In [ ]:
```
import pandas as pd
import seaborn as sb
import numpy as np
import matplotlib.pyplot as plt
import scipy as stats
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculatio
```

In [ ]:
```
df = pd.read_csv('BitcoinHeistData.csv')
```

In [ ]:
```
df
```

Out[ ]:

	address	year	day	length	weight	count	looped	neigh
0	111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.008333	1	0	
1	1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.000244	1	0	
2	112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.000000	1	0	
3	1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.003906	1	0	
4	1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.072848	456	0	
...	...	...	...	...	...	...	...	...
2916692	12D3trgho1vJ4mGtWBRPyHdMJK96TRYSry	2018	330	0	0.111111	1	0	
2916693	1P7PputTcVkhXBmXBvSD9MJ3UYPsiou1u2	2018	330	0	1.000000	1	0	
2916694	1KYiKJEfdJtap9QX2v9BXJMpz2SfU4pgZw	2018	330	2	12.000000	6	6	
2916695	15iPUJsRNZQZHmZZVwmQ63srsmughCXV4a	2018	330	0	0.500000	1	0	
2916696	3LFFBxp15h9KSFtaw55np8eP5fv6kdK17e	2018	330	144	0.073972	6800	0	

2916697 rows × 10 columns

In [ ]:
```
# df.info()
df.isna().sum().sum()
```

Out[ ]:
```
0
```

No nan values found

In [ ]:
```
#still for precaution
df.dropna()
```

Out[ ]:

	address	year	day	length	weight	count	looped	neigh

	address	year	day	length	weight	count	looped	neigh
0	111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.008333	1	0	
1	1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.000244	1	0	
2	112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.000000	1	0	
3	1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.003906	1	0	
4	1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.072848	456	0	
...	...	...	...	...	...	...	...	...
2916692	12D3trgho1vJ4mGtWBRPyHdMJK96TRYSry	2018	330	0	0.111111	1	0	
2916693	1P7PputTcVkhXBmXBvSD9MJ3UYPsiou1u2	2018	330	0	1.000000	1	0	
2916694	1KYiKJEfdJtap9QX2v9BXJMpz2SfU4pgZw	2018	330	2	12.000000	6	6	
2916695	15iPUJsRNZQZHmZZVwmQ63srsmughCXV4a	2018	330	0	0.500000	1	0	
2916696	3LFFBxp15h9KSFtaw55np8eP5fv6kdK17e	2018	330	144	0.073972	6800	0	

2916697 rows × 10 columns

In [ ]:
```
n = len(pd.unique(df['address']))
n
```

Out[ ]:
```
2631095
```

In [ ]:
```
df=df.drop(['address'],axis=1)
#dropped address column because two many unique values will only
#contribute to computation complexity when the address feature is not significant
df
```

Out[ ]:

	year	day	length	weight	count	looped	neighbors	income	label
0	2017	11	18	0.008333	1	0	2	1.000500e+08	princetonCerber
1	2016	132	44	0.000244	1	0	1	1.000000e+08	princetonLocky
2	2016	246	0	1.000000	1	0	2	2.000000e+08	princetonCerber
3	2016	322	72	0.003906	1	0	2	7.120000e+07	princetonCerber
4	2016	238	144	0.072848	456	0	1	2.000000e+08	princetonLocky
...	...	...	...	...	...	...	...	...	...
2916692	2018	330	0	0.111111	1	0	1	1.255809e+09	white
2916693	2018	330	0	1.000000	1	0	1	4.409699e+07	white
2916694	2018	330	2	12.000000	6	6	35	2.398267e+09	white
2916695	2018	330	0	0.500000	1	0	1	1.780427e+08	white
2916696	2018	330	144	0.073972	6800	0	2	1.123500e+08	white

2916697 rows × 9 columns

Part 1

In [ ]:
```
target=['label']
features=df.columns[:-1]
features
```

Out[ ]:
```
Index(['year', 'day', 'length', 'weight', 'count', 'looped', 'neighbors',
       'income'],
      dtype='object')
```

In [ ]:
```
X=df[features]
print(X)
Y=df[target]
print(Y)
```

```
         year  day  length     weight  count  looped  neighbors        income
0        2017   11      18   0.008333      1       0          2  1.000500e+08
1        2016  132      44   0.000244      1       0          1  1.000000e+08
2        2016  246       0   1.000000      1       0          2  2.000000e+08
3        2016  322      72   0.003906      1       0          2  7.120000e+07
4        2016  238     144   0.072848    456       0          1  2.000000e+08
...       ...  ...     ...        ...    ...     ...        ...           ...
2916692  2018  330       0   0.111111      1       0          1  1.255809e+09
2916693  2018  330       0   1.000000      1       0          1  4.409699e+07
2916694  2018  330       2  12.000000      6       6         35  2.398267e+09
2916695  2018  330       0   0.500000      1       0          1  1.780427e+08
2916696  2018  330     144   0.073972   6800       0          2  1.123500e+08

[2916697 rows x 8 columns]
                  label
0        princetonCerber
1         princetonLocky
2        princetonCerber
3        princetonCerber
4         princetonLocky
...                  ...
2916692           white
2916693           white
2916694           white
2916695           white
2916696           white

[2916697 rows x 1 columns]
```

In [ ]:
```
#encoding the target column
label_encoder = preprocessing.LabelEncoder()
df['label']= label_encoder.fit_transform(df['label'])
```

In [ ]:
```
df = df.sample(frac=1)
train_size = 0.70
test_size = 0.15
valid_size=0.15
```

```
train_index = int(len(df)*train_size)

df_train = df[0:train_index]
df_rem = df[train_index:]

valid_index = int(len(df)*valid_size)

df_valid = df[train_index:train_index+valid_index]
df_test = df[train_index+valid_index:]

X_train, y_train = df_train.drop(columns='label'), df_train['label']
X_valid, y_valid = df_valid.drop(columns='label'), df_valid['label']
X_test, y_test = df_test.drop(columns='label'), df_test['label']
trees_both=[]
```

In [ ]:

```
# Create Decision Tree classifer object with entropy
accuracy_entropy=[]
depth=[4,8,10,15,20]
for i in depth:
    clf = DecisionTreeClassifier(criterion="entropy", max_depth=i)
    # Train Decision Tree Classifer
    clf = clf.fit(X_train,y_train)
    #Predict the response for test dataset
    y_pred = clf.predict(X_valid)
    print("Accuracy for depth",i,":",metrics.accuracy_score(y_valid, y_pred))
    accuracy_entropy.append(metrics.accuracy_score(y_valid, y_pred))
```

```
Accuracy for depth 4 : 0.9855155610005851
Accuracy for depth 8 : 0.9857807014335869
Accuracy for depth 10 : 0.9870424041837331
Accuracy for depth 15 : 0.9879201104447045
Accuracy for depth 20 : 0.9860984128145114
```

In [ ]:

```
# Create Decision Tree classifer object with gini index
accuracy_gini=[]
for i in depth:
    clf = DecisionTreeClassifier(criterion="gini", max_depth=i)
    # Train Decision Tree Classifer
    clf = clf.fit(X_train,y_train)
    #Predict the response for test dataset
    y_pred = clf.predict(X_valid)
    print("Accuracy for depth",i,":",metrics.accuracy_score(y_valid, y_pred))
    accuracy_gini.append(metrics.accuracy_score(y_valid, y_pred))
```

```
Accuracy for depth 4 : 0.9855155610005851
Accuracy for depth 8 : 0.9862081260971328
Accuracy for depth 10 : 0.9867864065242832
Accuracy for depth 15 : 0.9878286827091867
Accuracy for depth 20 : 0.9863589818607372
```

In [ ]:

```
for i in range(len(depth)):
    print("Accuracy for depth",depth[i],"with entropy:",accuracy_entropy[i])
    print("Accuracy for depth",depth[i],"with gini:",accuracy_gini[i])
```

```
Accuracy for depth 4 with entropy: 0.9855155610005851
Accuracy for depth 4 with gini: 0.9855155610005851
Accuracy for depth 8 with entropy: 0.9857807014335869
```

```
Accuracy for depth 8 with gini: 0.9862081260971328
Accuracy for depth 10 with entropy: 0.9870424041837331
Accuracy for depth 10 with gini: 0.9867864065242832
Accuracy for depth 15 with entropy: 0.9879201104447045
Accuracy for depth 15 with gini: 0.9878286827091867
Accuracy for depth 20 with entropy: 0.9860984128145114
Accuracy for depth 20 with gini: 0.9863589818607372
```

The best accuracy for the ginni and entropy creteria is best with the max depth of 15

In [ ]:

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=15)
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_valid)
print("Accuracy for depth",15,":",metrics.accuracy_score(y_valid, y_pred))
accuracy_entropy.append(metrics.accuracy_score(y_valid, y_pred))
```

```
Accuracy for depth 15 : 0.9878766822703335
```

In [ ]:

```
clf = DecisionTreeClassifier(criterion="gini", max_depth=15)
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy for depth",15,":",metrics.accuracy_score(y_test, y_pred))
# accuracy_gini.append(metrics.accuracy_score(y_valid, y_pred))
```

```
Accuracy for depth 15 : 0.9881921619360649
```

Part 2

Ensembling is a method to combine multiple not-so-good models to get a better performing model. Create 100 different decision stumps (max depth 3). For each stump, train it on randomly selected 50% of the training data, i.e., select data for each stump separately. Now, predict the test samples' labels by taking a majority vote of the output of the stumps. How is the performance affected as compared to parts (a)

In [ ]:

```
def train_trees(X, Y, num_trees):
    indices = [i for i in range(X.shape[0])]

    # Here, note that we have set max_depth = 3 which
    # makes all the classifiers weak
    trees = [DecisionTreeClassifier(max_depth=3) for _ in range(num_trees)]
    for tree in trees:
        # Selecting n random samples with replacement from training set
        random_indices = np.random.choice(indices, X.shape[0])
        print(random_indices)
        # Bootstrap training data
        X_bootstrap= X.iloc[random_indices]
        Y_bootstrap= Y.iloc[random_indices]
        # X_bootstrap = X[random_indices]
        # Y_bootstrap = Y[random_indices]

        tree.fit(X_bootstrap, Y_bootstrap)
```

```
    return trees
```

In [ ]:

```
def predict(X, trees):
    predictions = []
    for tree in trees:
        Y_pred = tree.predict(X)
        predictions.append(Y_pred)

    predictions = np.array(predictions)

    # Aggregating all predictions to get final prediction
    # Since this is a classification problem, we use mode
    # i.e. the prediction that occurs the maximum number
    # of times. In case of regression problem, we use mean
    prediction = np.array(stats.stats.mode(predictions))
    return prediction[0, 0, :]
```

In [ ]:

```
# NUM_RANDOM_FEATURES =
# put 50% of data from the dataset df into x train and y train
df = df.sample(frac=1)
train_size = 0.50
# test_size = 0.50

train_index = int(len(df)*train_size)

df_train = df[0:train_index]
df_rem = df[train_index:]

# df_test = df[train_index+valid_index:]

X_train, y_train = df_train.drop(columns='label'), df_train['label']
# X_valid, y_valid = df_valid.drop(columns='label'), df_valid['label']
X_test, y_test = df_rem.drop(columns='label'), df_rem['label']
```

In [ ]:

```
# accuracies = []
trees = train_trees(X_train, y_train, 100)
prediction = predict(X_test, trees)
# accuracy = (prediction == y_test).sum() / prediction.shape[0]
# accuracies.append(accuracy)

# plt.plot(accuracies)
```

```
[ 487038 1007506  563817 ... 1368765 1150143   97461]
[1407639   39934  202282 ...   92420  423861  476474]
[ 790485 1354128  382393 ...  244555  422526  909608]
[ 300892  159636  801999 ... 1300258 1271287 1307805]
[ 812440 1316672  967794 ... 1256109  171250 1216143]
[380163 598884 413676 ... 458473 324453 856961]
[ 519269  548718 1156057 ...  183524   86952  243393]
[507258 233889 106903 ... 805654 701789 505699]
[ 546497  658392  698508 ...  786156 1368889  217902]
[1244210  739834 1293874 ...  594322  737519  569317]
[1038793  910238  966858 ...   81740  819588  153268]
[1240813 1078517  307703 ...  938405  924306  337887]
[492803 674715 517661 ... 408470 492369 776330]
```

```
[ 664939  735775  963678 ...    8297 1247053  459211]
[ 51247 636122 261997 ... 755708 521831 792985]
[993794 286242 565969 ... 659036 765881 641545]
[569821 769165 680852 ... 735125 779739 959945]
[ 605713 1404682  976885 ...  740845  944513  477987]
[  97579 1294435  467748 ...   45698 1303553  756960]
[ 664463   29629  719953 ... 1427001  529414 1419260]
[1287116  636249  427210 ... 1253818 1075314  715063]
[ 757228 1274626  334159 ...  279499 1424781 1428208]
[ 175747  930739  953735 ... 1161898 1116252  959795]
[ 983968  488997   46597 ... 1185829  930460 1208424]
[1361592   69999  591252 ... 1163116   49587  951567]
[  61837  975933  864424 ... 1397180 1001041  946954]
[ 974121  479586  255236 ... 1308838    9128  591585]
[ 716991  949564  717526 ...  490460  886779 1365359]
[ 493381  150331 1034749 ... 1358965  672295  985807]
[ 283845  859449 1447358 ... 1258167  233250  436222]
[ 982817  356944 1454024 ...  330816 1425310  181151]
[476331 978760 897728 ...  45593 499549 719541]
[ 677586  738497 1175960 ... 1066365  731885  378190]
[ 872939  857057  181534 ...  872671    1663 1402775]
[  45960  418015   50944 ... 1328696  836912  368057]
[ 443113 1183829  885177 ...  974229  614334 1415379]
[355546 845810 492055 ... 347048 112993 152460]
[ 487304  676929 1044852 ...  504399  208439 1213199]
[1428359  671319  643070 ... 1409111  357178   82470]
[  32192  860616  167497 ...  655095 1031646 1002204]
[ 766056 1146619  917254 ...  939197  676145 1310461]
[1238937 1098531  774903 ... 1053023   72965  418299]
[1242303  505996 1300313 ...  696197 1304215 1095193]
[  89426  184651  787849 ...  270194 1046648  745780]
[1160325  834564  353747 ... 1211148  510153  876046]
[452027 811146 805860 ... 825108 384110 322750]
[123501 411749 842365 ... 432345 195655 338327]
[1342797  361052  453262 ...  577745  290144 1302703]
[1368190 1277343  397451 ...  719887   66039  187308]
[656832 803070 281982 ... 763158 725581 227831]
[1156313  961013  313496 ...  216963  669786  220843]
[1380000  641153  952141 ...  587820 1351046  619124]
[ 494407  312887  820859 ... 1447779 1411868  276948]
[1108418  989178  389155 ...  700877 1230786  442667]
[  87023  249857 1161162 ... 1100196  250857  798845]
[ 677355  165861 1048584 ...  884577  211382  688622]
[1010041  438591  843989 ...  710285  253748  189112]
[ 579724 1299307  414920 ...  171022 1310999 1090755]
[1156825 1336039  162910 ...  858178  336273 1066958]
[ 824044 1373202   72400 ... 1394963 1004906 1087592]
[1216310  798048 1078414 ...  261018 1096595  802039]
[224611 316911 662692 ... 554168 699407 510108]
[ 238799  322937  579349 ...  845988 1024206  399991]
[ 394539  485123 1096147 ... 1073816 1415213  908968]
[ 598277 1035679 1311569 ...  268564  178296  615053]
[ 414048  886174 1096776 ... 1404711  679012 1249364]
[1133408 1195699  136425 ...  798810 1022772  987490]
[  66851  885814  294697 ... 1073227  620932  630341]
[ 975644  486284 1303351 ... 1394504  339615 1092543]
[926371 237732 580636 ... 899039 695894 105346]
[  47709  560631  794034 ... 1103234  721924 1151959]
[  95622  571195  648057 ... 1349376  349776  937730]
[1082161 1398378  360932 ...   34291 1013879  623918]
```

```
[ 352656 1282263   79299 ... 1323511  259815 1110811]
[1215300  998254  341342 ... 1099292 1215581  595246]
[ 443669  928346  999042 ... 1010714 1071909  190991]
[ 953301  773781  346109 ... 1453316  265343 1020354]
[ 410902 1289046 1429213 ...  284940 1269382 1184667]
[ 427767 1418616  454440 ...  462357  474576  671479]
[ 645545  804430 1324636 ...  428144  332429  762827]
[1064735 1251350  562385 ...  688578 1371566  414722]
[ 937302 1142144  570637 ...  844122  660801 1019414]
[1203832 1402864  646885 ...  908482  481109 1388782]
[622465 283471 706428 ... 202711 133764 220826]
[1175705 1208527 1193103 ...  296395  176641 1078478]
[ 774639  397155  812513 ... 1138326 1009635 1378101]
[742162  10341 897202 ... 984904 800333 799714]
[ 245072 1399541  856299 ...  200808  716042 1325927]
[ 873308  974438  452058 ... 1264068 1058405  346096]
[ 255207 1327077  587955 ...   91385  323130  639814]
[393722 403170 297684 ... 685136 650412 115588]
[ 586775  680916 1075386 ...  184380 1455744  318247]
[ 529183    4582  205242 ... 1002904  710015  157413]
[1361544  353017  321029 ...  425699 1334477  474951]
[ 770879 1390030  782581 ...   70825  590291  465473]
[1455195  424385  187844 ...  221484  341448  175547]
[ 366292  980816  260365 ... 1247176   19642  352156]
[1167245  111012   41229 ...  638733  778621 1264197]
[ 312166 1293266  253087 ... 1274547  222766  995451]
[ 372170  381616  133265 ...  323696  401585 1019053]
```

In [ ]:
```
accuracy = (prediction == y_test).sum() / prediction.shape[0]
accuracy
```

Out[ ]:
```
0.985858666204043
```

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=100, learning_rate=1, random_state=42)

# Train Adaboost Classifer
model1 = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_abc = model1.predict(X_test)
```

In [ ]:
```
accuracy = (y_pred_abc == y_test).sum() / y_pred_abc.shape[0]
accuracy
```

Out[ ]:
```
0.9858230094442414
```

In [ ]:
```

```