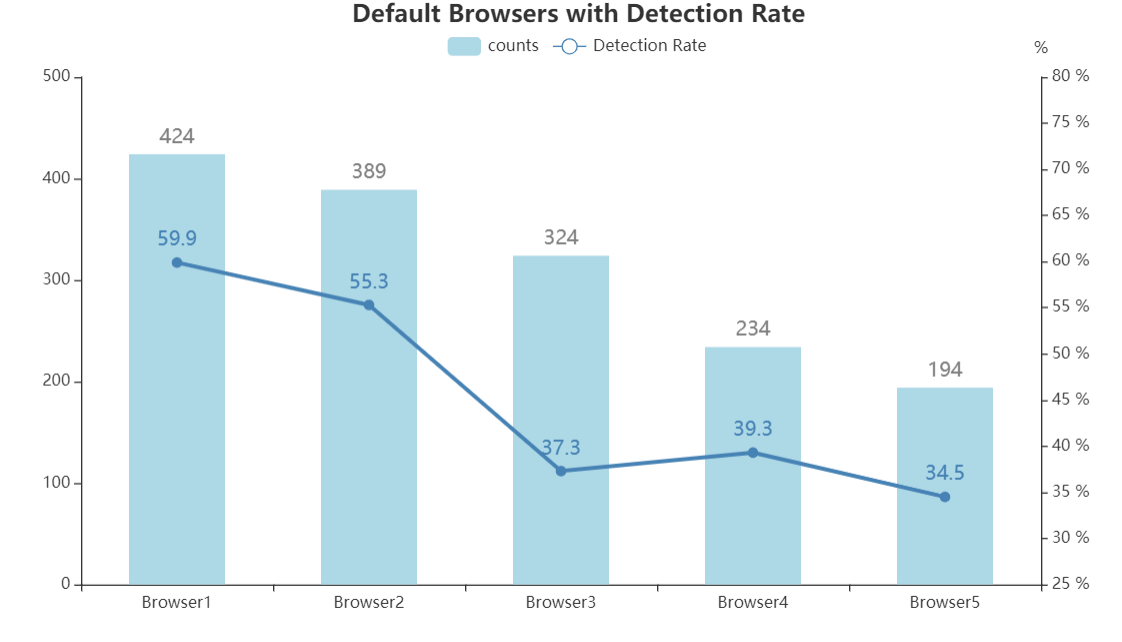
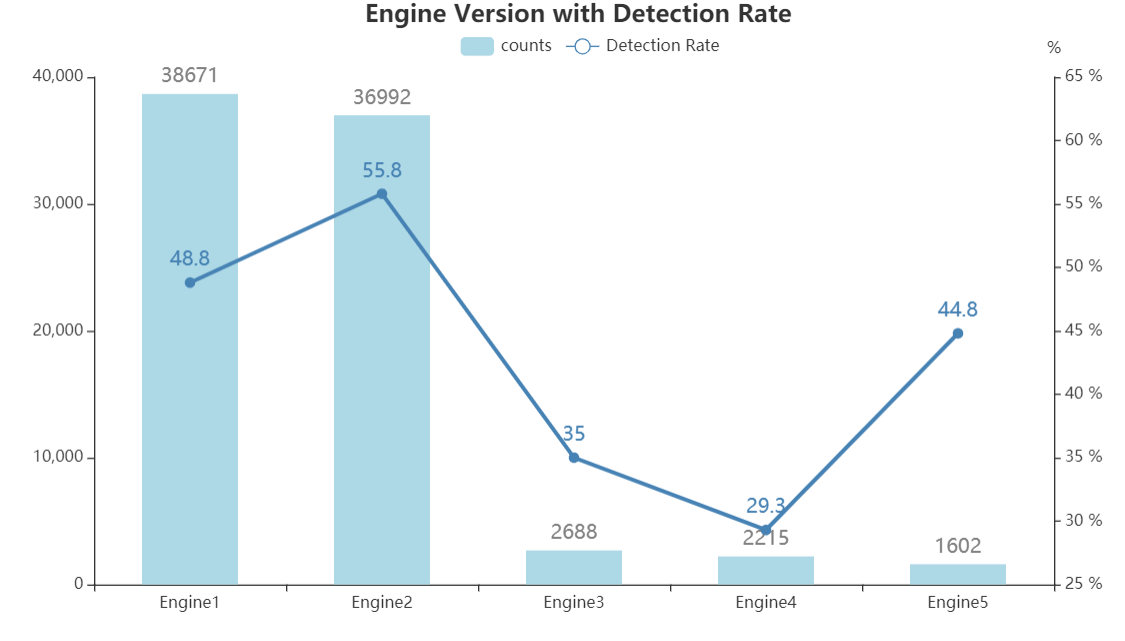
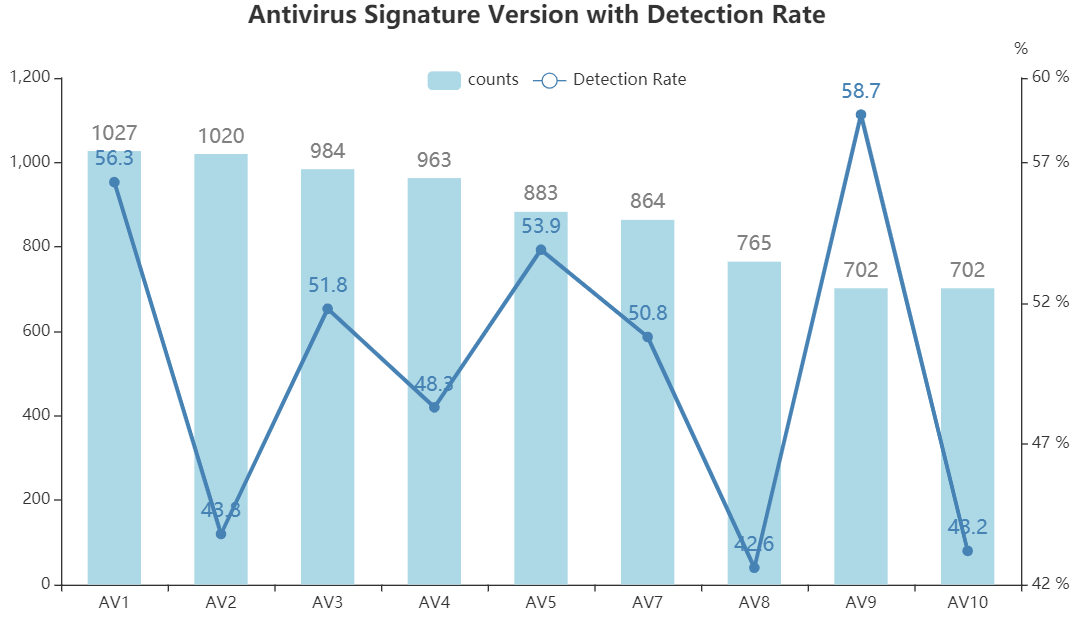
**Appendix**

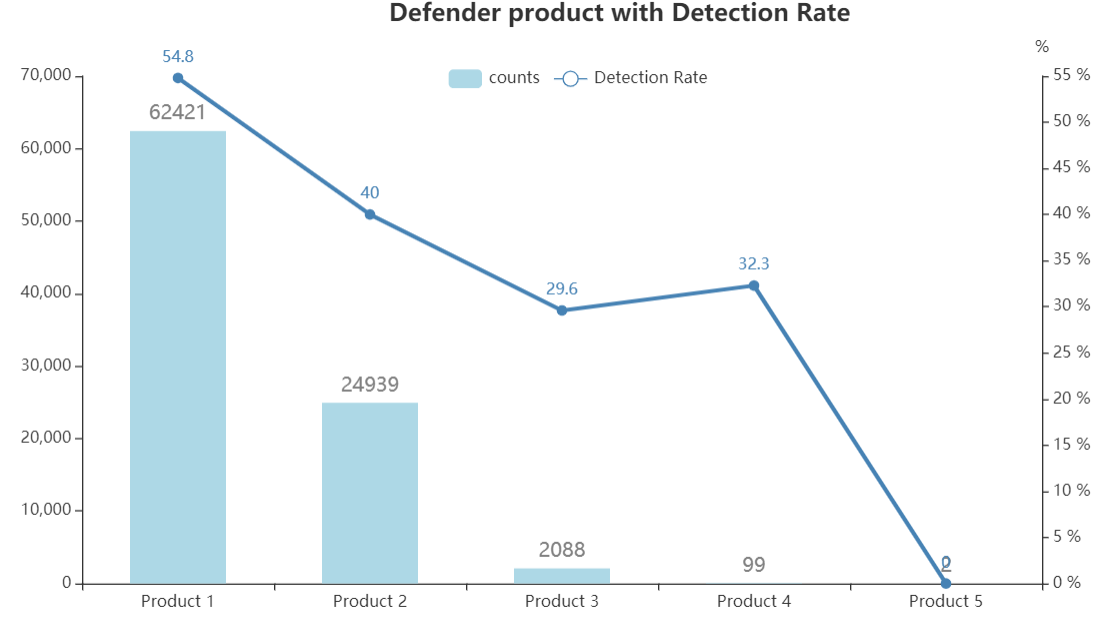
1. **Data Visualization**

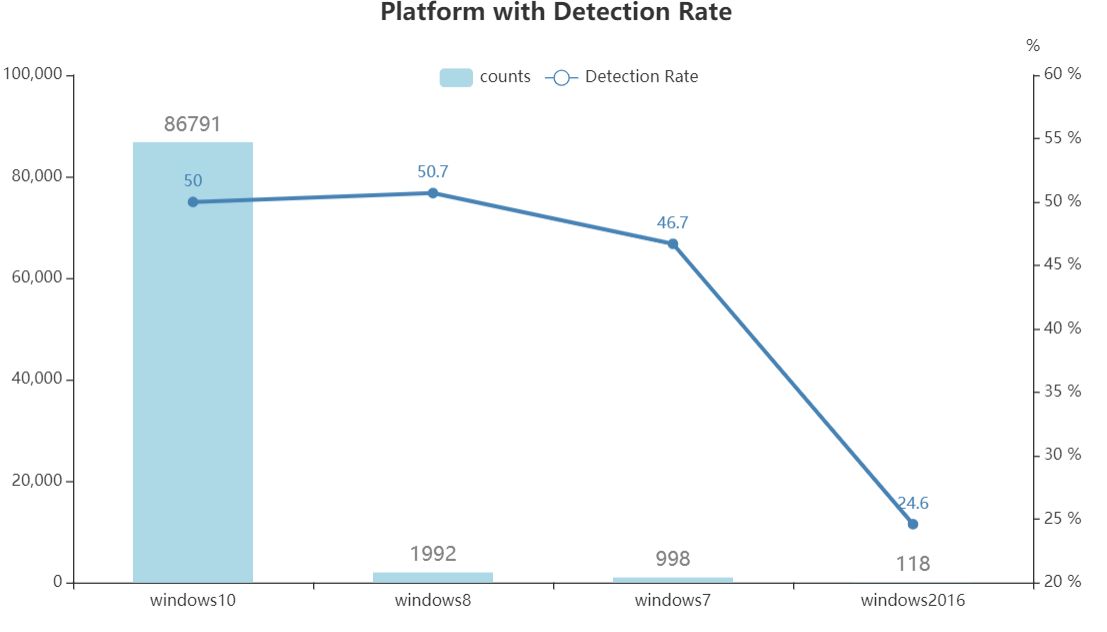
Correlations Between Predictors and Response

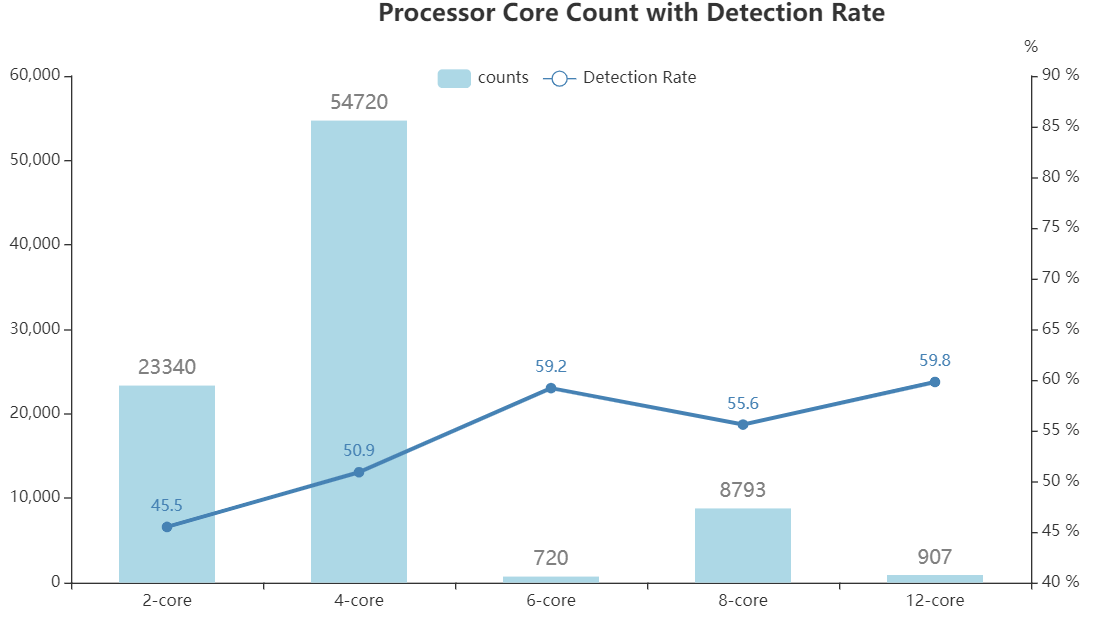


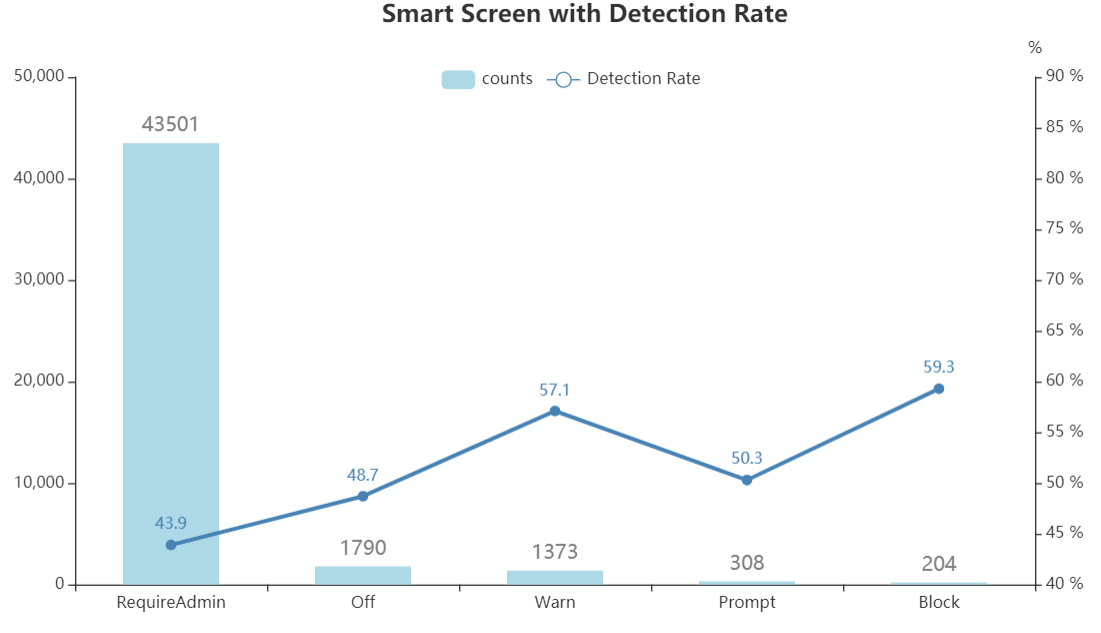


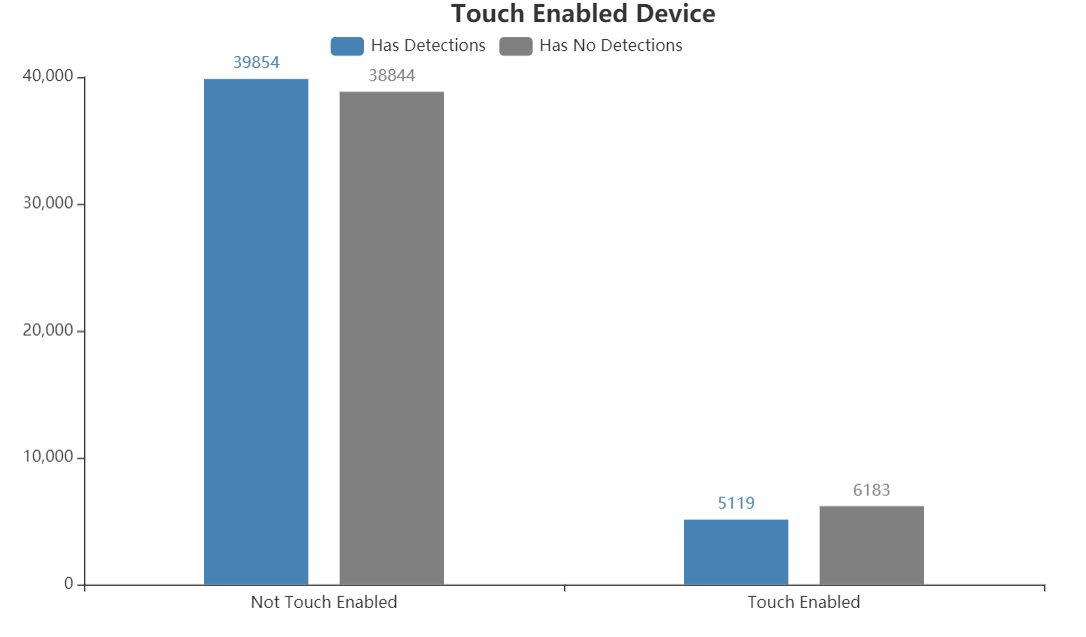


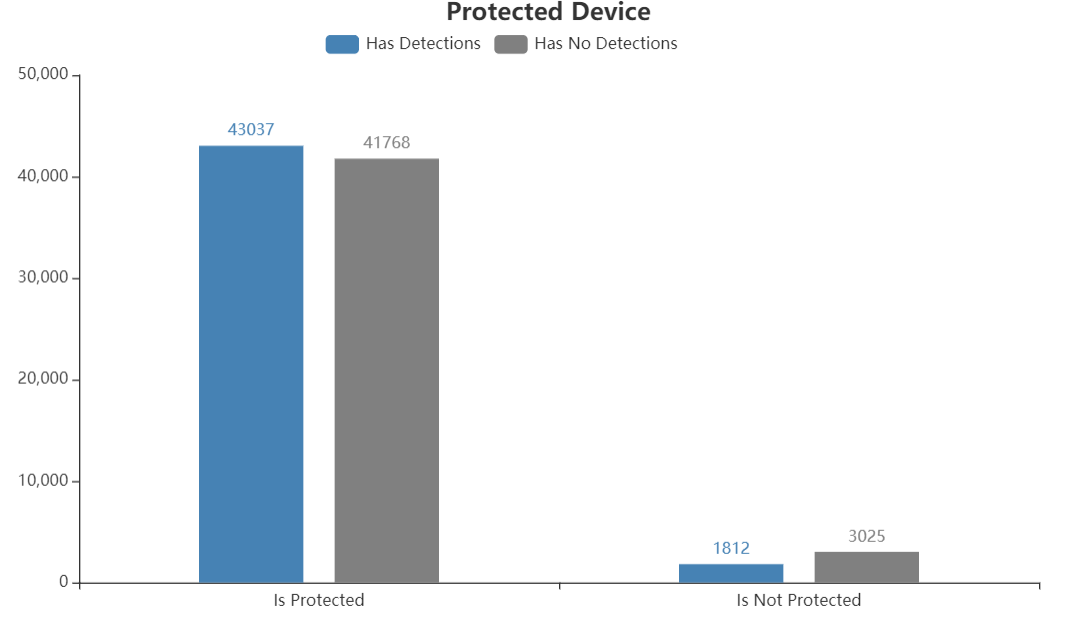


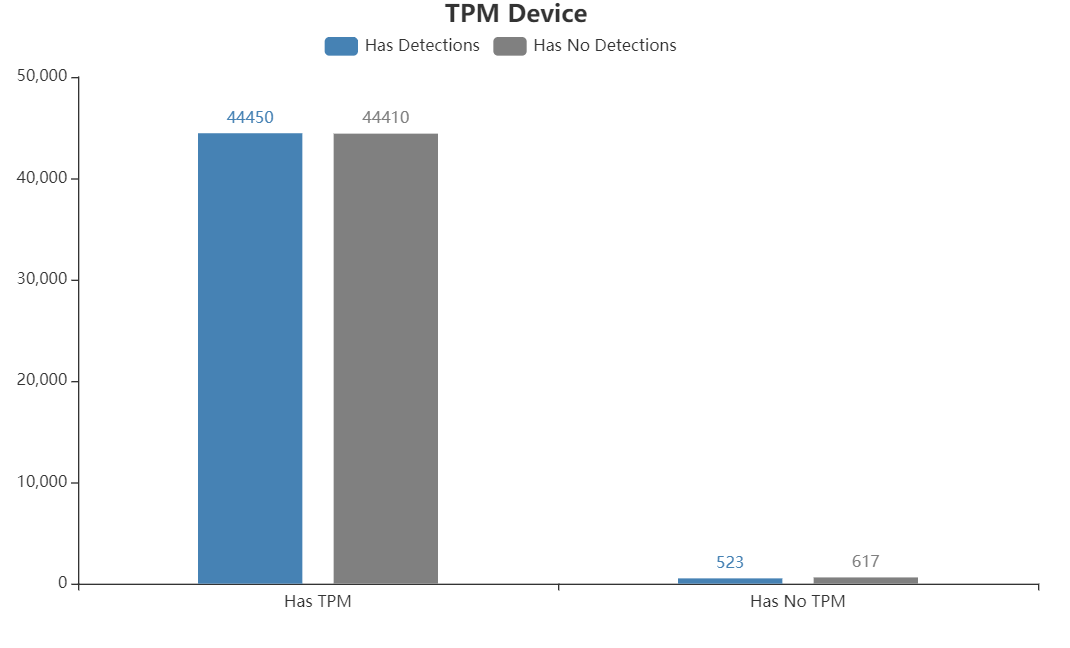


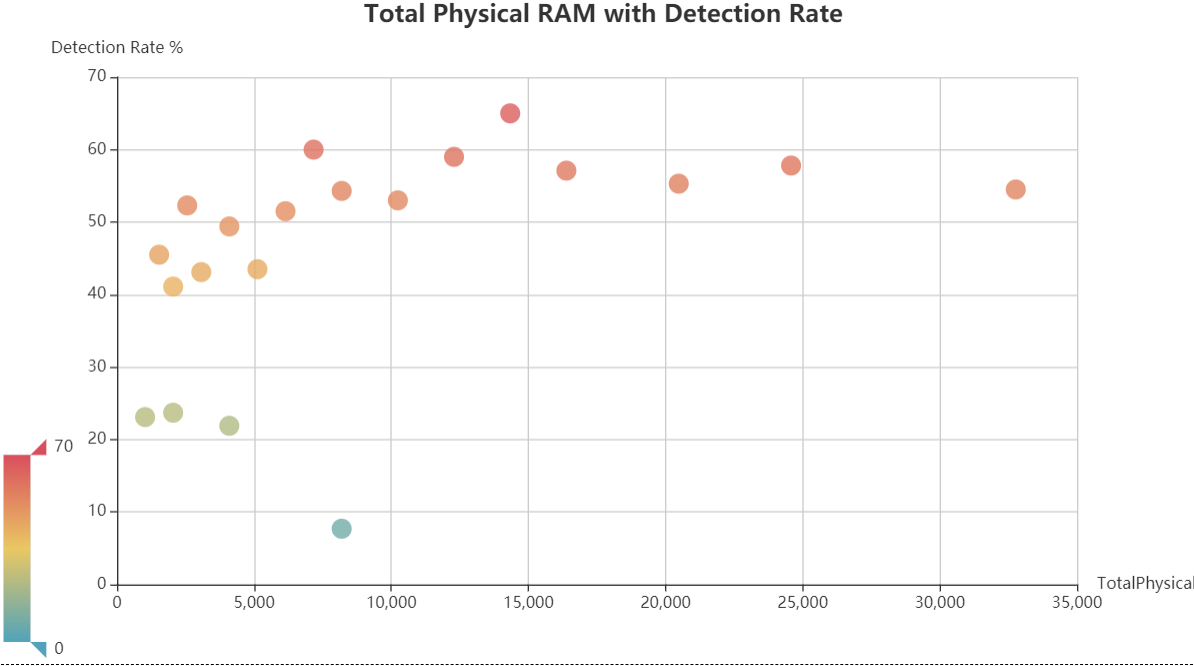


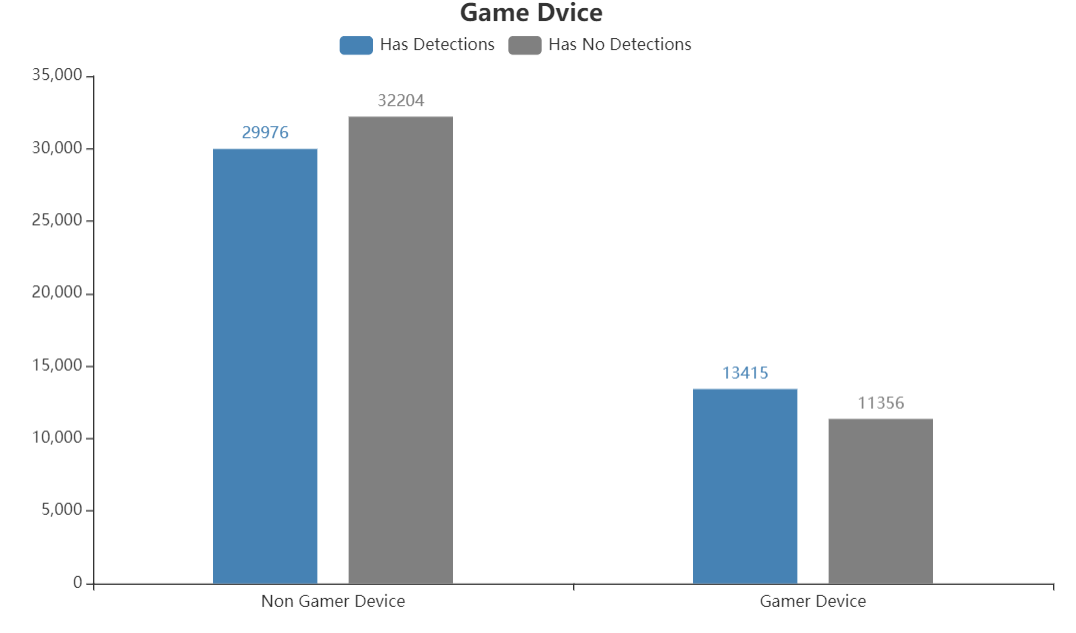












1. **Code(Python)**

def rename\_edition(x):

x = x.lower()

if 'core' in x:

return 'Core'

elif 'pro' in x:

return 'pro'

elif 'enterprise' in x:

return 'Enterprise'

elif 'server' in x:

return 'Server'

elif 'home' in x:

return 'Home'

elif 'education' in x:

return 'Education'

elif 'cloud' in x:

return 'Cloud'

else:

return x

#%% Feature Engineering

def fe(train):

print('Begin Feature Engineering...')

df=train.copy()

#Splitting everything out

df['OsBuildLab'] = df['OsBuildLab'].cat.add\_categories(['0.0.0.0.0-0'])

df['OsBuildLab'] = df['OsBuildLab'].fillna('0.0.0.0.0-0')

df['EngineVersion\_2'] = df['EngineVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['EngineVersion\_3'] = df['EngineVersion'].apply(lambda x: x.split('.')[3]).astype('category')

df['AppVersion\_1'] = df['AppVersion'].apply(lambda x: x.split('.')[1]).astype('category')

df['AppVersion\_2'] = df['AppVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['AppVersion\_3'] = df['AppVersion'].apply(lambda x: x.split('.')[3]).astype('category')

df['AvSigVersion\_0'] = df['AvSigVersion'].apply(lambda x: x.split('.')[0]).astype('category')

df['AvSigVersion\_1'] = df['AvSigVersion'].apply(lambda x: x.split('.')[1]).astype('category')

df['AvSigVersion\_2'] = df['AvSigVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['OsBuildLab\_0'] = df['OsBuildLab'].apply(lambda x: x.split('.')[0]).astype('category')

df['OsBuildLab\_1'] = df['OsBuildLab'].apply(lambda x: x.split('.')[1]).astype('category')

df['OsBuildLab\_2'] = df['OsBuildLab'].apply(lambda x: x.split('.')[2]).astype('category')

df['OsBuildLab\_3'] = df['OsBuildLab'].apply(lambda x: x.split('.')[3]).astype('category')

df['Census\_OSVersion\_0'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[0]).astype('category')

df['Census\_OSVersion\_1'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[1]).astype('category')

df['Census\_OSVersion\_2'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[2]).astype('category')

df['Census\_OSVersion\_3'] = df['Census\_OSVersion'].apply(lambda x: x.split('.')[3]).astype('category')

# https://www.kaggle.com/adityaecdrid/simple-feature-engineering-xd

df['primary\_drive\_c\_ratio'] = df['Census\_SystemVolumeTotalCapacity']/ df['Census\_PrimaryDiskTotalCapacity']

#df['non\_primary\_drive\_MB'] = df['Census\_PrimaryDiskTotalCapacity'] - df['Census\_SystemVolumeTotalCapacity']

df['aspect\_ratio'] = df['Census\_InternalPrimaryDisplayResolutionHorizontal']/ df['Census\_InternalPrimaryDisplayResolutionVertical']

#df['monitor\_dims'] = df['Census\_InternalPrimaryDisplayResolutionHorizontal'].astype(str) + '\*' + df['Census\_InternalPrimaryDisplayResolutionVertical'].astype('str')

#df['monitor\_dims'] = df['monitor\_dims'].astype('category')

df['dpi'] = ((df['Census\_InternalPrimaryDisplayResolutionHorizontal']\*\*2 + df['Census\_InternalPrimaryDisplayResolutionVertical']\*\*2)\*\*.5)/(df['Census\_InternalPrimaryDiagonalDisplaySizeInInches'])

#df['dpi\_square'] = df['dpi'] \*\* 2

df['MegaPixels'] = (df['Census\_InternalPrimaryDisplayResolutionHorizontal'] \* df['Census\_InternalPrimaryDisplayResolutionVertical'])/1e6

#df['Screen\_Area'] = (df['aspect\_ratio']\* (df['Census\_InternalPrimaryDiagonalDisplaySizeInInches']\*\*2))/(df['aspect\_ratio']\*\*2 + 1)

df['ram\_per\_processor'] = df['Census\_TotalPhysicalRAM']/ df['Census\_ProcessorCoreCount']

df['new\_num\_0'] = df['Census\_InternalPrimaryDiagonalDisplaySizeInInches'] / df['Census\_ProcessorCoreCount']

df['new\_num\_1'] = df['Census\_ProcessorCoreCount'] \* df['Census\_InternalPrimaryDiagonalDisplaySizeInInches']

df['Census\_IsFlightingInternal'] = df['Census\_IsFlightingInternal'].fillna(1)

df['Census\_ThresholdOptIn'] = df['Census\_ThresholdOptIn'].fillna(1)

df['Census\_IsWIMBootEnabled'] = df['Census\_IsWIMBootEnabled'].fillna(1)

df['Wdft\_IsGamer'] = df['Wdft\_IsGamer'].fillna(0)

#

df.loc[df['AVProductsInstalled'].isin([1, 2]) == False, 'AVProductsInstalled'] = 3

df.loc[df['OrganizationIdentifier'].isin([27, 18]) == False, 'OrganizationIdentifier'] = 48

df['OrganizationIdentifier'] = df['OrganizationIdentifier'].astype('category')

df['Census\_OSSkuName'] = df['Census\_OSSkuName'].astype(str)

df['Census\_OSSkuName'] = df['Census\_OSSkuName'].apply(rename\_edition)

df['Census\_OSSkuName'] = df['Census\_OSSkuName'].astype('category')

df.loc[df['Census\_ProcessorCoreCount'].isin([2, 4, 8, 12]) == False, 'Census\_ProcessorCoreCount'] = 1

top\_10 = df['Census\_TotalPhysicalRAM'].value\_counts(dropna=False, normalize=True).cumsum().index[:10]

df.loc[df['Census\_TotalPhysicalRAM'].isin(top\_10) == False, 'Census\_TotalPhysicalRAM'] = 1000

def group\_battery(x):

x = x.lower()

if 'li' in x:

return 1

else:

return 0

df['Census\_InternalBatteryType'] = df['Census\_InternalBatteryType'].apply(group\_battery)

df.loc[df['SmartScreen'].isnull(), 'SmartScreen'] = 'ExistsNotSet'

df.loc[df['SmartScreen'].isin(['RequireAdmin', 'ExistsNotSet', 'Off', 'Warn']) == False, 'SmartScreen'] = 'Prompt'

df['SmartScreen'] = df['SmartScreen'].cat.remove\_unused\_categories()

df.loc[df['Census\_PrimaryDiskTypeName'].isin(['HDD', 'SSD']) == False, 'Census\_PrimaryDiskTypeName'] = 'UNKNOWN'

df['Census\_PrimaryDiskTypeName'] = df['Census\_PrimaryDiskTypeName'].cat.remove\_unused\_categories()

df.loc[df['Census\_ProcessorManufacturerIdentifier'].isin([5.0, 1.0]) == False, 'Census\_ProcessorManufacturerIdentifier'] = 0.0

df['Census\_ProcessorManufacturerIdentifier'] = df['Census\_ProcessorManufacturerIdentifier'].astype('category')

df.loc[df['Census\_PowerPlatformRoleName'].isin(['Mobile', 'Desktop', 'Slate']) == False, 'Census\_PowerPlatformRoleName'] = 'UNKNOWN'

df['Census\_PowerPlatformRoleName'] = df['Census\_PowerPlatformRoleName'].cat.remove\_unused\_categories()

top\_cats = list(df['Census\_OSWUAutoUpdateOptionsName'].value\_counts().index[:3])

df.loc[df['Census\_OSWUAutoUpdateOptionsName'].isin(top\_cats) == False, 'Census\_OSWUAutoUpdateOptionsName'] = 'Off'

df['Census\_OSWUAutoUpdateOptionsName'] = df['Census\_OSWUAutoUpdateOptionsName'].cat.remove\_unused\_categories()

df.loc[df['Census\_GenuineStateName'] == 'UNKNOWN', 'Census\_GenuineStateName'] = 'OFFLINE'

df['Census\_GenuineStateName'] = df['Census\_GenuineStateName'].cat.remove\_unused\_categories()

df.loc[df['Census\_ActivationChannel'].isin(['Retail', 'OEM:DM']) == False, 'Census\_ActivationChannel'] = 'Volume:GVLK'

df['Census\_ActivationChannel'] = df['Census\_ActivationChannel'].cat.remove\_unused\_categories()

# Drop for memory

df.drop('Census\_InternalPrimaryDisplayResolutionHorizontal', axis=1)

df.drop('Census\_OSUILocaleIdentifier', axis=1)

df.drop('AppVersion\_3',axis=1)

remove\_cols = ['PuaMode', 'Census\_ProcessorClass',

'Census\_IsWIMBootEnabled', 'IsBeta',

'Census\_IsFlightsDisabled', 'Census\_IsFlightingInternal',

'AutoSampleOptIn', 'Census\_ThresholdOptIn', 'SMode',

'Census\_IsPortableOperatingSystem', 'Census\_DeviceFamily',

'UacLuaenable', 'Census\_IsVirtualDevice', 'Platform',

'Census\_OSSkuName', 'Census\_OSInstallLanguageIdentifier', 'Processor']

df.drop(remove\_cols, axis=1, inplace=True)

gc.collect()

print('Finish Feature Engineering','Total Column:',df.shape[1])

return df

#%%

def Combiner(df, col, val\_lower, otherval):

top\_cat = list(df[col].value\_counts(dropna=False).index[:])

df\_split=df.loc[df[col].isin(top\_cat)]

a = df\_split[col].value\_counts()

m = df\_split[col].isin(a.index[a<val\_lower])

m = list(m)

df.loc[m, col] = otherval

return df

#%%

def dateFE(df, col,datedict):

print('Begin add time stamp...')

#Date information

label='Date'+str(col)

df['Date'+str(col)] = df[col].map(datedict)

add\_datepart(df, 'Date'+str(col), drop=False)

datetime = df['Date'+str(col)].max()

print(datetime)

df['DaysElapsed'+col]= ((datetime)- df['Date'+str(col)]).dt.days

df=df.drop(label,axis =1)

print('Variable is '+str(label))

print(f'Final Columns: {df.shape[1]}, Finish adding time stamp')

return df

#%%

def add\_datepart(df, fldname, drop=True, time=False):

"Helper function that adds columns relevant to a date."

fld = df[fldname]

fld\_dtype = fld.dtype

if isinstance(fld\_dtype, pd.core.dtypes.dtypes.DatetimeTZDtype):

fld\_dtype = np.datetime64

if not np.issubdtype(fld\_dtype, np.datetime64):

df[fldname] = fld = pd.to\_datetime(fld, infer\_datetime\_format=True)

targ\_pre = re.sub('[Dd]ate$', '', fldname)

attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek']

#, 'Dayofyear', 'Is\_month\_end'], 'Is\_month\_start', 'Is\_quarter\_end', 'Is\_quarter\_start', 'Is\_year\_end', 'Is\_year\_start']

if time: attr = attr + ['Hour', 'Minute', 'Second']

for n in attr: df[targ\_pre + n] = getattr(fld.dt, n.lower())

#df[targ\_pre + 'Elapsed'] = fld.astype(np.int64) // 10 \*\* 9

if drop: df.drop(fldname, axis=1, inplace=True)

#%%

#Decrease number of unique values

def DecreaseUniqueValues(train):

print("Decreasing Unique Values for 'AvSigVersion','AVProductStatesIdentifier','CityIdentifier','Census\_OEMNameIdentifier','Census\_InternalBatteryNumberOfCharges','Census\_FirmwareVersionIdentifier'")

print("Original Unique Values:",

train['AvSigVersion'].nunique(),train['AVProductStatesIdentifier'].nunique(), train['CityIdentifier'].nunique(),train['Census\_OEMNameIdentifier'].nunique(),train['Census\_InternalBatteryNumberOfCharges'].nunique(),train['Census\_FirmwareVersionIdentifier'].nunique()

)

cutoff=50

#Combiner(train,'AvSigVersion',cutoff,'0.0.0.0')

Combiner(train,'AVProductStatesIdentifier',cutoff, '0.0')

Combiner(train,'CityIdentifier',cutoff,'0.0')

Combiner(train,'Census\_OEMNameIdentifier',cutoff,'3')

Combiner(train,'Census\_InternalBatteryNumberOfCharges',cutoff,'50')

Combiner(train,'Census\_FirmwareVersionIdentifier',cutoff,'50')

print("Final Unique Values:",

train['AvSigVersion'].nunique(),train['AVProductStatesIdentifier'].nunique(), train['CityIdentifier'].nunique(),train['Census\_OEMNameIdentifier'].nunique(),train['Census\_InternalBatteryNumberOfCharges'].nunique(),train['Census\_FirmwareVersionIdentifier'].nunique()

)

return train

#%%

#Remove columns that have one category that take up more than 95% of the data

def RemoveMonopolyCol(train):

print('Removing Columns with Monopoly Categories...')

removeCols=[]

for col in train.columns:

rate = train[col].value\_counts(normalize=True, dropna=False).values[0]

if rate > 0.95:

removeCols.append(col)

train.drop(removeCols,axis=1,inplace=True)

print(f'Dropped {len(removeCols)} columns','Total Column:',train.shape[1])

return train

#%%

def DataPipeline(df,with\_engineering=True):

for i in df.columns:

if df[i].dtype=='O' and not np.issubdtype(df[i].dtype, np.number):

df[i]=df[i].astype('category')

print('Original Columns:',df.shape[1])

datedictAS = np.load('AvSigVersionTimestamps.npy',allow\_pickle=True)[()]

datedictOS = np.load('OSVersionTimestamps.npy',allow\_pickle=True)[()]

print('Begin Pipe...')

if with\_engineering:

df=fe(df).pipe(DecreaseUniqueValues)\

.pipe(RemoveMonopolyCol)\

.pipe(dateFE,'AvSigVersion',datedictAS)\

.pipe(dateFE,'Census\_OSVersion',datedictOS)

else:

df=df.pipe(DecreaseUniqueValues)\

.pipe(RemoveMonopolyCol)\

.pipe(dateFE,'AvSigVersion',datedictAS)\

.pipe(dateFE,'Census\_OSVersion',datedictOS)

df.fillna(method='bfill',inplace=True)

df.fillna(method='ffill',inplace=True)

print('Discretize data for certain column...')

if with\_engineering:

bin\_1=np.arange(df['primary\_drive\_c\_ratio'].min(),df['primary\_drive\_c\_ratio'].max(),500)

df['primary\_drive\_c\_ratio']=np.digitize(df['primary\_drive\_c\_ratio'],bins=bin\_1)

bin\_4=np.arange(df['aspect\_ratio'].min(),df['aspect\_ratio'].max(),20)

df['aspect\_ratio']=np.digitize(df['aspect\_ratio'],bins=bin\_4)

bin\_5=np.arange(df['dpi'].min(),df['dpi'].max(),100)

df['dpi']=np.digitize(df['dpi'],bins=bin\_5)

bin\_6=np.arange(df['MegaPixels'].min(),df['MegaPixels'].max(),20)

df['MegaPixels']=np.digitize(df['MegaPixels'],bins=bin\_6)

bin\_2=np.arange(df['Census\_SystemVolumeTotalCapacity'].min(),df['Census\_SystemVolumeTotalCapacity'].max(),500)

df['Census\_SystemVolumeTotalCapacity']=np.digitize(df['Census\_SystemVolumeTotalCapacity'],bins=bin\_2)

bin\_3=np.arange(df['Census\_OEMModelIdentifier'].min(),df['Census\_OEMModelIdentifier'].max(),300)

df['Census\_OEMModelIdentifier']=np.digitize(df['Census\_OEMModelIdentifier'],bins=bin\_3)

for usecol in df.columns.tolist():

df[usecol] = df[usecol].astype('str')

#Fit LabelEncoder

le = LabelEncoder().fit(np.unique(df[usecol].unique().tolist()))

#At the end 0 will be used for dropped values

df[usecol] = le.transform(df[usecol])+1

print(f'Cleaned data has {df.shape[0]} rows,{df.shape[1]} columns')

return df

#%%

def TrainTest(df,with\_engineering=True):

train=DataPipeline(df,with\_engineering=with\_engineering)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(train.drop(['HasDetections','MachineIdentifier'],axis=1),

train['HasDetections'], test\_size=0.4,random\_state=42)

Y\_train = np.array(Y\_train)

Y\_test = np.array(Y\_test)

train\_ids = X\_train.index

test\_ids = X\_test.index

print('Y\_train',Y\_train.shape)

print('Y\_test',Y\_test.shape)

return X\_train,X\_test,Y\_train,Y\_test,train\_ids,test\_ids

#%% Load saved sample data

train\_path=r'C:\Users\sherr\Sherry\Carlson\Courses\6110 Business Essentials\Final Project\microsoft\_train\_data\_2.csv'

#test\_path=r'C:\Users\sherr\Sherry\Carlson\Courses\6110 Business Essentials\Final Project\microsoft\_test\_data\_2.csv'

df\_raw=pd.read\_csv(train\_path)

df=df\_raw.copy()

X\_train\_1,X\_test\_1,Y\_train\_1,Y\_test\_1,train\_ids\_1,test\_ids\_1=TrainTest(df,with\_engineering=False)

X\_train,X\_test,Y\_train,Y\_test,train\_ids,test\_ids=X\_train\_1,X\_test\_1,Y\_train\_1,Y\_test\_1,train\_ids\_1,test\_ids\_1

print(X\_train.shape)

#%%

#Drop test dataset categories that does not exist in train data set

X\_test['Y']=Y\_test

count=0

for i in X\_train.columns:

print(f'start the {count} column {i}')

count+=1

train\_cat=X\_train[i].value\_counts().index

test\_cat=X\_test[i].value\_counts().index

diff=np.setdiff1d(test\_cat, train\_cat)

print(len(diff))

for j in diff:

print('begin drop',j)

X\_test.drop(X\_test[X\_test[i]==j].index,axis=0,inplace=True)

Y\_test=X\_test['Y']

test\_ids=Y\_test.index

X\_test.drop('Y',axis=1,inplace=True)

print(f'X\_train: {X\_train.shape},X\_test:{X\_test.shape},Y\_train:{Y\_train.shape},X\_test:{Y\_test.shape}')

#Fit OneHotEncoder

ohe = OneHotEncoder(categories='auto', sparse=True, dtype='uint8').fit(X\_train)

#%%

#Transform data using small groups to reduce memory usage

m = 100000

train = vstack([ohe.transform(X\_train[i\*m:(i+1)\*m]) for i in range(X\_train.shape[0] // m + 1)])

test = vstack([ohe.transform(X\_test[i\*m:(i+1)\*m]) for i in range(X\_test.shape[0] // m + 1)])

save\_npz('train.npz', train, compressed=True)

save\_npz('test.npz', test, compressed=True)

del ohe, train, test

gc.collect()

#%%

def ROC(y\_true,pred\_list,pos\_label,color,label):

plt.figure(figsize=[10,10])

for i,j,k in zip(pred\_list,color,label):

fpr, tpr, thresholds = roc\_curve(y\_true,i, pos\_label=pos\_label)

auc\_score = auc(fpr, tpr)

print(f'auc for {k} is {auc\_score}')

plt.plot(fpr,tpr,label=k,color=j)

plt.xlabel('FPR',fontsize=30)

plt.ylabel('TPR',fontsize=30)

plt.legend()

plt.show()

#%%

def print\_score(m):

res = [roc\_auc\_score(m.predict(X\_train), Y\_train), roc\_auc\_score(m.predict(X\_test), Y\_test),

m.score(X\_train, Y\_train), m.score(X\_test, Y\_test)

]

if hasattr(m,'oob\_score\_'): res.append(m.oob\_score\_)

print("train\_auc,test\_auc,train\_accuracy,test\_accuracy\n",res)

#%% LightGBM

skf = StratifiedKFold(n\_splits=2, shuffle=True, random\_state=42)

skf.get\_n\_splits(train\_ids, Y\_train)

lgb\_test\_result = np.zeros(test\_ids.shape[0])

#lgb\_train\_result = np.zeros(train\_ids.shape[0])

#xgb\_test\_result = np.zeros(test\_ids.shape[0])

#xgb\_train\_result = np.zeros(train\_ids.shape[0])

counter = 0

print('\nLightGBM\n')

for train\_index, test\_index in skf.split(train\_ids, Y\_train):

print('Fold {}\n'.format(counter + 1))

train = load\_npz('train.npz')

X\_fit = vstack([train[train\_index[i\*m:(i+1)\*m]] for i in range(train\_index.shape[0] // m + 1)])

X\_val = vstack([train[test\_index[i\*m:(i+1)\*m]] for i in range(test\_index.shape[0] // m + 1)])

X\_fit, X\_val = csr\_matrix(X\_fit, dtype='float32'), csr\_matrix(X\_val, dtype='float32')

y\_fit, y\_val = Y\_train[train\_index], Y\_train[test\_index]

del train

gc.collect()

lgb\_model = lgb.LGBMClassifier(max\_depth=-1,

n\_estimators=30000,

learning\_rate=0.015,

num\_leaves=2\*\*11-200,

colsample\_bytree=0.28,

random\_state=23,

min\_child\_samples=20,

objective='binary',

subsample=0.5,

n\_jobs=-1)

#xgb\_model = xgb.XGBClassifier(max\_depth=6,

# n\_estimators=30000,

# colsample\_bytree=0.2,

# learning\_rate=0.1,

# objective='binary:logistic',

# n\_jobs=-1)

lgb\_model.fit(X\_fit, y\_fit, eval\_metric='auc',

eval\_set=[(X\_val, y\_val)],

verbose=100, early\_stopping\_rounds=100)

del X\_fit, X\_val, y\_fit, y\_val, train\_index, test\_index

gc.collect()

test = load\_npz('test.npz')

test = csr\_matrix(test, dtype='float32')

lgb\_test\_result += lgb\_model.predict\_proba(test)[:,1]

#xgb\_test\_result += xgb\_model.predict\_proba(test)[:,1]

counter += 1

del test

gc.collect()

#%% Predict

lgb\_pred=lgb\_test\_result/3

ROC(Y\_test,[lgb\_pred],2,'red',['LGB'])

#%% Random Forest

m = RandomForestClassifier(n\_estimators=40, min\_samples\_leaf=60, max\_features=0.5, n\_jobs=-1, oob\_score=False)

%time m.fit(X\_train, Y\_train)

print\_score(m)

#%% Predict

pred\_proba=m.predict\_proba(X\_test)

forest\_pred=[]

for i in pred\_proba:

forest\_pred.append(i[1])

result=pd.DataFrame({'ActualResult':list(Y\_test),'PredictProbility':forest\_pred}).sort\_values('PredictProbility',ascending=False)

result.sample(10)

#%% Naive Bayes Model

gnb = GaussianNB()

nb\_model=gnb.fit(X\_train, Y\_train)

pred\_proba =nb\_model.predict\_proba(X\_test)

#%% Predict

nb\_pred=[]

for i in pred\_proba:

nb\_pred.append(i[1])

result=pd.DataFrame({'ActualResult':list(Y\_test),'PredictProbility':nb\_pred}).sort\_values('PredictProbility',ascending=False)

result.sample(10)

#%%

#Save the 3 prediction result

lgb\_pred\_2,forest\_pred\_2,nb\_pred\_2=lgb\_pred.copy(),forest\_pred.copy(),nb\_pred.copy()

eng\_predict=pd.DataFrame({'Actual':Y\_test,'LGB':lgb\_pred\_2,'Random\_Forest':forest\_pred\_2,'Naive Bayes':nb\_pred\_2})

eng\_predict.to\_csv('eng\_predict.csv')

#%% Define function that returns metrics

def Metrics(fig,ax,color,df,i):

df=df[['Actual',i]]

df=df.sort\_values(i,ascending=False).reset\_index()[['Actual',i]]

df['Percision\_pos']=np.cumsum(df['Actual'])/(np.array(df.index)+1)

df['Recall\_pos']=np.cumsum(df['Actual'])/sum(df['Actual'])

df['Combined\_Score']=df['Percision\_pos']\*df['Recall\_pos']\*2/(df['Recall\_pos']\*0.9+df['Percision\_pos'])

max\_value=df[df['Combined\_Score']==df['Combined\_Score'].max()][i].values

print('The cutoff is:',max\_value)

print(df[df['Combined\_Score']==df['Combined\_Score'].max()])

predict=[]

for j in range(len(df)):

if df.loc[j,i]>max\_value:

predict.append(1),

elif df.loc[j,i]<=max\_value:

predict.append(0)

df['predict']=predict

final=df[['Actual','predict']]

total=final.shape[0]

act\_T=final['Actual'].sum()

accu=final[final['Actual']==final['predict']]

TP=final[(final['Actual']==final['predict'])&(final['Actual']==1)]

Total\_P=final[final['predict']==1]

print('Accuracy:',round(len(accu)/total,3))

print('Percision:',round(len(TP)/len(Total\_P),3))

print('Recall:',round(len(TP)/act\_T,3))

ax.plot(df['Combined\_Score'],label=i,color=color,linewidth=2)

#%% Visualize Result

fig,ax=plt.subplots(figsize=[12,8])

Metrics(fig,ax,'steelblue',df,'LGB')

Metrics(fig,ax,'gray',df,'Random\_Forest')

Metrics(fig,ax,'black',df,'Naive Bayes')

plt.legend(fontsize=20,frameon=False)

plt.xticks([1],[""])

plt.yticks([1],[""])

ax.spines['top'].set\_linewidth(0)

ax.spines['right'].set\_linewidth(0)

plt.savefig('modelresult.png',transparent=True,bbox\_inches='tight')

For Finding Part:

Our after model 5% malware Infection rate is calculated based on the original 49.97% which times the 90% recall rate.

We assume there are 200 thousand dollars loss on average for every company due to cyber attacks.

For Business part:-

**References:-**

[**https://www.cnbc.com/2019/10/13/cyberattacks-cost-small-companies-200k-putting-many-out-of-business.html**](https://www.cnbc.com/2019/10/13/cyberattacks-cost-small-companies-200k-putting-many-out-of-business.html)

[**https://www.opswat.com/blog/most-destructive-malware-all-time**](https://www.opswat.com/blog/most-destructive-malware-all-time)

[**https://www.safetydetectives.com/blog/malware-statistics/**](https://www.safetydetectives.com/blog/malware-statistics/)

[**https://heimdalsecurity.com/blog/the-malware-economy/**](https://heimdalsecurity.com/blog/the-malware-economy/)

[**https://blog.emsisoft.com/en/28892/10-most-ridiculous-ransomware-ever/**](https://blog.emsisoft.com/en/28892/10-most-ridiculous-ransomware-ever/)