer to their corresponding log values.

I started by taking uniform weights for each of my selected columns in computer_power_cols. Due to distribution not being an exact bell curve I tried to adjust weights by giving more weight to processor count and gaming laptop indicator.

Final formula = 0.3 x Census_ProcessorCoreCount_log + 0.1 x Census_TotalPhysicalRAM_log + 0.1 x Census_PrimaryDiskTotalCapacity_log + 0.2 x Census_SystemVolumeTotalCapacity_log + 0.3 x Wdft_IsGamer

[7]:
```
for col in computer_power_cols:
    if col not in ['HasDetections','Wdft_IsGamer']:
        string = col + "_log"
        computer_power_df[string] = np.log2(computer_power_df[col])
```

```
/Users/gauribaraskar/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/arraylike.py:358: RuntimeWarning: divide by zero
encountered in log2
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

[8]:
```
computer_power_df = computer_power_df[(computer_power_df != 0).all(1)]
```

[9]:
```
computer_power_cols_log = [
    'Census_ProcessorCoreCount_log',
    'Census_TotalPhysicalRAM_log',
    'Census_PrimaryDiskTotalCapacity_log',
    'Census_SystemVolumeTotalCapacity_log',
]
```

[10]:
```
computer_power_df['power'] = 0.
↪3*computer_power_df['Census_ProcessorCoreCount_log'] + 0.
↪1*computer_power_df['Census_TotalPhysicalRAM_log']+ 0.
↪1*computer_power_df['Census_PrimaryDiskTotalCapacity_log']+ 0.
↪2*computer_power_df['Census_SystemVolumeTotalCapacity_log'] + 0.
↪3*computer_power_df['Wdft_IsGamer']
computer_power_df.head()
```

[10]:
```
    Census_ProcessorCoreCount  Census_TotalPhysicalRAM  \
9                         4.0                   8192.0
14                        8.0                   8192.0
19                        4.0                   4096.0
21                        4.0                   8192.0
32                        2.0                   2048.0

    Census_PrimaryDiskTotalCapacity  Census_SystemVolumeTotalCapacity  \
9                          953869.0                          203252.0
14                         953869.0                          252439.0
19                         476940.0                          432003.0
21                         477102.0                          363314.0
32                         476940.0                          450063.0

    Wdft_IsGamer  HasDetections  Census_ProcessorCoreCount_log  \
9            1.0              1                            2.0
14           1.0              1                            3.0
19           1.0              1                            2.0
21           1.0              1                            2.0
32           1.0              1                            1.0

    Census_TotalPhysicalRAM_log  Census_PrimaryDiskTotalCapacity_log  \
9                          13.0                            19.863432
14                         13.0                            19.863432
19                         12.0                            18.863448
21                         13.0                            18.863938
32                         11.0                            18.863448

    Census_SystemVolumeTotalCapacity_log     power
9                              17.632910  7.712925
14                             17.945575  8.075458
19                             18.720682  7.730481
```

```
21                                 18.470857  7.780565
32                                 18.779767  7.342298
```

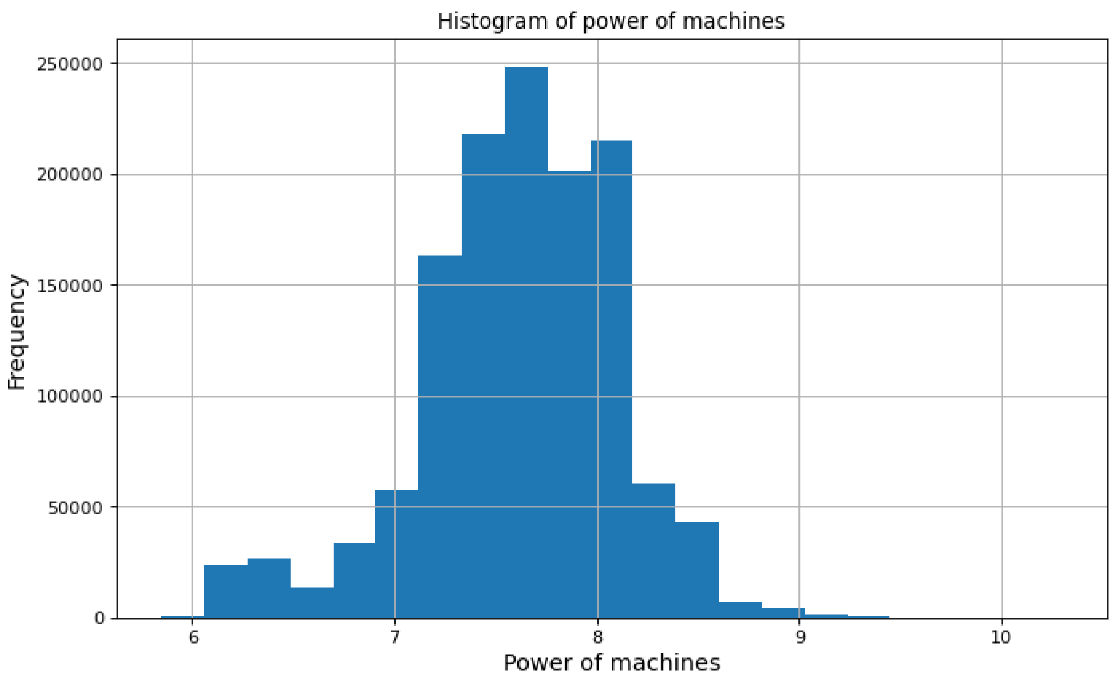
[11]:
```
print("Mean power: ",computer_power_df['power'].mean())
print("Min power: ",computer_power_df['power'].min())
print("Max power: ",computer_power_df['power'].max())
print("Variance of power: ",computer_power_df.var()['power'])
print("Standard deviation of power: ",computer_power_df.std()['power'])
```

```
Mean power:  7.624206023649132
Min power:  5.8479909841683755
Max power:  10.297702867409537
Variance of power:  0.2368349435771742
Standard deviation of power:  0.48665690540377027
```

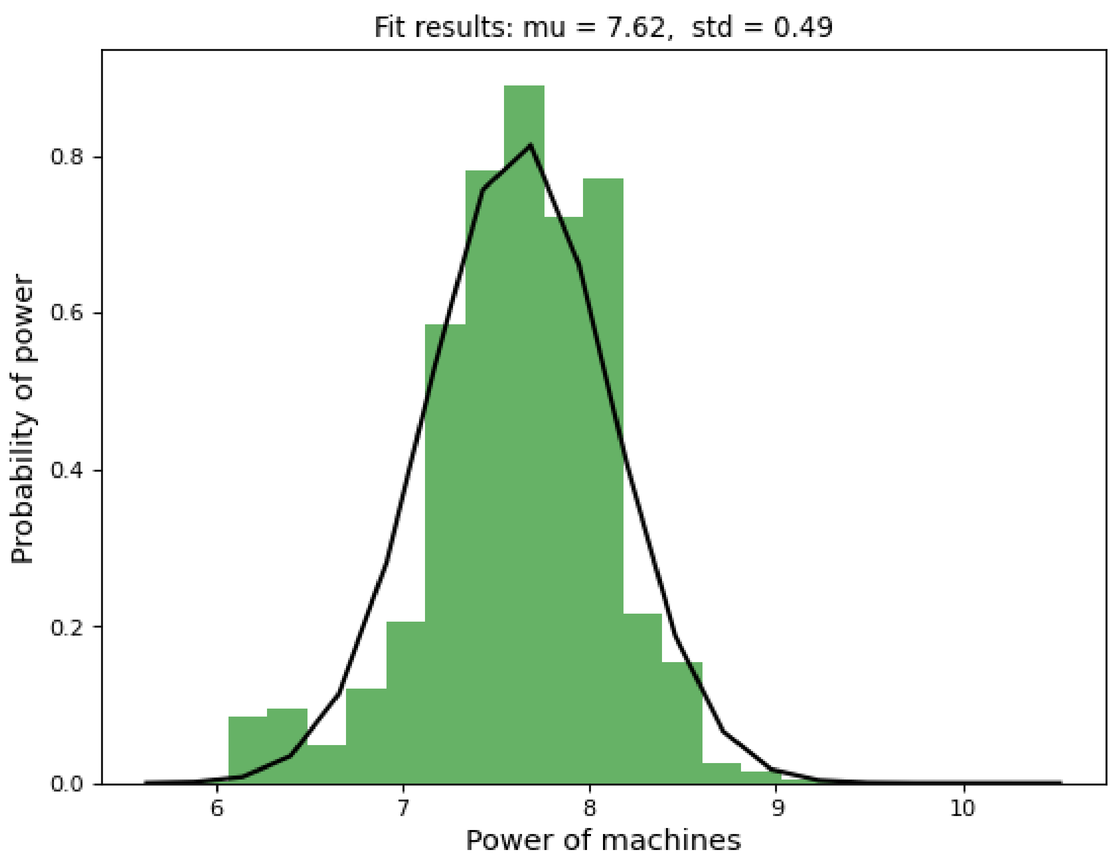
The below cell shows the distribution of computer power with a histogram. We can see the values for power lie roughly between 6 to 10. The distribution can be approximated to a bell curve which has been done later. The most number of machines lie between 7-8 which is near the mean of the histogram.

[12]:
```
figure(figsize=(10, 6), dpi=80)

computer_power_df['power'].hist(bins=21)
plt.xlabel('Power of machines', fontsize=13)
plt.ylabel('Frequency', fontsize=13)
plt.title("Histogram of power of machines")
```

[12]:
```
Text(0.5, 1.0, 'Histogram of power of machines')
```



[13]:
```
computer_power_df = computer_power_df.dropna(subset=['power'])

figure(figsize=(8, 6), dpi=80)

data = computer_power_df['power']
mu, std = norm.fit(data)
plt.hist(data, bins=21, density=True, alpha=0.6, color='g')
plt.xlabel('Power of machines', fontsize=13)
plt.ylabel('Probability of power', fontsize=13)
plt.title("Histogram of power of machines")

xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 20)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)
title = "Fit results: mu = %.2f,  std = %.2f" % (mu, std)
plt.title(title)

plt.show()
```



Observations: 1. The distribution of power can be seen as a normal distribution. 2. Roughly, it has a standard deviation of 0.49. 3. Most machines have power lying in the range [7,8].

(2b) Next we visualise if there is any relation between power of the machine and malware. Since the values of frequency are spread out over a wide range it will be better if we plot percentages of machines in different bins.

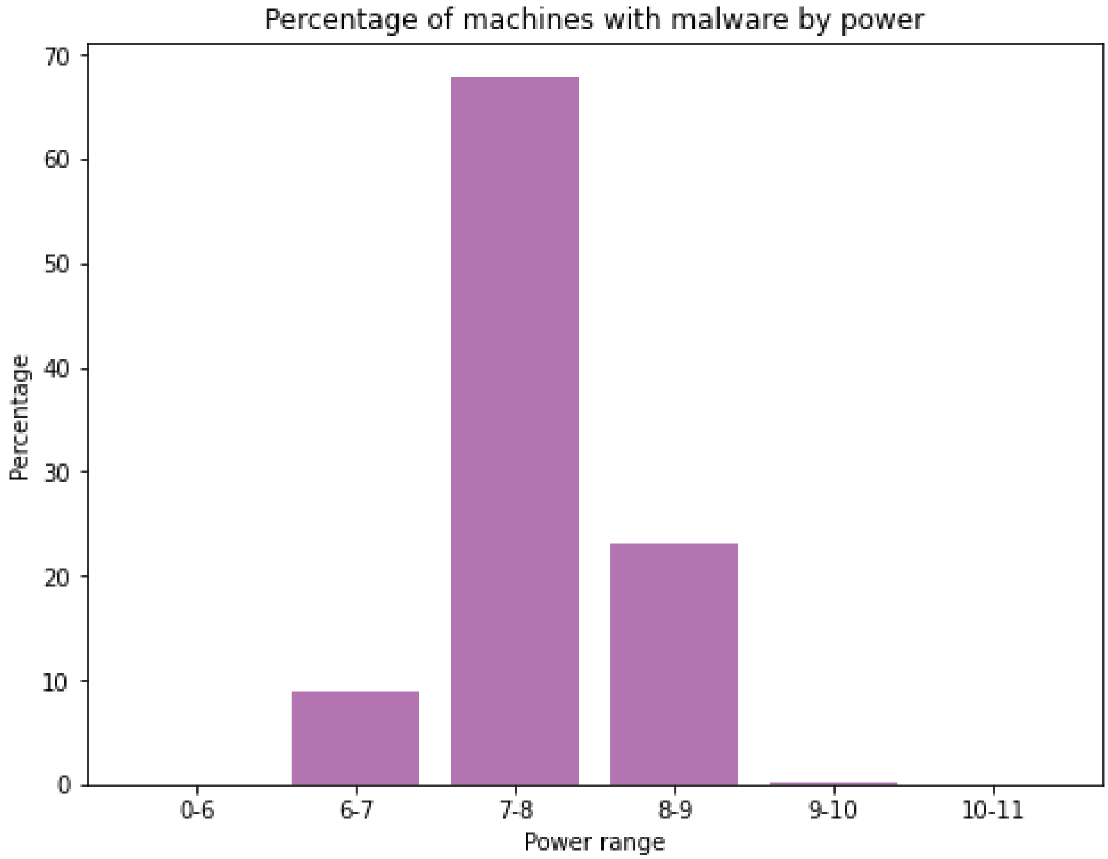
[14]:
```
'''Bins to distribute power'''
ranges = [0,6,7,8,9,10,11]
df_group_by_power = computer_power_df.groupby(pd.cut(computer_power_df.power,␣
↪ranges),as_index=False)['HasDetections'].sum()
column_sum = df_group_by_power['HasDetections'].sum()
df_group_by_power['HasDetections_percent'] = df_group_by_power['HasDetections'].
↪div(column_sum).mul(100)
```

[15]:
```
df_group_by_power.head()
```

[15]:

	HasDetections	HasDetections_percent
0	35	0.002657
1	118156	8.970635
2	893034	67.800890
3	304654	23.129928
4	1260	0.095662

[16]:
```
fig = plt.gcf()
fig.set_size_inches(8,6)

bars = ('0-6', '6-7', '7-8', '8-9', '9-10','10-11')
x_pos = np.arange(len(bars))

plt.bar(x_pos, df_group_by_power['HasDetections_percent'], color = (0.5,0.1,0.
↪5,0.6))

plt.title('Percentage of machines with malware by power')
plt.xlabel('Power range')
plt.ylabel('Percentage')

plt.xticks(x_pos, bars)

plt.show()
plt.clf()
```



<Figure size 432x288 with 0 Axes>

Observations 1. From the above graph I can see that an average machine defined by our formula is more likely to be affected by malware. 2. Machines that are defined to have very low / high power are usually not affected my malware. 3. The machines with power within 0.6 range of mean can be seen to have 70% of all malware detections.

0.5 Section 3: OS version vs Malware detected (Q3)

To observe the pattern between OS Build number and malware detections create a new dataframe that has the previously mentioned two columns.

[17]:
```
cols = ['Census_OSBuildNumber','HasDetections']
df_os_version_vs_malware = df[cols].copy()
```

[18]:
```
df_number = df_os_version_vs_malware.
↪groupby(df_os_version_vs_malware['Census_OSBuildNumber']).
↪agg({'HasDetections':'sum'}).reset_index()
df_number.head()
```

[18]:

	Census_OSBuildNumber	HasDetections
0	7600	0
1	7601	4
2	9200	3
3	9600	3
4	10240	132103

[19]:
```
df_number.shape
```

[19]: (165, 2)

[20]:
```
print("Min: ", df_number['HasDetections'].min())
print("Max: ", df_number['HasDetections'].max())
```

```
Min:  0
Max:  2092768
```

[21]:
```
df_os_with_0_malware = df_number[df_number['HasDetections']==0]
df_os_with_0_malware.shape
```

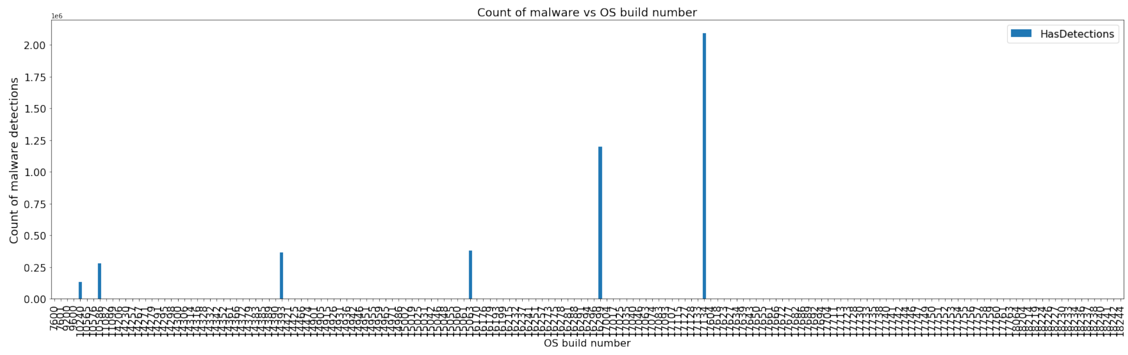
[21]: (40, 2)

Observations

Out of 165 OS build numbers, 40 build numbers do not have any malware detections

The above dataframe gives us the count of malware detections against each OS build number. The minimum and maximum number of detections is 0 and 2092768. Since this is a wide range of numbers I have plotted both raw counts and percenatges of count.

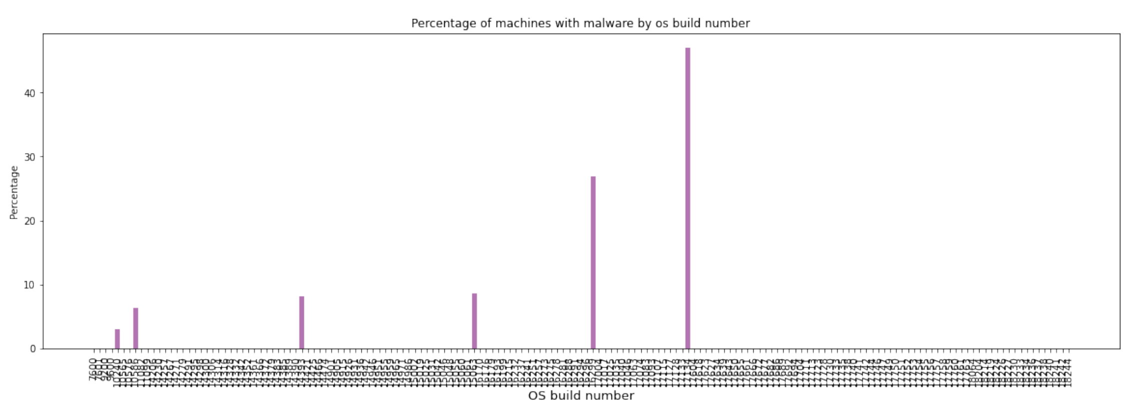
[22]:
```
df_number.plot.
 ↪bar(x='Census_OSBuildNumber',y='HasDetections',figsize=(30,8),fontsize=16)

plt.title('Count of malware vs OS build number',fontsize=18)
plt.xlabel('OS build number',fontsize=16)
plt.ylabel('Count of malware detections',fontsize=18)
plt.legend(loc=1, prop={'size': 16})
plt.show()
```



The above bar graph is very sparse because some os build versions have a very high number of malware detections while others have a relatively low number of these detections. Therefore, in the following graphs I have plotted the percentages and the logarithms of these frequency values.

[23]:
```python
column_sum = df_number['HasDetections'].sum()
df_number['HasDetections_percent'] = df_number['HasDetections'].div(column_sum).mul(100)
```

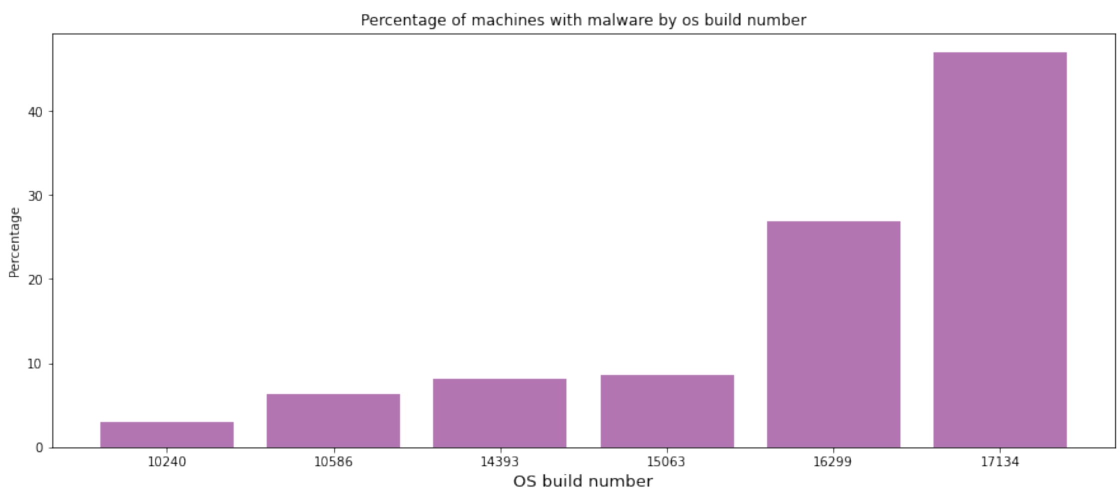
[24]:
```python
fig = plt.gcf()
fig.set_size_inches(20,6, forward = False)

bars = df_number['Census_OSBuildNumber']
x_pos = np.arange(len(bars))

ax = plt.bar(x_pos,df_number['HasDetections_percent'], color = (0.5,0.1,0.5,0.6))

plt.title('Percentage of machines with malware by os build number')
plt.xlabel('OS build number',fontsize=13)
plt.ylabel('Percentage')

plt.xticks(x_pos,bars,rotation='vertical')

plt.show()
```



Observations 1. From the abve graph, we can infer that over 40% of the malwares were detected in OS build number 17134. 2. The next in OS build number with the highest malware detections in 16299. 3. Since over 60% of the malware detections are from two OS build numbers we can conclude that this might be an important factor in predicting malwares. It also could be that there are bugs in these versions which cause high malware percentages.

The below graph closely observes all OS build versions that have more than 2% of malwares.

[25]:
```
fig = plt.gcf()
fig.set_size_inches(15,6)

df_number = df_number[df_number['HasDetections_percent'] > 2]
bars = df_number['Census_OSBuildNumber']
x_pos = np.arange(len(bars))

ax = plt.bar(x_pos,df_number['HasDetections_percent'], color = (0.5,0.1,0.5,0.
↪6))

plt.title('Percentage of machines with malware by os build number')
plt.xlabel('OS build number',fontsize=13)
plt.ylabel('Percentage')

plt.xticks(x_pos,bars)

plt.show()
```



(3b) Next we observe the relation between OS build revision number and malware detection. To do so, I created another view of the original dataframe to have Census_OSBuildNumber, Census_OSBuildRevision and HasDetections.

[26]:
```
cols = ['Census_OSBuildRevision','HasDetections']
df_os_version_vs_malware = df[cols].copy()
```

[27]:
```
df_number = df_os_version_vs_malware.
↪groupby(df_os_version_vs_malware['Census_OSBuildRevision']).
↪agg({'HasDetections':'sum'}).reset_index()
df_number.head()
```

[27]:
	Census_OSBuildRevision	HasDetections
0	0	85650
1	1	55303
2	3	3588
3	4	239
4	5	3125

[28]:
```
df_number.shape
```

[28]: (285, 2)

[29]:
```
print("Min:  ", df_number['HasDetections'].min())
print("Max:  ", df_number['HasDetections'].max())
```

```
Min:   0
Max:   754503
```

[30]:
```
df_os_with_0_malware = df_number[df_number['HasDetections']==0]
df_os_with_0_malware.shape
```

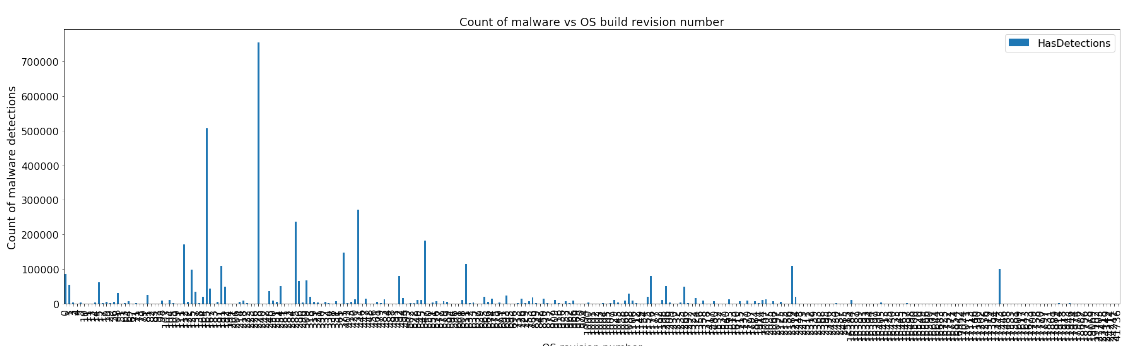
[30]: (30, 2)

Observaations

We can see that out of 285 build numbers 30 have 0 count for malwares.

The above dataframe gives us the count of malware detections against each OS build number. The minimum and maximum number of detections is 0 and 754503. Since this is a wide range of numbers I have plotted both raw counts and percenatges of count.

[31]:
```
df_number.plot.
 ↪bar(x='Census_OSBuildRevision',y='HasDetections',figsize=(30,8),fontsize=16)

plt.title('Count of malware vs OS build revision number',fontsize=18)
plt.xlabel('OS revision number',fontsize=16)
plt.ylabel('Count of malware detections',fontsize=18)
plt.legend(loc=1, prop={'size': 16})
plt.show()
```

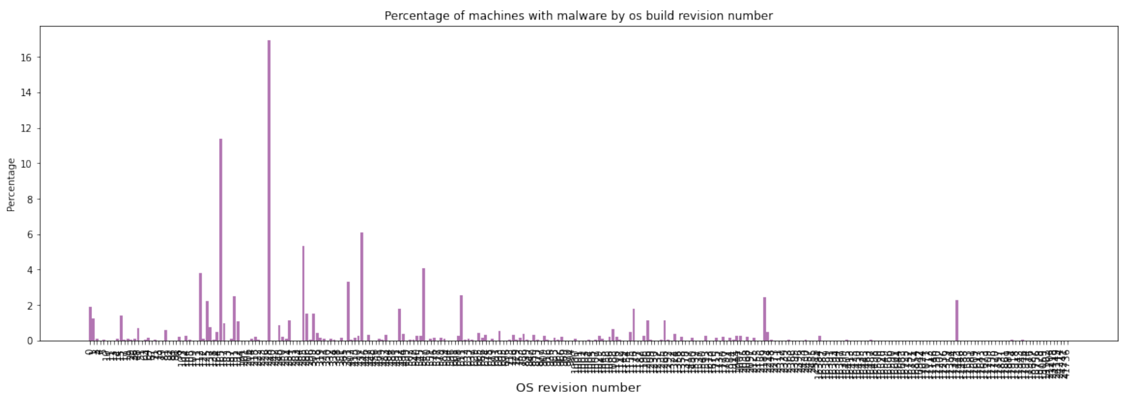


[32]:
```
column_sum = df_number['HasDetections'].sum()
df_number['HasDetections_percent'] = df_number['HasDetections'].div(column_sum).
    ↪mul(100)
```

[33]:
```
df_number['HasDetections_percent'].min()
```

[33]: 0.0

[34]:
```
fig = plt.gcf()
fig.set_size_inches(20,6, forward = False)

bars = df_number['Census_OSBuildRevision']
x_pos = np.arange(len(bars))

ax = plt.bar(x_pos,df_number['HasDetections_percent'], color = (0.5,0.1,0.5,0.
    ↪6))

plt.title('Percentage of machines with malware by os build revision number')
plt.xlabel('OS revision number',fontsize=13)
plt.ylabel('Percentage')

plt.xticks(x_pos,bars,rotation='vertical')

plt.show()
```

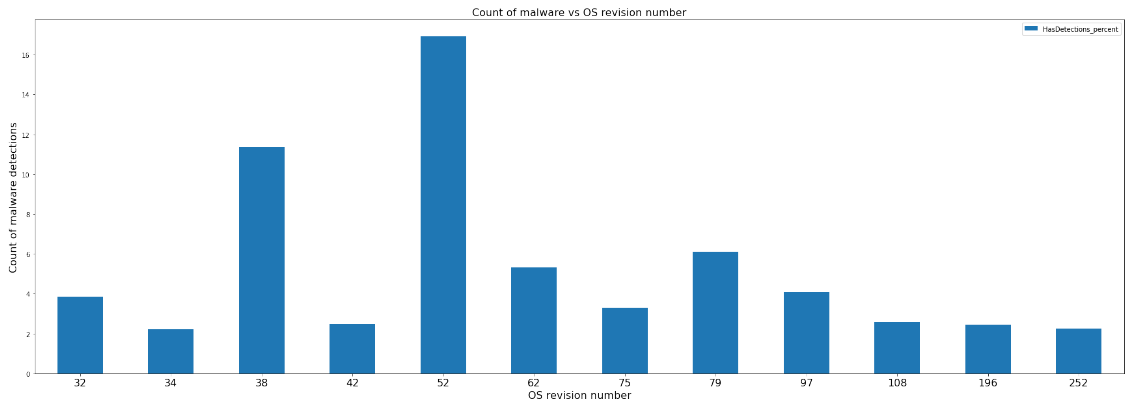


Since the above graph has too many entries we focus our attention to the ones that have a significant number of malware detection against them.

[35]:
```
import matplotlib.pyplot as plt

figure(figsize=(30, 6))
```

```
df_number = df_number[df_number['HasDetections_percent'] > 2]
df_number.plot.bar(y='HasDetections_percent',figsize=(30,10))

plt.title('Count of malware vs OS revision number',fontsize=16)
plt.xlabel('OS revision number',fontsize=16)
plt.ylabel('Count of malware detections',fontsize=16)
plt.xticks(rotation='horizontal',fontsize=16)
plt.show()
```

<Figure size 2160x432 with 0 Axes>



[36]:
```
df_number
```

[36]:

	Census_OSBuildRevision	HasDetections	HasDetections_percent
32	112	171022	3.835527
34	125	98279	2.204113
38	165	507414	11.379823
42	191	110462	2.477342
52	228	754503	16.921311
62	285	236785	5.310400
75	371	147289	3.303265
79	431	272301	6.106921
97	547	182365	4.089917
108	611	115024	2.579654
196	2189	108951	2.443455
252	17443	100759	2.259732

Observations

1. The above graph closely observes OS build revision numbers that have malware pecentages over 2%.
2. Revision number 52 has the highest percentage of malwares with roughly 17% of the total malwares.
3. Revision number 38 has the second highest percentages of malwares with 11.3% of the total

malwares.

0.6 Section 3: Effect of Number of AV Products Installed (Q4)

To study if the presence of AV products have any impact on malwares I use the column IsProtected. This column returns a boolean value/ null indicating the status of the AV products on the machine. True means at least one AV product is active with continuous update. False indicates the absence of any active AV products. Null means that no AV product is installed.

[37]:
```
cols = ['IsProtected','HasDetections']
```

[38]:
```
df_av = df[cols].copy()
df_av.head()
df_av.shape
```

[38]:
```
(8921483, 2)
```

Firstly, we plot the percentage of AV and Non-AV machines against malware percentages.

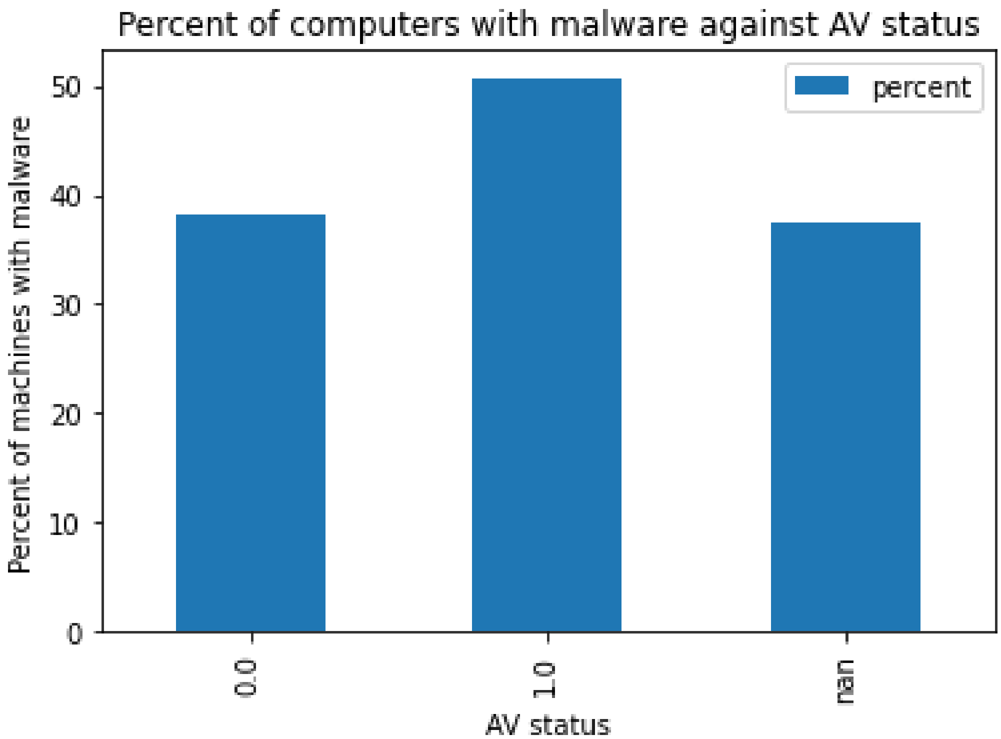
[39]:
```
df_av = df_av.groupby('IsProtected',dropna=False)['HasDetections'].
 ↪agg(['sum','count']).reset_index()
df_av['percent'] = (df_av['sum']/df_av['count'])*100
df_av.head()
```

[39]:

	IsProtected	sum	count	percent
0	0.0	184253	483157	38.135223
1	1.0	4261098	8402282	50.713580
2	NaN	13541	36044	37.567972

[40]:
```
figure(figsize=(8,6))

df_av.plot.bar(x='IsProtected',y='percent')

plt.title('Percent of computers with malware against AV status')
plt.xlabel('AV status')
plt.ylabel('Percent of machines with malware')

plt.show()
```

```
<Figure size 576x432 with 0 Axes>
```



Observations

The above graph displays quite counter intuitive data. We can see that the products with 1 AV status, which means that these machines have at least one active AV product have the highest percentage of malwares. Ths value is almost 50%. Whereas, machines with either no active product or no captured AV product have roughly only 37% machines with malwares. A possible explanation for this data can be that SpyNet could not record some third party AV products and these machines are currently falling under 'nan' label.

Next, we study the impact of number of AV products installed on the malware detection. For this study I have used the column AVProductsInstalled.

[41]:
```
av_cols = ['AVProductsInstalled','HasDetections']
```

[42]:
```
df_5 = df[av_cols].copy()
df_5.dropna()
df_5.head()
```

[42]:

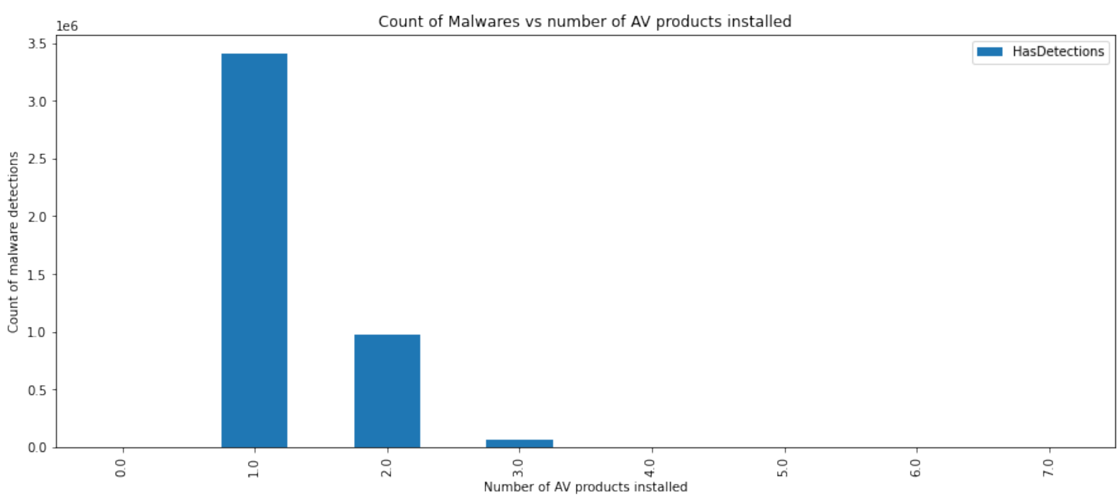
	AVProductsInstalled	HasDetections
0	1.0	0
1	1.0	0
2	1.0	0
3	1.0	1
4	1.0	1

[43]:
```
df_5 = df_5.groupby(['AVProductsInstalled']).sum()
df_5
```

[43]:
```
                     HasDetections
AVProductsInstalled
0.0                              0
1.0                        3406078
2.0                         975996
3.0                          60682
4.0                           2371
5.0                            125
6.0                              6
7.0                              1
```

[44]:
```
figure(figsize=(15, 6))

df_5.plot.bar(y='HasDetections',figsize=(15, 6))

plt.title('Count of Malwares vs number of AV products installed')
plt.xlabel('Number of AV products installed')
plt.ylabel('Count of malware detections')

plt.show()
```

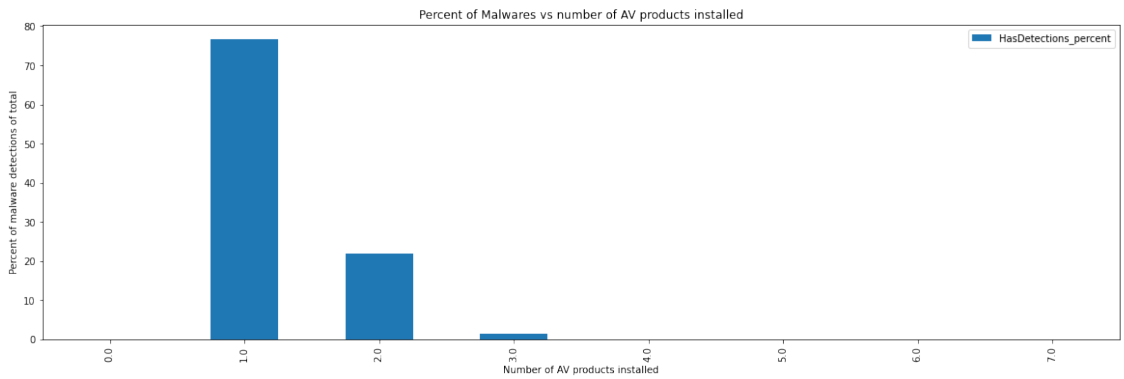
```
<Figure size 1080x432 with 0 Axes>
```



[45]:
```
column_sum = df_5['HasDetections'].sum()
df_5['HasDetections_percent'] = df_5['HasDetections'].div(column_sum).mul(100)
```

[46]:
```
figure(figsize=(20, 6))

df_5.plot.bar(y='HasDetections_percent',figsize=(20, 6))

plt.title('Percent of Malwares vs number of AV products installed')
plt.xlabel('Number of AV products installed')
plt.ylabel('Percent of malware detections of total')

plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```



Observations 1. From the above graph we can infer that AV products do actually have an impact on reducing the malwares. As we move along the x-axis i.e the number of AV products installed increases and we see a decline in the number of malwares detected. For the products with 0 AV products number of detections are 0. This can probably be explained by an anomaly, maybe these computers don't connect to Internet often.

0.7 Section 4: Interesting findings (Q5)

[47]:
```
df = pd.read_csv('train.csv')
df.head()
```

```
/Users/gauribaraskar/opt/anaconda3/lib/python3.8/site-
packages/IPython/core/interactiveshell.py:3165: DtypeWarning: Columns (28) have
mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

[47]:

	MachineIdentifier	ProductName	EngineVersion	\
0	0000028988387b115f69f31a3bf04f09	win8defender	1.1.15100.1	
1	000007535c3f730efa9ea0b7ef1bd645	win8defender	1.1.14600.4	
2	000007905a28d863f6d0d597892cd692	win8defender	1.1.15100.1	
3	00000b11598a75ea8ba1beea8459149f	win8defender	1.1.15100.1	
4	000014a5f00daa18e76b81417eeb99fc	win8defender	1.1.15100.1	

```
        AppVersion   AvSigVersion  IsBeta  RtpStateBitfield  IsSxsPassiveMode  \
0  4.18.1807.18075  1.273.1735.0       0               7.0                 0
1     4.13.17134.1    1.263.48.0       0               7.0                 0
2  4.18.1807.18075  1.273.1341.0       0               7.0                 0
3  4.18.1807.18075  1.273.1527.0       0               7.0                 0
4  4.18.1807.18075  1.273.1379.0       0               7.0                 0

   DefaultBrowsersIdentifier  AVProductStatesIdentifier  …  \
0                        NaN                    53447.0  …
1                        NaN                    53447.0  …
2                        NaN                    53447.0  …
3                        NaN                    53447.0  …
4                        NaN                    53447.0  …

   Census_FirmwareVersionIdentifier  Census_IsSecureBootEnabled  \
0                           36144.0                           0
1                           57858.0                           0
2                           52682.0                           0
3                           20050.0                           0
4                           19844.0                           0

   Census_IsWIMBootEnabled  Census_IsVirtualDevice  Census_IsTouchEnabled  \
0                      NaN                     0.0                      0
1                      NaN                     0.0                      0
2                      NaN                     0.0                      0
3                      NaN                     0.0                      0
4                      0.0                     0.0                      0

   Census_IsPenCapable  Census_IsAlwaysOnAlwaysConnectedCapable  Wdft_IsGamer  \
0                    0                                      0.0           0.0
1                    0                                      0.0           0.0
2                    0                                      0.0           0.0
3                    0                                      0.0           0.0
4                    0                                      0.0           0.0

  Wdft_RegionIdentifier HasDetections
0                  10.0             0
1                   8.0             0
2                   3.0             0
3                   3.0             1
4                   1.0             1

[5 rows x 83 columns]
```

1. Firstly, I want to study the distribution of malware affected computers across different countries.

[48]:
```
cols = ['CountryIdentifier','HasDetections']
```

[49]:
```
df_country = df[cols].copy()
```

[50]:
```
df_country = df_country.groupby(['CountryIdentifier'])['HasDetections'].
 ↪agg(['sum','count']).reset_index()
df_country['percent'] = df_country['sum']/df_country['count']
df_country.head()
```

[50]:

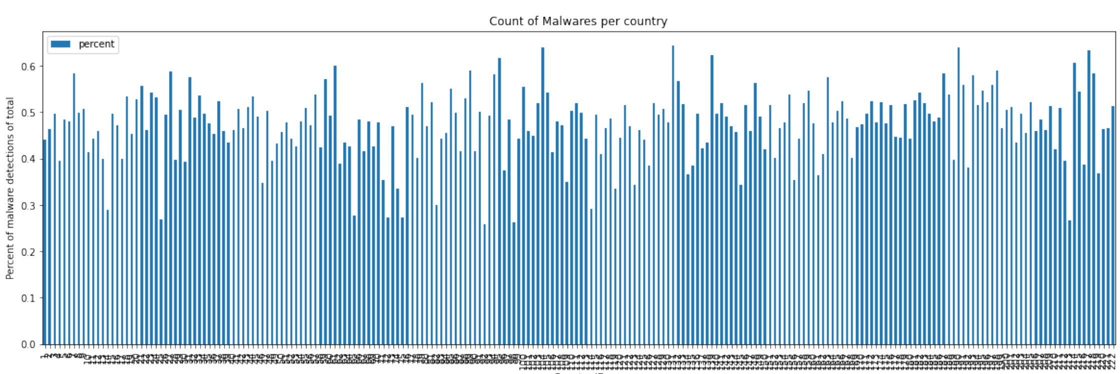
	CountryIdentifier	sum	count	percent
0	1	945	2141	0.441383
1	2	30708	66243	0.463566
2	3	2345	4722	0.496612
3	4	874	2210	0.395475
4	5	222	459	0.483660

[51]:
```
print("Mean :",df_country['percent'].mean())
print("Min :",df_country['percent'].min())
print("Max :",df_country['percent'].max())
print("Standard deviation :",df_country['percent'].std())
print("Variance :",df_country['percent'].var())
```

```
Mean : 0.47268067375376926
Min : 0.25883069427527405
Max : 0.6438531815996154
Standard deviation : 0.07524026969137118
Variance : 0.005661098183230268 5
```

[52]:
```
figure(figsize=(20, 6))

df_country.plot.bar(x='CountryIdentifier',y='percent',figsize=(20, 6))

plt.title('Count of Malwares per country')
plt.xlabel('Country ID')
plt.ylabel('Percent of malware detections of total')
plt.xticks(rotation='vertical')
plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```



Observations

The standard deviation and variance of the percentage of malware affected machines is almost the same for all countries. The above graph shows that country of a machine does not tell us much about whether it will be affected by malware. From whatever country, every machine has roughly the same percentage of getting affected by malware. The mean probability for getting affected by malware is roughly 47%.

(2) Second, we can study the malwares by products.

[53]:
```
cols = ['ProductName','HasDetections']
```

[54]:
```
df_product = df[cols].copy()
```

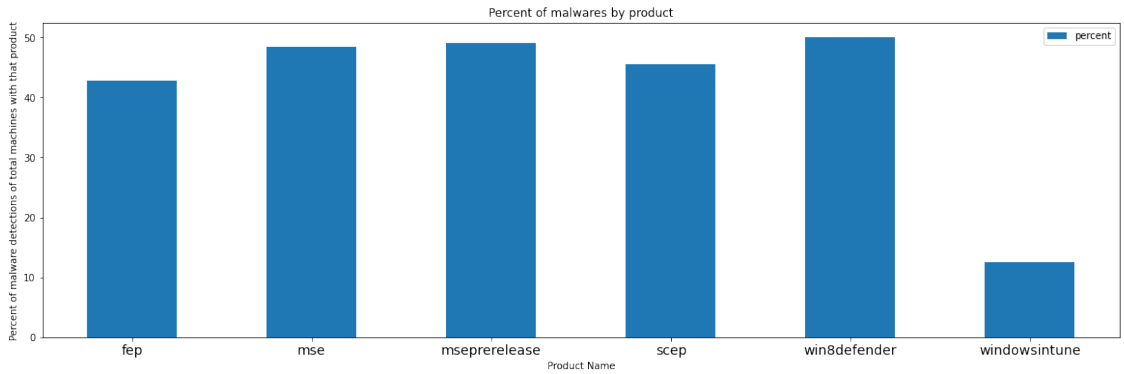
[55]:
```
df_product = df_product.groupby(['ProductName'])['HasDetections'].
 ↪agg(['sum','count']).reset_index()
df_product['percent'] = (df_product['sum']/df_product['count'])*100
df_product.head()
```

[55]:

	ProductName	sum	count	percent
0	fep	3	7	42.857143
1	mse	45961	94873	48.444763
2	mseprerelease	26	53	49.056604
3	scep	10	22	45.454545
4	win8defender	4412891	8826520	49.995819

[56]:
```
figure(figsize=(20, 6))

df_product.plot.bar(x='ProductName',y='percent',figsize=(20, 6))

plt.title('Percent of malwares by product')
plt.xlabel('Product Name')
plt.ylabel('Percent of malware detections of total machines with that product')
plt.xticks(rotation='horizontal',fontsize=14)
plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```



Observations 1. From the above graph we can see that except windowsintune all products have approximately the same percentage of malware affected machines. But windowsintune has significantly less value for malware affected machines.

(3) We can study the correlation of boolean variables like Firewall, SMode,Census_IsSecureBootEnabled.

[57]:
```
cols = ['Firewall','SMode','Census_IsSecureBootEnabled','HasDetections']
```

[58]:
```
df_new = df[cols].copy()
df_new.head()
```

[58]:

	IsProtected	Firewall	SMode	Census_IsSecureBootEnabled	HasDetections
0	1.0	1.0	0.0	0	0
1	1.0	1.0	0.0	0	0
2	1.0	1.0	0.0	0	0
3	1.0	1.0	0.0	0	1
4	1.0	1.0	0.0	0	1

[59]:
```
l = ['Firewall','HasDetections']
df_intermediate = df[l].copy()
df_intermediate = df_intermediate.
↪groupby(['Firewall'],dropna=False)['HasDetections'].agg(['sum','count']).
↪reset_index()
df_intermediate['percent'] = df_intermediate['sum']/df_intermediate['count']
df_intermediate.head()
```

[59]:

	Firewall	sum	count	percent
0	0.0	92590	189119	0.489586
1	1.0	4321114	8641014	0.500070
2	NaN	45188	91350	0.494669

Observations 1. From the above table we can say that almost the same percent of machines get affected by malware regardless of their Firewall status.So we can say that this column might not be very useful in predicting malware.

[60]:
```python
l = ['SMode','HasDetections']
df_intermediate = df[l].copy()
df_intermediate = df_intermediate.
 ↪groupby(['SMode'],dropna=False)['HasDetections'].agg(['sum','count']).
 ↪reset_index()
df_intermediate['percent'] = df_intermediate['sum']/df_intermediate['count']
df_intermediate.head()
```

[60]:

	SMode	sum	count	percent
0	0.0	4234781	8379843	0.505353
1	1.0	650	3881	0.167483
2	NaN	223461	537759	0.415541

Observations 1. From the above table we can say that percent of machines with SMode status on have significantly lesser malware detection. This value is less than half of those machines that do not have SMode.Hence, this column can be useful in predicting malware.

[61]:
```python
l = ['Census_IsSecureBootEnabled','HasDetections']
df_intermediate = df[l].copy()
df_intermediate = df_intermediate.
 ↪groupby(['Census_IsSecureBootEnabled'],dropna=False)['HasDetections'].
 ↪agg(['sum','count']).reset_index()
df_intermediate['percent'] = df_intermediate['sum']/df_intermediate['count']
df_intermediate.head()
```

[61]:

	Census_IsSecureBootEnabled	sum	count	percent
0	0	2295583	4585438	0.500625
1	1	2163309	4336045	0.498913

Observations 1. From the above table we can say that almost the same percent of machines get affected by malware regardless of their Secure boot enabled status. So we can say that this column might not be very useful in predicting malware.

[62]:
```python
l = ['Census_IsVirtualDevice','HasDetections']
df_intermediate = df[l].copy()
df_intermediate = df_intermediate.
 ↪groupby(['Census_IsVirtualDevice'],dropna=False)['HasDetections'].
 ↪agg(['sum','count']).reset_index()
df_intermediate['percent'] = df_intermediate['sum']/df_intermediate['count']
df_intermediate.head()
```

[62]:

	Census_IsVirtualDevice	sum	count	percent
0	0.0	4438599	8842840	0.501943
1	1.0	12172	62690	0.194162
2	NaN	8121	15953	0.509058

Observations 1. From the above table we can say that a virtual device is a lot less likely to get affected by malware than other categories.So we can say that this column might be useful in predicting malware.

[63]:
```
l = ['Census_IsAlwaysOnAlwaysConnectedCapable','HasDetections']
df_intermediate = df[l].copy()
df_intermediate = df_intermediate.
↪groupby(['Census_IsAlwaysOnAlwaysConnectedCapable'],dropna=False)['HasDetections'].
↪agg(['sum','count']).reset_index()
df_intermediate['percent'] = df_intermediate['sum']/df_intermediate['count']
df_intermediate.head()
```

[63]:

	Census_IsAlwaysOnAlwaysConnectedCapable	sum	count	percent
0	0.0	4232858	8341972	0.507417
1	1.0	189287	508168	0.372489
2	NaN	36747	71343	0.515075

Observations 1. From the above table we can say that a device that is always on always connected capable is a lot less likely to get affected by malware than other categories. So we can say that this column might be useful in predicting malware.

[64]:
```
l = ['Census_IsPortableOperatingSystem','HasDetections']
df_intermediate = df[l].copy()
df_intermediate = df_intermediate.
↪groupby(['Census_IsPortableOperatingSystem'],dropna=False)['HasDetections'].
↪agg(['sum','count']).reset_index()
df_intermediate['percent'] = df_intermediate['sum']/df_intermediate['count']
df_intermediate.head()
```

[64]:

	Census_IsPortableOperatingSystem	sum	count	percent
0	0	4456201	8916619	0.499764
1	1	2691	4864	0.553248

Observations 1. From the above table we can say that almost the same percent of machines get affected by malware regardless of their portable operating system value. So we can say that this column might not be very useful in predicting malware.

0.8 Section 5: Baseline modelling (Q6)

[65]:
```
use_cols_model = [
            "AVProductsInstalled",
            "Wdft_IsGamer",
            "Census_TotalPhysicalRAM",
            "Census_PrimaryDiskTotalCapacity",
            "Census_IsTouchEnabled",
            "Census_IsAlwaysOnAlwaysConnectedCapable",
            "Census_IsSecureBootEnabled",
            "Census_SystemVolumeTotalCapacity",
            "Census_PrimaryDiskTotalCapacity",
            "HasDetections"
           ]
```

[213]:
```
df_7 = pd.read_csv('train.csv',usecols=use_cols_model)
df_7 = df_7.dropna()

y = df_7.HasDetections
x = df_7.drop('HasDetections',axis = 1)


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

clf = LogisticRegression()
clf.fit(x_train,y_train)
```

[213]:
```
LogisticRegression()
```

[214]:
```
filename = 'logistic_regression_without_preprocessing.sav'
pickle.dump(clf, open(filename, 'wb'))
```

[215]:
```
y_pred = clf.predict(x_test)
```

[216]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy: ",accuracy)
print("Error rate: ",1-accuracy)
print("Accuracy percentage: ",100 * accuracy)
```

```
Accuracy:  0.5002897407056246
Error rate:  0.4997102592943754
Accuracy percentage:  50.02897407056246
```

Observations

The error rate simply means that the model fails to predict malware attack 49.9% of the times. In simpler terms, our baseline model can only correctly predict malware attack with 50% success rate. Our goal is to reduce this error rate so we can predict malware attacks with a higher accuracy.The error rate is approximately 0.50 which is quite high which means the model makes a mistake in predicting 50% of the malwares. I believe the error rate is such because "Garbage in,Garbage out". There was no preprocessing of columns which can deteriorate the performance of the model.

0.9 Section 6: Feature Cleaning and Additional models (Q7a & 7b)

[217]:
```
use_cols_model = [
            "SmartScreen",
            "Census_OSVersion",
            "AVProductsInstalled",
            "Wdft_IsGamer",
            "Census_TotalPhysicalRAM",
            "Census_PrimaryDiskTotalCapacity",
            "Census_IsTouchEnabled",
            "Census_IsAlwaysOnAlwaysConnectedCapable",
            "Census_IsSecureBootEnabled",
```

```
            "Census_SystemVolumeTotalCapacity",
            "HasDetections"
           ]
```

[218]:
```
df = pd.read_csv('train.csv',usecols=use_cols_model)
```

Step 1: Analysing missing data to find the percentage of na values in each column

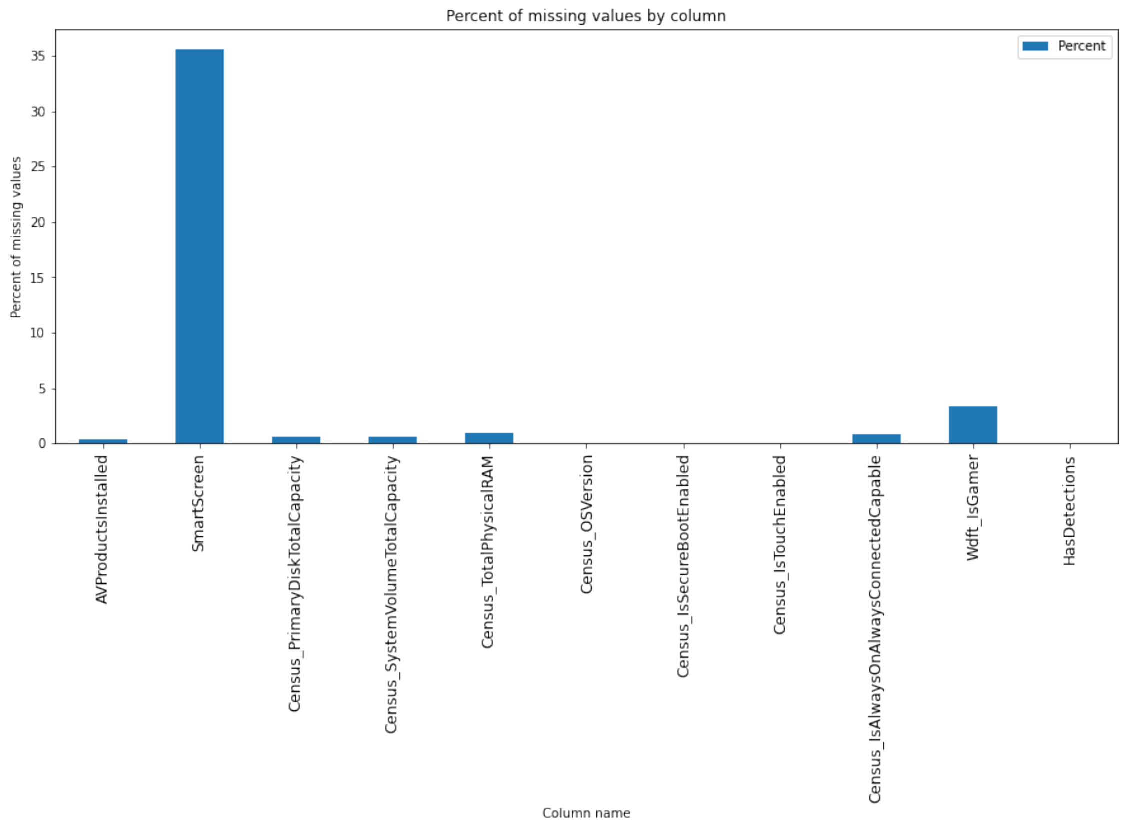
Nan values should either be imputed or processed because the model cannot process these values. In this step, I have first observed the percent of missing values in each of the selected columns.

1. If the number of missing values are high, it does not make sense to impute these values because we do not have a distribution to follow and this can alter our data easily.
2. If the variable is unique to that row, it should not be replaced.
3. Since, all the variables in the selected set of columns are unique to a machine I have not imputed any values.

[219]:
```
ser = df.isna().sum()/89214.83
```

[220]:
```
columns = pd.Series(ser.index)
values = pd.Series(ser.values)
```

[221]:
```
df_missing_data_cols = pd.DataFrame({'Column':columns, 'Percent':values})

df_missing_data_cols.plot.bar(x='Column', y='Percent', rot=0,figsize=(15,6))

plt.title('Percent of missing values by column')
plt.xlabel('Column name')
plt.ylabel('Percent of missing values')
plt.xticks(rotation='vertical',fontsize=12)
plt.show()
```



Observations

1. We can see from the above graph that the SmartScreen has 35% missing values. Replacing such a huge number of rows with imputed values can alter the data significantly, hence I will drop this column.

[222]:
```
df = df.drop(columns=['SmartScreen'])
```

Step 2: Analysing range of data

In this step, first I have observe the range of values of each columns which are of the type int and float. Large values can complicate the model and make the calculations slower. Also, when the range of columns is different, it might so happen that the influence of one variable is directly proportional to its value. To avoid such bias, we scale all the features.

[223]:
```
df_intermediate = df.select_dtypes(exclude=['object'])
```

[224]:
```
df_intermediate.max() - df_intermediate.min()
```

[224]:
```
AVProductsInstalled                        7.000000e+00
Census_PrimaryDiskTotalCapacity            8.160437e+12
Census_SystemVolumeTotalCapacity           4.768710e+07
Census_TotalPhysicalRAM                    1.572609e+06
```

```
Census_IsSecureBootEnabled                  1.000000e+00
Census_IsTouchEnabled                       1.000000e+00
Census_IsAlwaysOnAlwaysConnectedCapable     1.000000e+00
Wdft_IsGamer                                1.000000e+00
HasDetections                               1.000000e+00
dtype: float64
```

Observations

1. We can see that the ranges for Census_PrimaryDiskTotalCapacity, Census_SystemVolumeTotalCapacity, Census_TotalPhysicalRAM are high in the order of $10^{6,10}$7 and 10^12. We can normalise these values to be in range 0-1 for easier calculation.

Normalising values of Census_PrimaryDiskTotalCapacity, Census_SystemVolumeTotalCapacity, Census_TotalPhysicalRAM columns

[225]:
```python
from sklearn.preprocessing import MinMaxScaler

df['Census_PrimaryDiskTotalCapacity'] = MinMaxScaler().fit_transform(np.
 ↪array(df['Census_PrimaryDiskTotalCapacity']).reshape(-1,1))
df['Census_SystemVolumeTotalCapacity'] = MinMaxScaler().fit_transform(np.
 ↪array(df['Census_SystemVolumeTotalCapacity']).reshape(-1,1))
df['Census_TotalPhysicalRAM'] = MinMaxScaler().fit_transform(np.
 ↪array(df['Census_TotalPhysicalRAM']).reshape(-1,1))
```

[226]:
```python
df.head()
```

[226]:
```
   AVProductsInstalled  Census_PrimaryDiskTotalCapacity  \
0                  1.0                     5.844540e-08
1                  1.0                     5.844540e-08
2                  1.0                     1.402780e-08
3                  1.0                     2.922331e-08
4                  1.0                     5.844540e-08

   Census_SystemVolumeTotalCapacity  Census_TotalPhysicalRAM Census_OSVersion  \
0                          0.006279                 0.002442   10.0.17134.165
1                          0.002147                 0.002442     10.0.17134.1
2                          0.002389                 0.002442   10.0.17134.165
3                          0.004763                 0.002442   10.0.17134.228
4                          0.002137                 0.003745   10.0.17134.191

   Census_IsSecureBootEnabled  Census_IsTouchEnabled  \
0                           0                      0
1                           0                      0
2                           0                      0
3                           0                      0
4                           0                      0
```

```
   Census_IsAlwaysOnAlwaysConnectedCapable  Wdft_IsGamer  HasDetections
0                                      0.0           0.0              0
1                                      0.0           0.0              0
2                                      0.0           0.0              0
3                                      0.0           0.0              1
4                                      0.0           0.0              1
```

Step 3: Encoding category variables/unique variables

In this step, we preprocess the categorical and object columns. These columns sometimes might be numbers but should not be treated like normal numbers. I have used the technique of 'One Hot Encoding' which converts each categorical column into k other columns which are boolean in nature. For example, if a column A has values 'Phone' and 'TV'. One hot encoding will create two columns say, 'Is_phone' and 'IsTV'.

[227]:
```
df.dtypes
```

[227]:
```
AVProductsInstalled                           float64
Census_PrimaryDiskTotalCapacity               float64
Census_SystemVolumeTotalCapacity              float64
Census_TotalPhysicalRAM                       float64
Census_OSVersion                               object
Census_IsSecureBootEnabled                      int64
Census_IsTouchEnabled                           int64
Census_IsAlwaysOnAlwaysConnectedCapable       float64
Wdft_IsGamer                                  float64
HasDetections                                   int64
dtype: object
```

Although Census_OSVersion is an integer type variable we know that is a unique identifier for a OS build number. The values of this column do not hold meaning like normal integers where 2 < 3 and so on. Hence, I have decided to one hot encode the values. Since encoding wiht the entire value will lead to a large number of values, I have broken down the Census_OSVersion into four parts split by '.'. Each part becomes a feature which is individually hard coded.

[228]:
```
df_new = pd.DataFrame(df['Census_OSVersion'].astype(str).str.split('.', 3).
 ↪to_list(), columns=['OS_Version1',␣
 ↪'OS_Version2','OS_Version3','OS_Version4'])
```

[229]:
```
df = pd.concat([df, df_new], axis=1)
df = df.drop(columns=['Census_OSVersion'])
```

[230]:
```
obj_cols = [
    'OS_Version1',
    'OS_Version2',
    'OS_Version3',
    'OS_Version4'
]
prefix_obj_cols = [
```

```
    'OS1',
    'OS2',
    'OS3',
    'OS4'
]
```

[231]:
```
df = pd.get_dummies(df, sparse= True,columns=obj_cols, prefix=prefix_obj_cols)
```

[232]:
```
df.head()
```

[232]:
```
   AVProductsInstalled  Census_PrimaryDiskTotalCapacity  \
0                  1.0                     5.844540e-08
1                  1.0                     5.844540e-08
2                  1.0                     1.402780e-08
3                  1.0                     2.922331e-08
4                  1.0                     5.844540e-08

   Census_SystemVolumeTotalCapacity  Census_TotalPhysicalRAM  \
0                          0.006279                 0.002442
1                          0.002147                 0.002442
2                          0.002389                 0.002442
3                          0.004763                 0.002442
4                          0.002137                 0.003745

   Census_IsSecureBootEnabled  Census_IsTouchEnabled  \
0                           0                      0
1                           0                      0
2                           0                      0
3                           0                      0
4                           0                      0

   Census_IsAlwaysOnAlwaysConnectedCapable  Wdft_IsGamer  HasDetections  \
0                                      0.0           0.0              0
1                                      0.0           0.0              0
2                                      0.0           0.0              0
3                                      0.0           0.0              1
4                                      0.0           0.0              1

   OS1_10  …  OS4_91  OS4_916  OS4_936  OS4_953  OS4_962  OS4_966  OS4_969  \
0       1  …       0        0        0        0        0        0        0
1       1  …       0        0        0        0        0        0        0
2       1  …       0        0        0        0        0        0        0
3       1  …       0        0        0        0        0        0        0
4       1  …       0        0        0        0        0        0        0

   OS4_970  OS4_98  OS4_994
0        0       0        0
```

```
1         0         0          0
2         0         0          0
3         0         0          0
4         0         0          0

[5 rows x 465 columns]
```

[234]:
```
df.shape
```

[234]:
```
(8921483, 465)
```

Step 4: Drop infinity and nan values

Finally we drop all infinity and non defined values because the model cannot understand them.

[233]:
```
df.replace([np.inf, -np.inf], np.nan, inplace=True)
```

[236]:
```
df = df.
 ↪dropna(subset=['Wdft_IsGamer','AVProductsInstalled','Census_PrimaryDiskTotalCapacity','Cens
```

[238]:
```
df.shape
```

[238]:
```
(8490349, 462)
```

Step 5: Building a Logistic Regression model with clean features

[239]:
```
y = df.HasDetections
x = df.drop(['HasDetections'],axis = 1)
```

[240]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

[241]:
```
''' Logistic regression model'''
clf = LogisticRegression(verbose=1)
clf.fit(x_train,y_train)
```

```
/Users/gauribaraskar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:515: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/Users/gauribaraskar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: 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
```

```
  n_iter_i = _check_optimize_result(
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 44.8min finished
```

[241]:
```
LogisticRegression(verbose=1)
```

[243]:
```
filename = 'logistic_regression_with_preprocessing.sav'
pickle.dump(clf, open(filename, 'wb'))
```

[245]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
print("Error rate: ",error_rate)
accuracy_percentage = 100 * accuracy
print("Accuracy percentage: ",accuracy_percentage)
```

```
Error rate:  0.41593397209773453
Accuracy percentage:  58.40660279022655
```

Observations

Thus, we can observe that after preprocessing the accuracy increases by a large margin when we preprocess features. The next step would be to carefully select these features and make sure all selected features have enough impact on our model.

(7b) In this section I have built a model using Decision tree algorithm. The preprocessing involves dropping infinity and nan values, normalising columns with high ranges and encoding categorical variables.

[92]:
```
use_cols_model = [
            "ProductName",
            "AVProductsInstalled",
            "Wdft_IsGamer",
            "Census_TotalPhysicalRAM",
            "Census_PrimaryDiskTotalCapacity",
            "Census_IsTouchEnabled",
            "Census_IsAlwaysOnAlwaysConnectedCapable",
            "Census_IsSecureBootEnabled",
            "SMode",
            "Census_OSArchitecture",
            "Census_IsVirtualDevice",
            "HasDetections"
           ]
```

[93]:
```
df = pd.read_csv('train.csv',usecols=use_cols_model)
```

[94]:
```
df = df.dropna()
```

[95]:
```
from sklearn.preprocessing import MinMaxScaler

df['Census_PrimaryDiskTotalCapacity'] = MinMaxScaler().fit_transform(np.
 ↪array(df['Census_PrimaryDiskTotalCapacity']).reshape(-1,1))
```

```
#df['Census_SystemVolumeTotalCapacity'] = MinMaxScaler().fit_transform(np.
↪array(df['Census_SystemVolumeTotalCapacity']).reshape(-1,1))
df['Census_TotalPhysicalRAM'] = MinMaxScaler().fit_transform(np.
↪array(df['Census_TotalPhysicalRAM']).reshape(-1,1))
```

[96]:
```
obj_cols = ["ProductName","Census_OSArchitecture"]
prefix_obj_cols = ["Product","OSArchitecture"]
```

[97]:
```
df = pd.get_dummies(df, sparse= True,columns=obj_cols, prefix=prefix_obj_cols)
```

[98]:
```
y = df.HasDetections
x = df.drop(['HasDetections'],axis = 1)
```

[99]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

[100]:
```
from sklearn import tree

clf1 = tree.DecisionTreeClassifier()
clf1 = clf1.fit(x_train, y_train)
```

```
/Users/gauribaraskar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:515: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
```

[101]:
```
y_pred = clf1.predict(x_test)
```

```
/Users/gauribaraskar/opt/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:515: UserWarning: pandas.DataFrame with
sparse columns found.It will be converted to a dense numpy array.
  warnings.warn(
```

[103]:
```
filename = 'decision_tree_with_preprocessing.sav'
pickle.dump(clf1, open(filename, 'wb'))
```

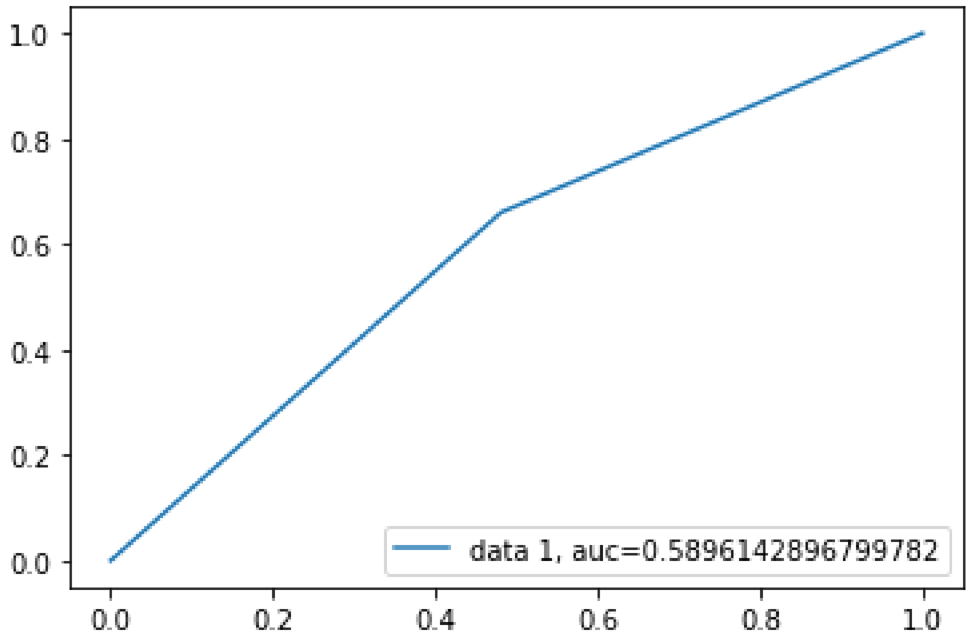
[102]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
print("Error rate: ",error_rate)
accuracy_percentage = 100 * accuracy
print("Accuracy percentage: ",accuracy_percentage)
```

```
Error rate:  0.40881716683545066
Accuracy percentage:  59.118283316454935
```

[130]:
```
loaded_model = pickle.load(open('decision_tree_with_preprocessing.sav', 'rb'))
y_pred = loaded_model.predict(x_test)
```

[132]:
```
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)
```

```python
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
print("AUC Score:", auc)
print("Error Rate:", 1-auc)
```




```
AUC Score: 0.5896142896799782
Error Rate: 0.4103857103200218
```

[122]:
```python
!pip3 install tabulate
```

```
Collecting tabulate
  Downloading tabulate-0.8.9-py3-none-any.whl (25 kB)
Installing collected packages: tabulate
Successfully installed tabulate-0.8.9
```

[131]:
```python
table = [["Model name","Accuracy %","Error rate %"],
         ["Model 0",50.02,49.97],
         ["Model 1",58.40,41.59],
         ["Model 2",58.96,41.03]]
```

[132]:
```python
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

```
<IPython.core.display.HTML object>
```

Explanations

1. From the above table we see a continuous improvement in accuarcy and reduction in error

rate.

2. Model 0 used non-preprocessed data and therefore has the least accuracy. In this model the features were not scaled. All the rows with nan values are dropped and not imputed which results in the loss of data.

3. Model 1 used cleaned features, therefore the model could train better. In this step, scaling of features was performed. This usually helps the model is training better because the sheer high range of a feature cannot implicitly mean that they are more important than others. In other words, all features are equally represented when the scales are same. The accuracy saw a very high jump of roughly 8%.

4. Model 2 used additional features than Model 1. This resulted only a marginal increase in accuracy.This could be because of the features that were added were not correlated stronly to the prediction.

0.10 Testing

[105]:
```
use_cols_model = [
              "MachineIdentifier",
              "ProductName",
              "AVProductsInstalled",
              "Wdft_IsGamer",
              "Census_TotalPhysicalRAM",
              "Census_PrimaryDiskTotalCapacity",
              "Census_IsTouchEnabled",
              "Census_IsAlwaysOnAlwaysConnectedCapable",
              "Census_IsSecureBootEnabled",
              "SMode",
              "Census_OSArchitecture",
              "Census_IsVirtualDevice"
             ]
```

[106]:
```
test_df = pd.read_csv('test.csv',usecols=use_cols_model)
```

[107]:
```
MachineIdentifier = test_df['MachineIdentifier']
```

[108]:
```
'''Preprocessing'''
test_df = test_df.drop(columns=['MachineIdentifier'])
test_df['Census_PrimaryDiskTotalCapacity'] = MinMaxScaler().fit_transform(np.
 ↪array(test_df['Census_PrimaryDiskTotalCapacity']).reshape(-1,1))
test_df['Census_TotalPhysicalRAM'] = MinMaxScaler().fit_transform(np.
 ↪array(test_df['Census_TotalPhysicalRAM']).reshape(-1,1))
test_df = pd.get_dummies(test_df, sparse= True,columns=obj_cols,␣
 ↪prefix=prefix_obj_cols)
test_df.replace([np.inf, -np.inf, np.nan], 0, inplace=True)
```

[109]:
```
y_pred = clf1.predict(test_df)
```

[110]:
```
MachineIdentifier = MachineIdentifier.to_frame()
MachineIdentifier['HasDetections'] = y_pred
MachineIdentifier = MachineIdentifier.reset_index(drop=True)
df_final = MachineIdentifier[['MachineIdentifier','HasDetections']]
df_final.to_csv('out.csv',index=False)
```

0.11 Section 7: Screenshots (Q8)

Public Score: 0.52175

Private Score:0.49956

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

Screenshot(s):

[120]:
```
from IPython.display import Image
Image("kaggle_submission.png")
```

[120]:

Submission and Description	Private Score	Public Score
out.csv 10 minutes ago by Gauri Baraskar Decision tree with more features.	0.49956	0.52175
out.csv an hour ago by Gauri Baraskar Decision tree model	0.50138	0.51758