COMP7703 Final Exam

qinhao xie 47334915

1.data processing

read data

first of all, we need to read data from file to DataFrame.

In [ ]:
```
import pandas as pd
# load dataset into Pandas DataFrame
df = pd.read_csv("CPU_benchmark_v4.csv")
```

In [ ]:
```
df.iloc[275:277,:]
```

Out[ ]:

	cpuName	price	cpuMark	cpuValue	threadMark	threadValue	TDP	powerPerf	cores	te
275	Intel Xeon E5-2695 v4 @ 2.10GHz	592.1	20373	34.41	1948	3.29	120.0	169.77	18	
276	AMD Ryzen 7 PRO 5875U	NaN	20369	NaN	2919	NaN	15.0	1,357.93	8	

from this table, we can discover that the number in the line 276, powerPerf column has a comma inside. to make it pure number, we use pandas to replace all "," in those columns.

In [ ]:
```
def comma(string):
    if isinstance(string,str):
        return string.replace(",","")
    return string

#df = df.iloc[:,1:-2].applymap(lambda x: x.replace(",",""))
df=pd.concat([df.iloc[:,0:1] , df.iloc[:,1:-2].applymap(comma),df.iloc[:,-2:]] , ax
```

change data type from str to float

the numbers are still stored in string. changing it to float for calculations later.

In [ ]:
```
df.iloc[:,1:-2]=df.iloc[:,1:-2].astype(float)
```

In [ ]:
```
import numpy as np
from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit(df.iloc[:,1:-2])
df.iloc[:,1:-2]=imp_mean.transform(df.iloc[:,1:-2])
df
```

Out[ ]:

	cpuName	price	cpuMark	cpuValue	threadMark	threadValue	TDP	pow
0	AMD Ryzen Threadripper PRO 5995WX	441.500946	108822.0	35.354911	3330.0	15.189009	280.000000	388.6
1	AMD EPYC 7763	7299.990000	88338.0	12.100000	2635.0	0.360000	280.000000	315.4
2	AMD EPYC 7J13	441.500946	86006.0	35.354911	2387.0	15.189009	62.375557	121.6
3	AMD EPYC 7713	7060.000000	85861.0	12.160000	2727.0	0.390000	225.000000	381.6
4	AMD Ryzen Threadripper PRO 3995WX	6807.980000	83971.0	12.330000	2626.0	0.390000	280.000000	299.9
...	...	...	...	...	...	...	...	
3820	Intel Pentium 4 1.60GHz	441.500946	84.0	35.354911	225.0	15.189009	38.000000	2.2
3821	Intel Pentium 4 1400MHz	441.500946	83.0	35.354911	180.0	15.189009	54.700000	1.5
3822	Intel Pentium 4 1500MHz	441.500946	81.0	35.354911	223.0	15.189009	57.800000	1.4
3823	VIA Eden 1000MHz	441.500946	80.0	35.354911	83.0	15.189009	5.000000	16.0
3824	Intel Pentium 4 1300MHz	441.500946	77.0	35.354911	203.0	15.189009	51.600000	1.5

3825 rows × 12 columns

first of all, let's have a look at the shape of the data

In [ ]:
```
row_num,col_num = df.shape
print(f"row_num: {row_num},col_num: {col_num}")
```

```
row_num: 3825,col_num: 12
```

descriptive statistics:

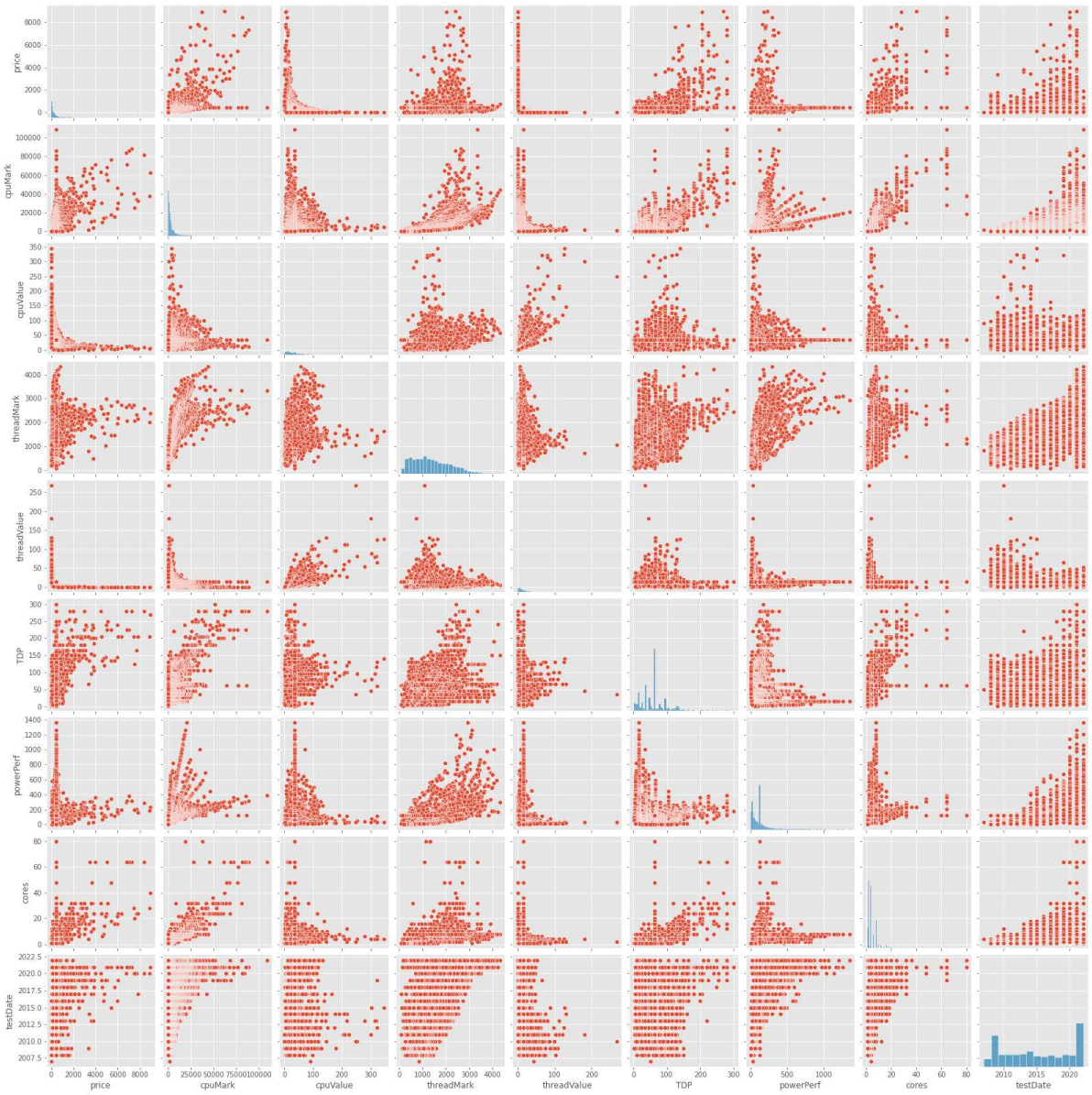
In [ ]:
```
pd.set_option('display.max_columns', None)
df.iloc[:,1:-2].agg(['mean', 'median', 'std', 'skew', 'kurtosis'])
```

Out[ ]:

	price	cpuMark	cpuValue	threadMark	threadValue	TDP	powerPerf	
mean	441.500946	5992.305882	35.354911	1391.107451	15.189009	62.375557	121.623239	5
median	441.500946	2331.000000	35.354911	1274.000000	15.189009	62.375557	101.860000	4
std	650.647576	9617.551479	25.678033	815.667528	12.839840	43.085772	137.600438	6
skew	6.936838	3.743892	4.517915	0.623742	5.425010	1.592552	3.195886	5
kurtosis	62.655218	19.958051	37.313777	-0.168973	61.345188	4.178892	14.868824	47

In [ ]:

```
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    import seaborn as sns
    sns.pairplot(df)
```



from these pictures we can see the core number are coorelated to nearly all other features, the more core the higher price, performance and tdp.

After a brief look on the data, we assume that:
we can differ amd and intel cpu base on those features.

To better analyse the data, we add a new column called brand:

In [ ]:
```
df['brand'] = df['cpuName'].apply(lambda x: x.split(' ')[0])
df1=df.loc[df['brand']== 'Intel' ]
df2=df.loc[df['brand']== 'AMD' ]
df =pd.concat([df1,df2] , axis = 0)
df=df.reset_index(drop=True)
df
```

Out[ ]:

	cpuName	price	cpuMark	cpuValue	threadMark	threadValue	TDP	powerF
0	Intel Xeon Platinum 8380 @ 2.30GHz	8978.000000	62317.0	6.940000	2385.0	0.270000	270.000000	230.810
1	Intel Xeon Platinum 8358 @ 2.60GHz	5097.750000	54416.0	10.670000	2382.0	0.470000	250.000000	217.670
2	Intel Xeon Gold 6348 @ 2.60GHz	3173.000000	53112.0	16.740000	2455.0	0.770000	235.000000	226.010
3	Intel Xeon Platinum 8375C @ 2.90GHz	441.500946	51836.0	35.354911	2439.0	15.189009	300.000000	172.790
4	Intel Xeon Platinum 8347C @ 2.10GHz	441.500946	49386.0	35.354911	2445.0	15.189009	210.000000	235.170
...	...	...	...	...	...	...	...	
3259	AMD Athlon XP 1600+	441.500946	174.0	35.354911	265.0	15.189009	62.375557	121.623
3260	AMD Athlon XP 1500+	441.500946	167.0	35.354911	251.0	15.189009	62.375557	121.623
3261	AMD Athlon 64 2000+	441.500946	154.0	35.354911	173.0	15.189009	8.000000	19.300
3262	AMD G-T44R	441.500946	130.0	35.354911	328.0	15.189009	9.000000	14.440
3263	AMD G-T40R	441.500946	120.0	35.354911	327.0	15.189009	5.500000	21.750

3264 rows × 13 columns

split train and test sets

In [ ]:
```
#split train and test set
import numpy as np
from sklearn.model_selection import train_test_split
```

```
features=df.iloc[:,1:-3]
target = df["brand"]
X_train, X_test ,y_train, y_test= train_test_split(features,target,test_size=0.33, r
```

In [ ]:
```
target
```

Out[ ]:
```
0       Intel
1       Intel
2       Intel
3       Intel
4       Intel
        ...
3259      AMD
3260      AMD
3261      AMD
3262      AMD
3263      AMD
Name: brand, Length: 3264, dtype: object
```

In [ ]:
```
#have a look at data
X_train
X_test
```

Out[ ]:

	price	cpuMark	cpuValue	threadMark	threadValue	TDP	powerPerf	cores	testDa
1641	441.500946	1148.0	35.354911	1079.0	15.189009	15.0	76.51	2.0	201
134	778.000000	22042.0	28.330000	2727.0	3.500000	165.0	133.59	10.0	202
411	320.290000	10747.0	33.550000	2652.0	8.280000	80.0	134.34	6.0	201
203	386.000000	18248.0	47.280000	3258.0	8.440000	65.0	280.74	8.0	202
889	250.800000	4819.0	19.220000	2493.0	9.940000	60.0	80.32	2.0	201
...	...	...	...	...	...	...	...	...	
1278	99.900000	2364.0	23.660000	1394.0	13.950000	35.0	67.53	2.0	201
1575	210.190000	1326.0	6.310000	1039.0	4.940000	82.0	16.17	4.0	201
1815	54.000000	752.0	13.930000	841.0	15.570000	65.0	11.57	2.0	200
2775	116.150000	1557.0	13.400000	1058.0	9.110000	45.0	34.60	3.0	201
1498	42.000000	1614.0	38.430000	1396.0	33.240000	35.0	46.11	2.0	201

1078 rows × 9 columns

2.Data Trainng

to train our data, we firstly conduct a hierachical clustering method to explore the characteristic of our dataset. the first step is to standardized the data to 0 mean and 1 variance to remove the effect of different scale of data. the second step we implement is seperate our data into training set and test set.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
# Standardizing the features
```

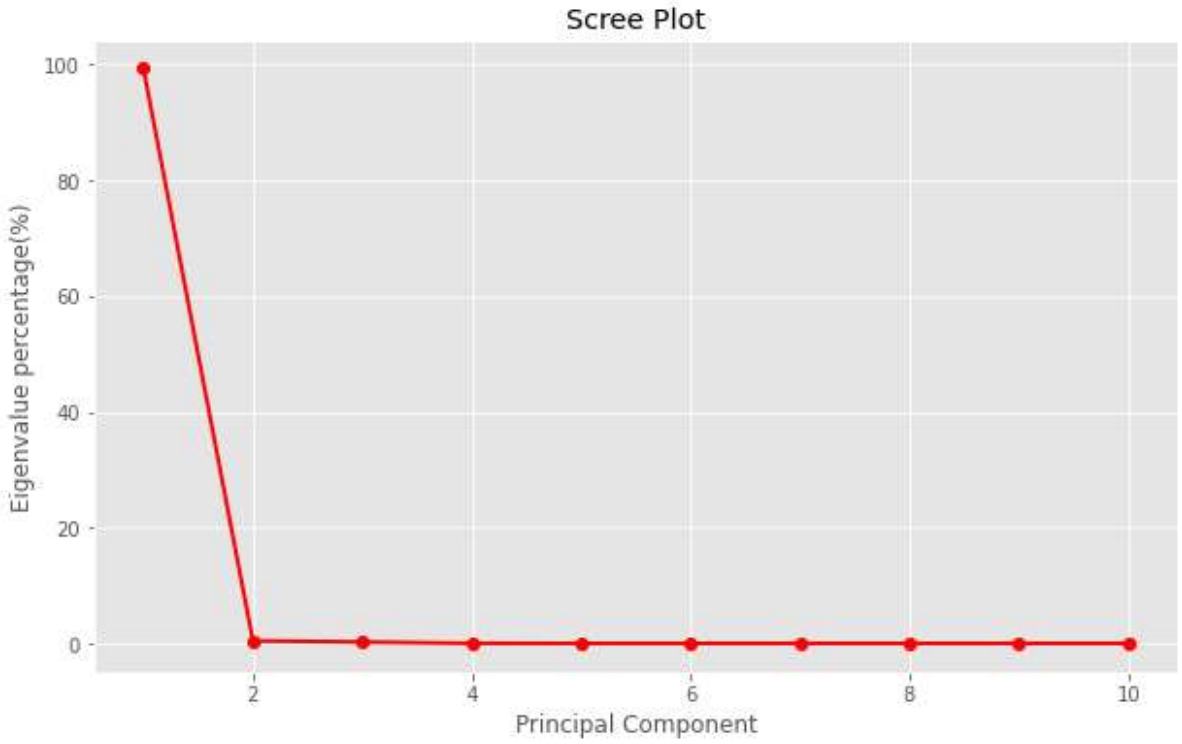
```
X_train = StandardScaler().fit_transform(X_train)
X_test = StandardScaler().fit_transform(X_test)
```

In [ ]:

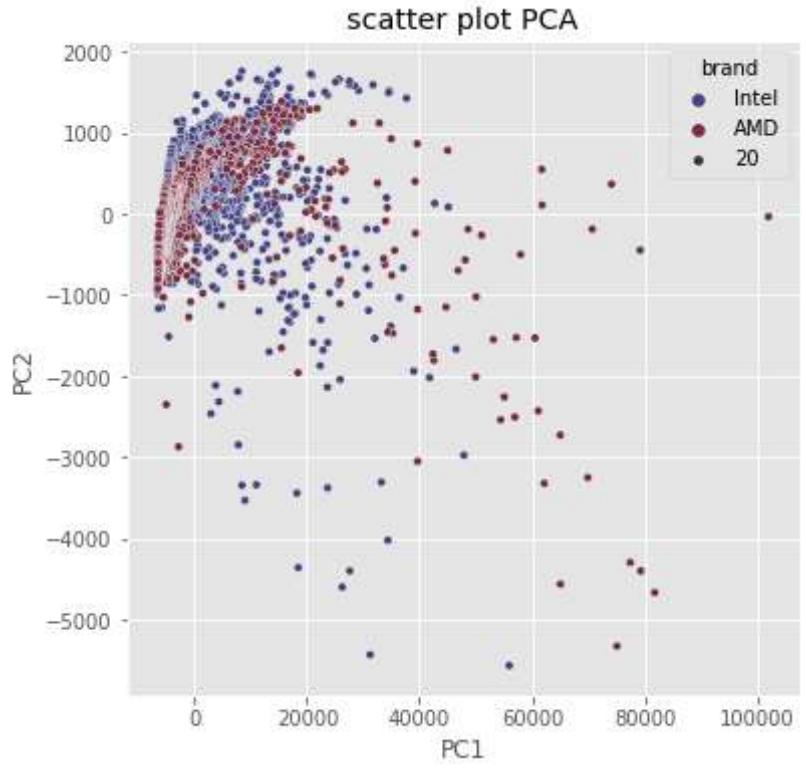
```
import numpy as np
import pandas as pd
from scipy.io import loadmat
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sb

def PCA(X , num_components):
    #Step-1
    X_meaned = X - np.mean(X , axis = 0)

    #Step-2
    cov_mat = np.cov(X_meaned , rowvar = False)

    #Step-3
    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

    #Step-4
    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    #Step-5
    eigenvector_subset = sorted_eigenvectors[:,0:num_components]

    #Step-6
    X_reduced = np.dot(eigenvector_subset.transpose() , X_meaned.transpose() ).trans


        # scree_plot:
    #print the first two
    print("percentage of first two conponents")
    print(sorted_eigenvalue[0:2]*100/np.sum(sorted_eigenvalue))

    figure=plt.figure(figsize=(10,6))
    x=np.arange(len(sorted_eigenvalue[0:num_components+10])) +1
    plt.plot(x,sorted_eigenvalue[0:num_components+10]*100/np.sum(sorted_eigenvalue),
    plt.title('Scree Plot')
    plt.xlabel('Principal Component')
    plt.ylabel('Eigenvalue percentage(%)')
    plt.show()
            # scree_plot2:

    if len(sorted_eigenvalue)>50:
        figure=plt.figure(figsize=(10,6))
        x=np.arange(len(sorted_eigenvalue)) +1
        plt.plot(x,sorted_eigenvalue/np.sum(sorted_eigenvalue)*100, 'ro-', linewidth
        plt.title('Scree Plot')
        plt.xlabel('Principal Component')
        plt.ylabel('Eigenvalue percentage(%)')
        plt.show()
    return X_reduced



"""swiss_roll"""
#Applying it to PCA function
mat_reduced = PCA(features , 2)

#Creating a Pandas DataFrame of reduced Dataset
```

```
principal_df = pd.DataFrame(mat_reduced , columns = ['PC1','PC2'])

#Concat it with target variable to create a complete Dataset
principal_df = pd.concat([principal_df , df['brand']] , axis = 1)

#ploting

plt.figure(figsize = (6,6))
plt.title('scatter plot PCA')
sb.scatterplot(data = principal_df , x = 'PC1',y = 'PC2' , hue = 'brand' , size = 2
```

```
percentage of first two conponents
[99.30757925  0.44057244]
```



Out[ ]: <AxesSubplot:title={'center':'scatter plot PCA'}, xlabel='PC1', ylabel='PC2'>



from the pca visualization, at least the data are not clearly splited into two cluster, and the

first principal explained all variances

here we can also see similar outcome. the data can be seen splited to two groups and one group can be separated further.

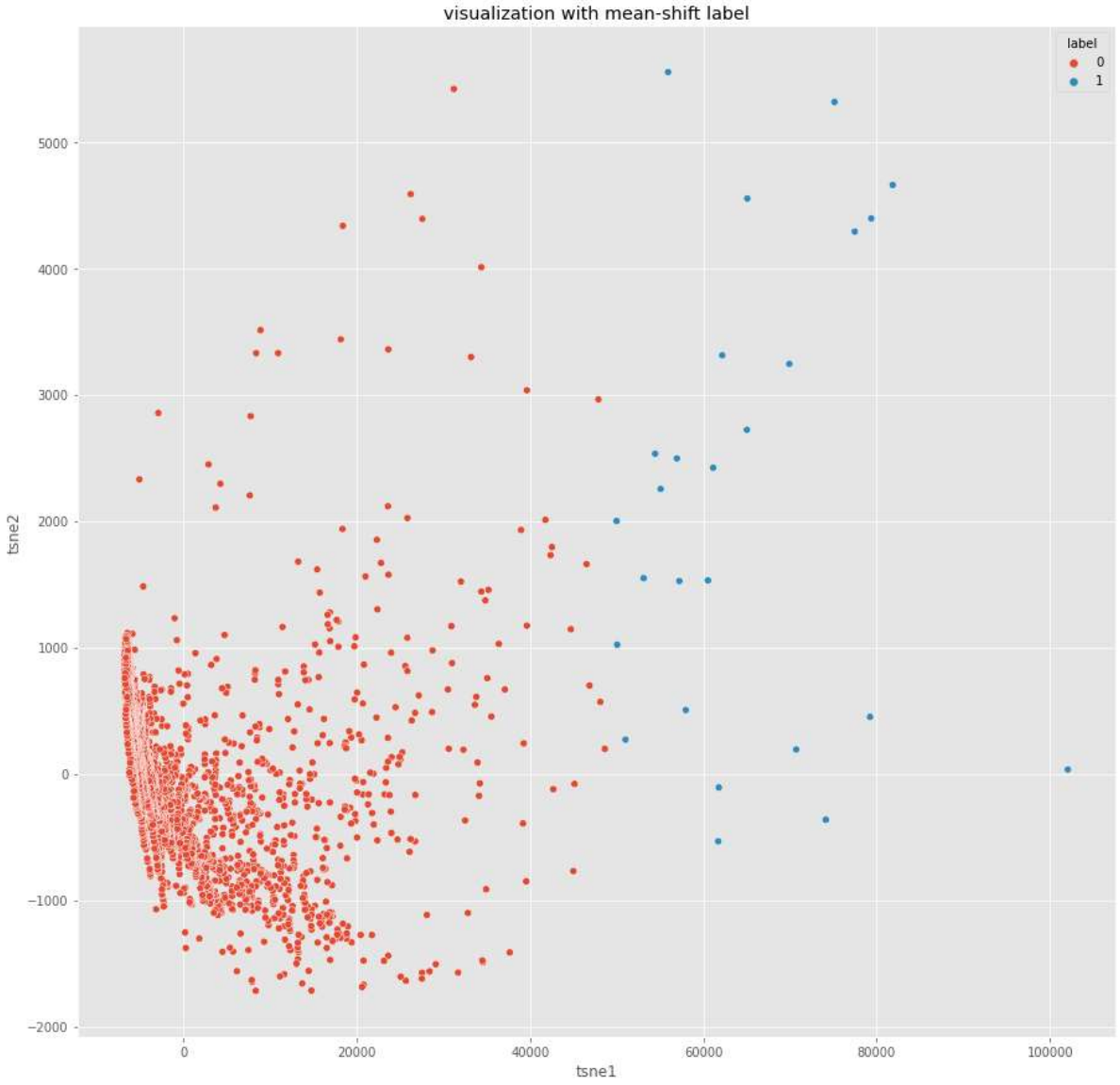
In [ ]:
```
#mean shift
from sklearn.cluster import estimate_bandwidth

band = estimate_bandwidth(features, n_jobs=-1)
print("estimated band:",band)
from sklearn.cluster import MeanShift
clustering = MeanShift(bandwidth=15000).fit(features.select_dtypes(include='float'))
features['label'] =clustering.labels_
cluster_centers = clustering.cluster_centers_
labels_unique = np.unique(clustering.labels_)
n_clusters_ = len(labels_unique)
print("number of clusters:",n_clusters_)
```

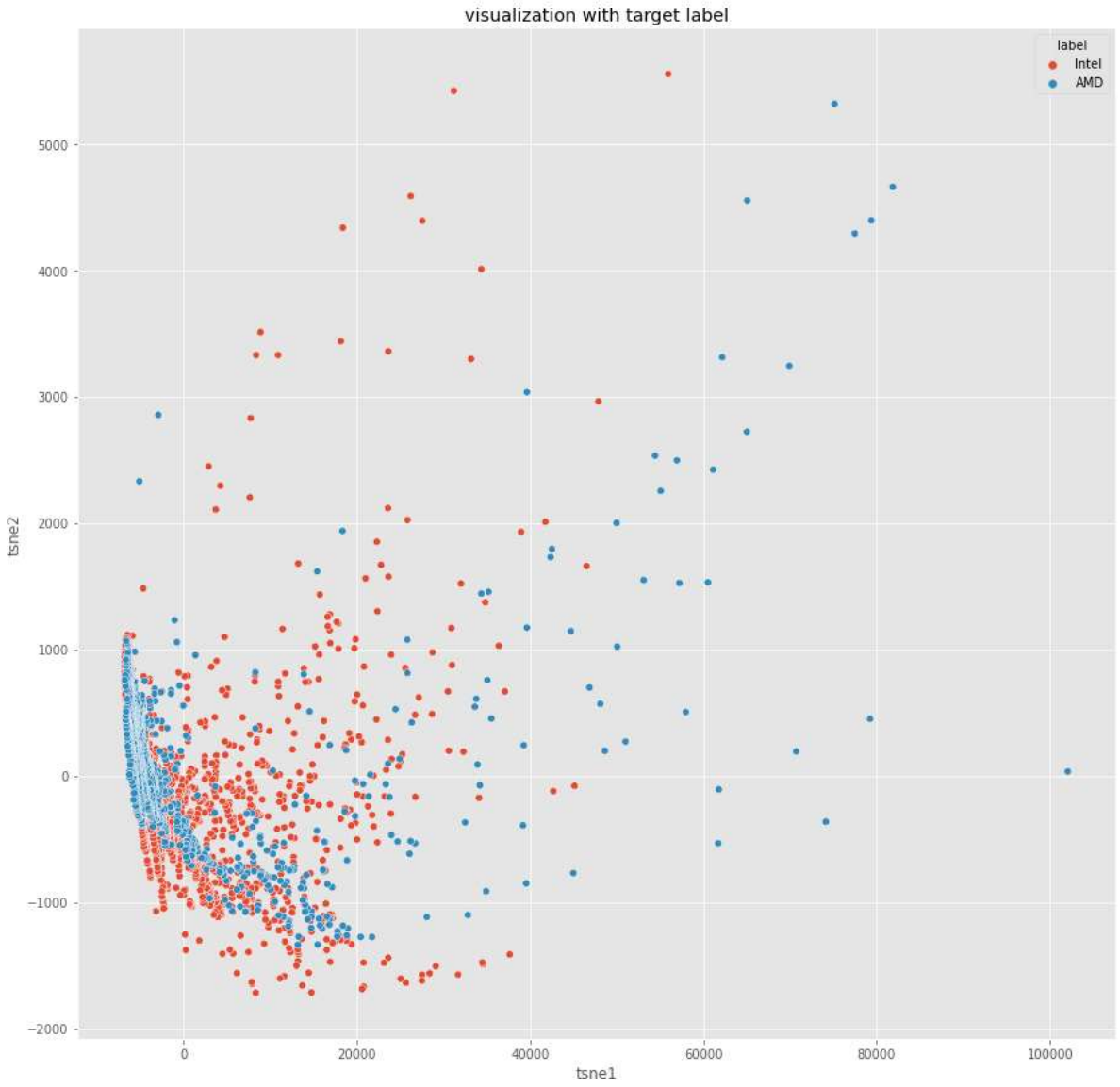
```
estimated band: 4894.307799636441
number of clusters: 2
```

the auto estimated bandwidth is really not reliable, when i set the bandwidth to 2, it clustered the data into many small clusters, i finally change bandwidth to 15000 and the mean-shift clustering specified data to two cluster

In [ ]:
```
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    from sklearn.manifold import TSNE
    import seaborn as sns
    #t-SNE is based on a stochastic (random) process, that is why I set the random_s
    tsne = TSNE(n_components=2,random_state=0,n_jobs=-1,learning_rate='auto',init='
    tsne_results = tsne.fit_transform(features)
    print(tsne_results)
    tsne_results=pd.DataFrame(tsne_results,columns=['tsne1', 'tsne2'])
    #Visualize the data
    tsne_results['label'] = features['label']
    sns.scatterplot(data = tsne_results, x = 'tsne1', y = 'tsne2', hue='label')
    plt.title("visualization with mean-shift label")
    plt.show()
    tsne_results['label'] = df['brand']
    sns.scatterplot(data = tsne_results, x = 'tsne1', y = 'tsne2', hue='label')
    plt.title("visualization with target label")
    plt.show()
```

```
[[55937.797    5551.881  ]
 [47898.99     2961.9062 ]
 [46525.336    1658.8221 ]
 ...
 [-6584.9697   1065.8534 ]
 [-6683.923     794.63855]
 [-6735.4224    756.2553 ]]
```


visualization with mean-shift label
label
0
1
tsne2
tsne1
5000
4000
3000
2000
1000
0
−1000
−2000
0
20000
40000
60000
80000
100000



here I used tsne to visualize the data, at first I used the labels given by mean-shift to draw the first picture, and then drawed the second picture with target labels.

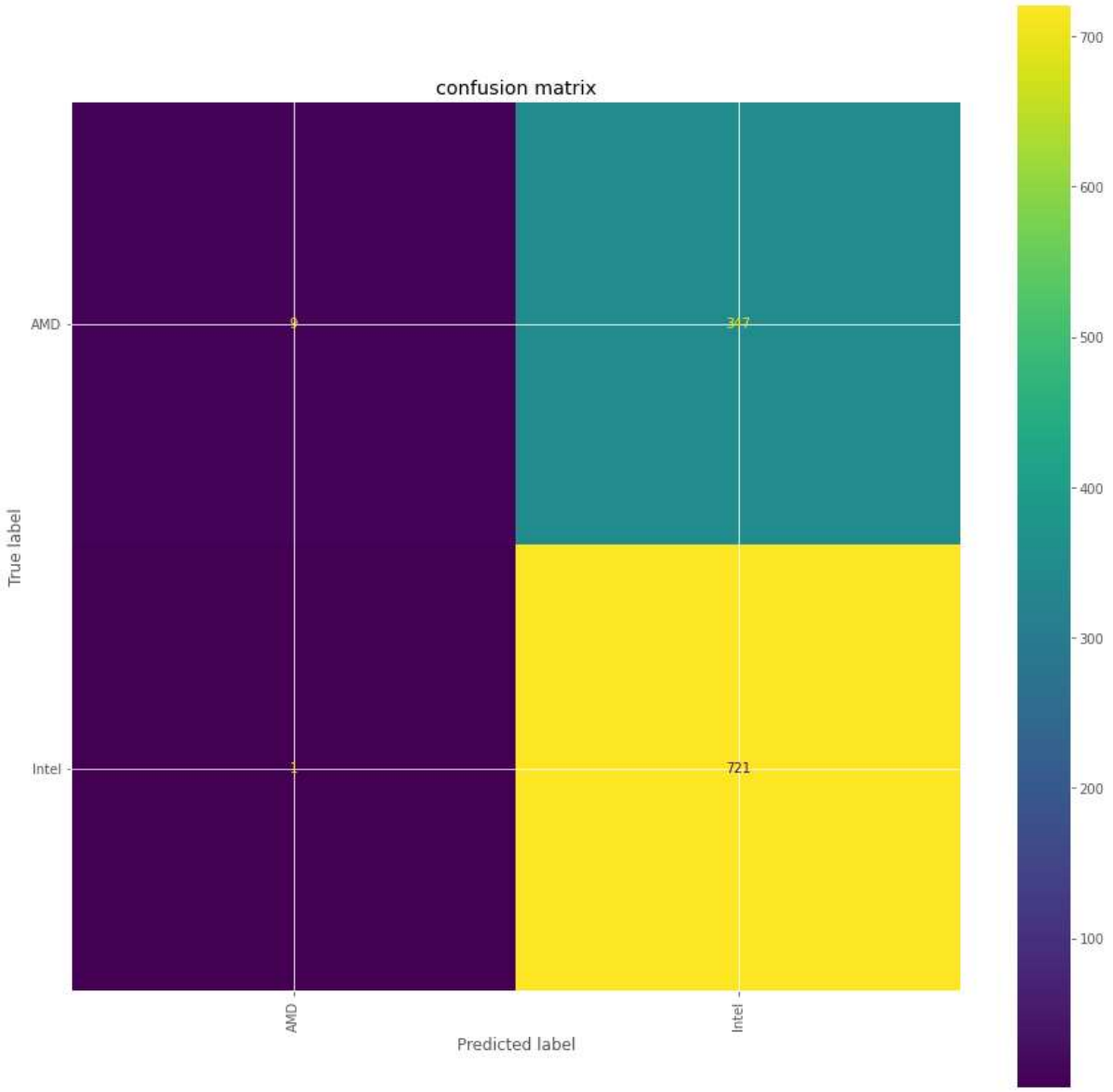
In [ ]:
```
#for i in X_train.select_dtypes('number').columns:
 #    sns.distplot(X_train[X_train['label'] == 0][i], label = 'label 0')
 #    sns.distplot(X_train[X_train['label'] == 1][i], label = 'label 1')
 #    plt.title(i)
 #    plt.legend()
#     plt.show()
```

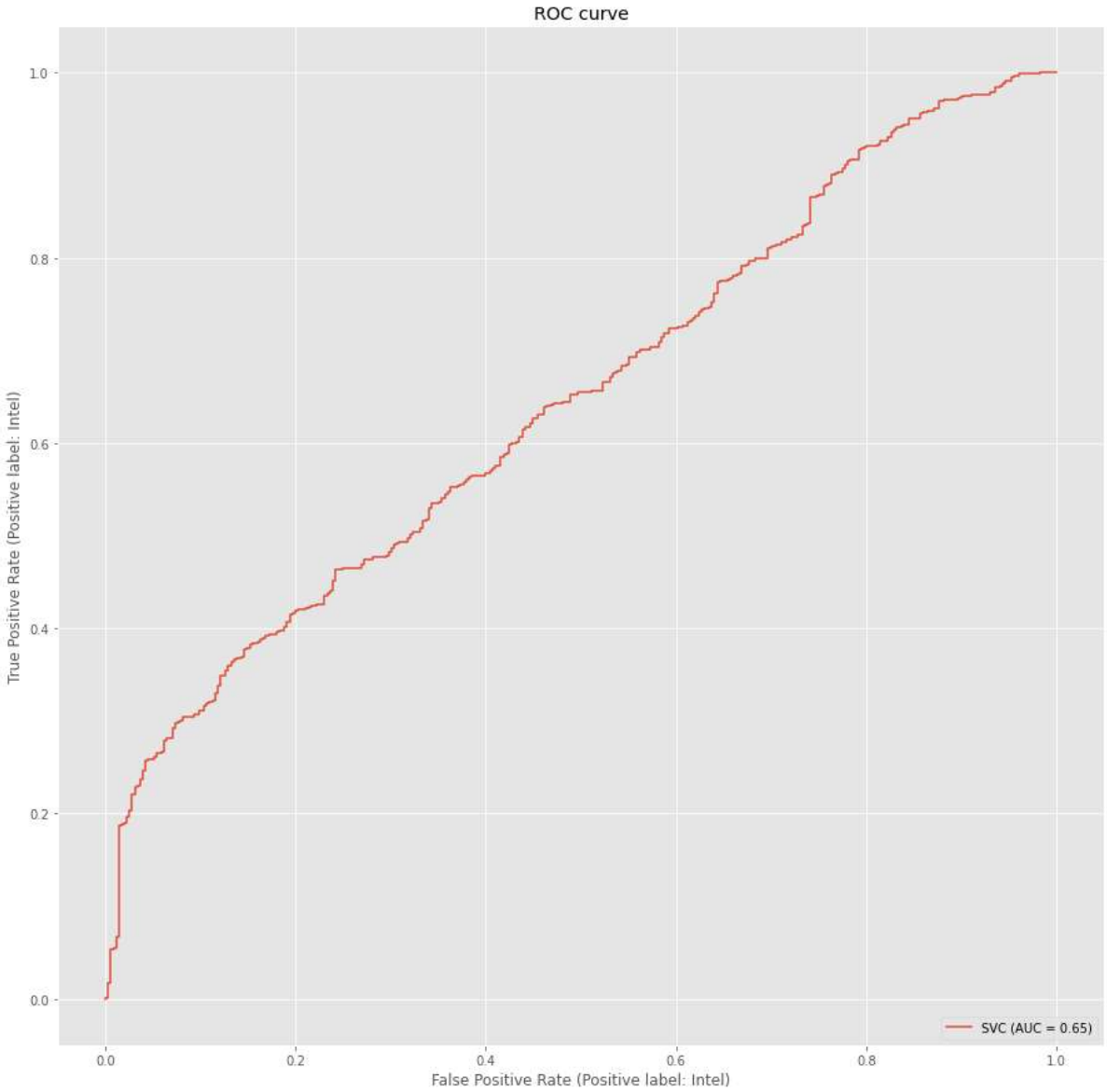
In [ ]:
```
#features.groupby('label').agg(['mean', 'median', 'std'])
```

In [ ]:
```
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
import warnings
with warnings.catch_warnings():
    clf = svm.SVC(kernel='linear', C=1, random_state=42)
    clf.fit(X_train,y_train)
    scores=clf.score(X_test,y_test)
    print("%0.2f train-test accuracy" % (scores.mean()))

    scores = cross_val_score(clf,X_train, y_train, cv=7)
    print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scor
    print(scores)
    #confusion matrix
```

```
    import matplotlib.pyplot as plt
    from sklearn.datasets import make_classification
    from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
    from sklearn.model_selection import train_test_split
    from sklearn.svm import SVC


    predictions = clf.predict(X_test)
    cm = confusion_matrix(y_test, predictions, labels=clf.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=clf.classes_)
    disp.plot()
    plt.xticks(rotation=90)
    plt.rcParams["figure.figsize"] = (15,15)
    plt.title('confusion matrix')
    plt.show()

    from sklearn.metrics import RocCurveDisplay
    RocCurveDisplay.from_estimator(clf, X_test, y_test)
    plt.title('ROC curve')
    plt.show()
```

```
0.68 train-test accuracy
0.66 accuracy with a standard deviation of 0.01
[0.65175719 0.66773163 0.66025641 0.66025641 0.65705128 0.6474359
 0.65384615]
```

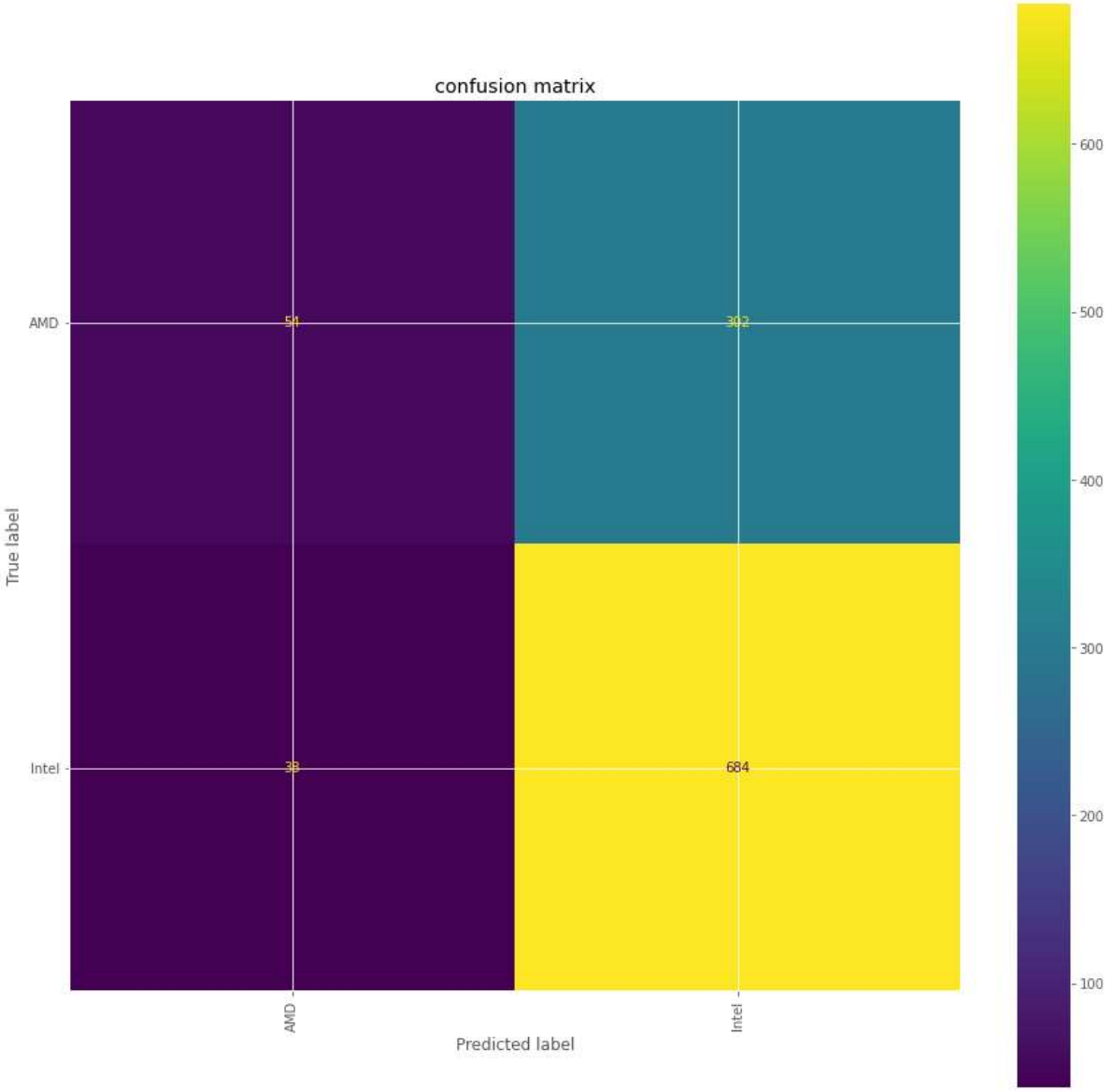


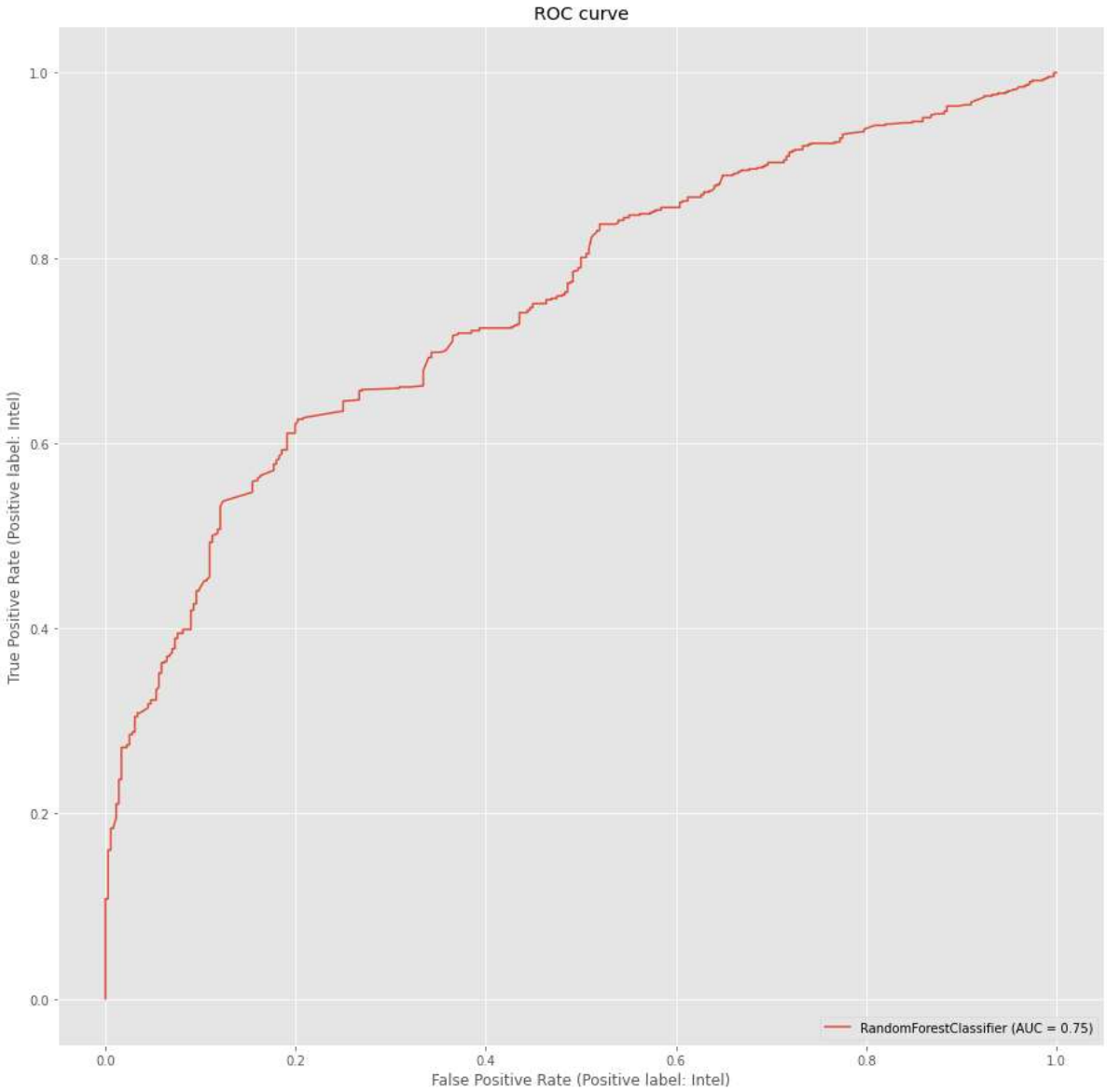


In [ ]:
```
#random forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
rf = RandomForestClassifier(max_depth=2, random_state=0)
rf.fit(X_train, y_train)
print(rf.score(X_test,y_test))
#confusion matrix
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC


predictions = rf.predict(X_test)
cm = confusion_matrix(y_test, predictions, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)

disp.plot()
plt.xticks(rotation=90)
plt.rcParams["figure.figsize"] = (15,15)
plt.title('confusion matrix')

plt.show()
from sklearn.metrics import RocCurveDisplay
```

```
RocCurveDisplay.from_estimator(rf, X_test, y_test)

plt.title('ROC curve')
plt.show()
```

0.6846011131725418



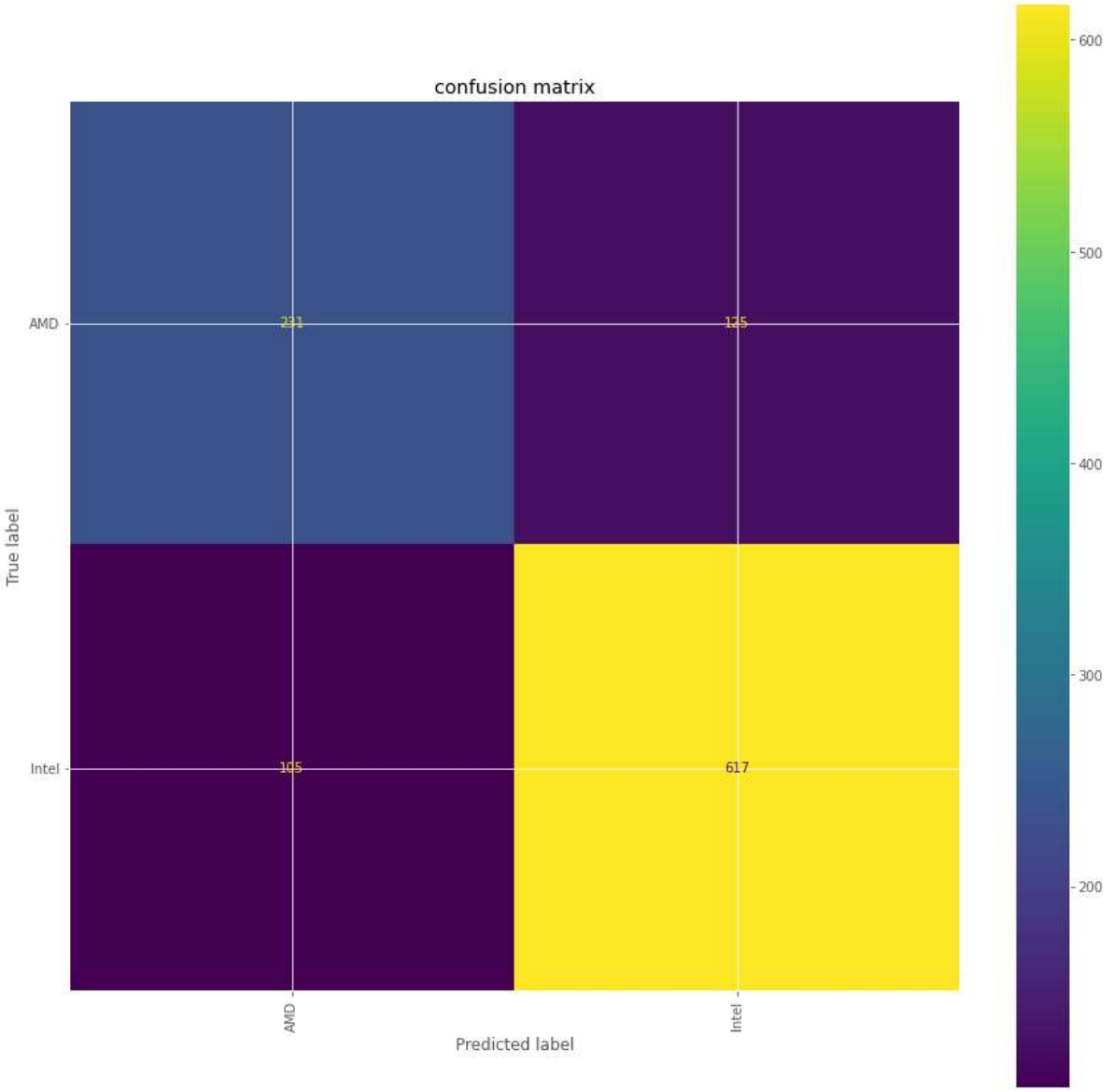


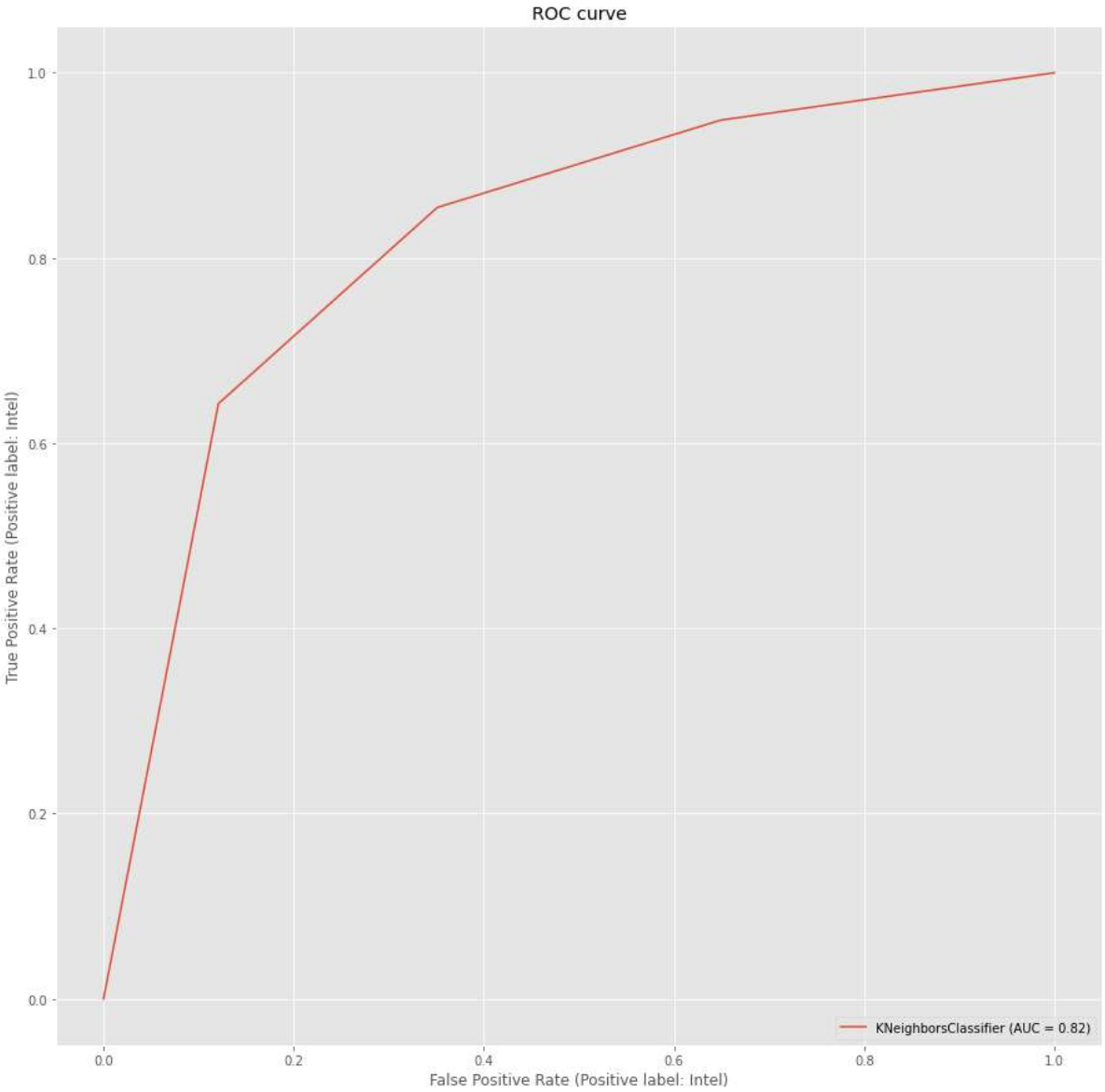
In [ ]:

```
#knn
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
neigh.score(X_test,y_test)
print(neigh.score(X_test,y_test))

#confusion matrix
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC


predictions = neigh.predict(X_test)
cm = confusion_matrix(y_test, predictions, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)

disp.plot()
plt.xticks(rotation=90)
plt.rcParams["figure.figsize"] = (15,15)
plt.title('confusion matrix')

plt.show()
```

```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_estimator(neigh, X_test, y_test)

plt.title('ROC curve')
plt.show()
```

0.7866419294990723



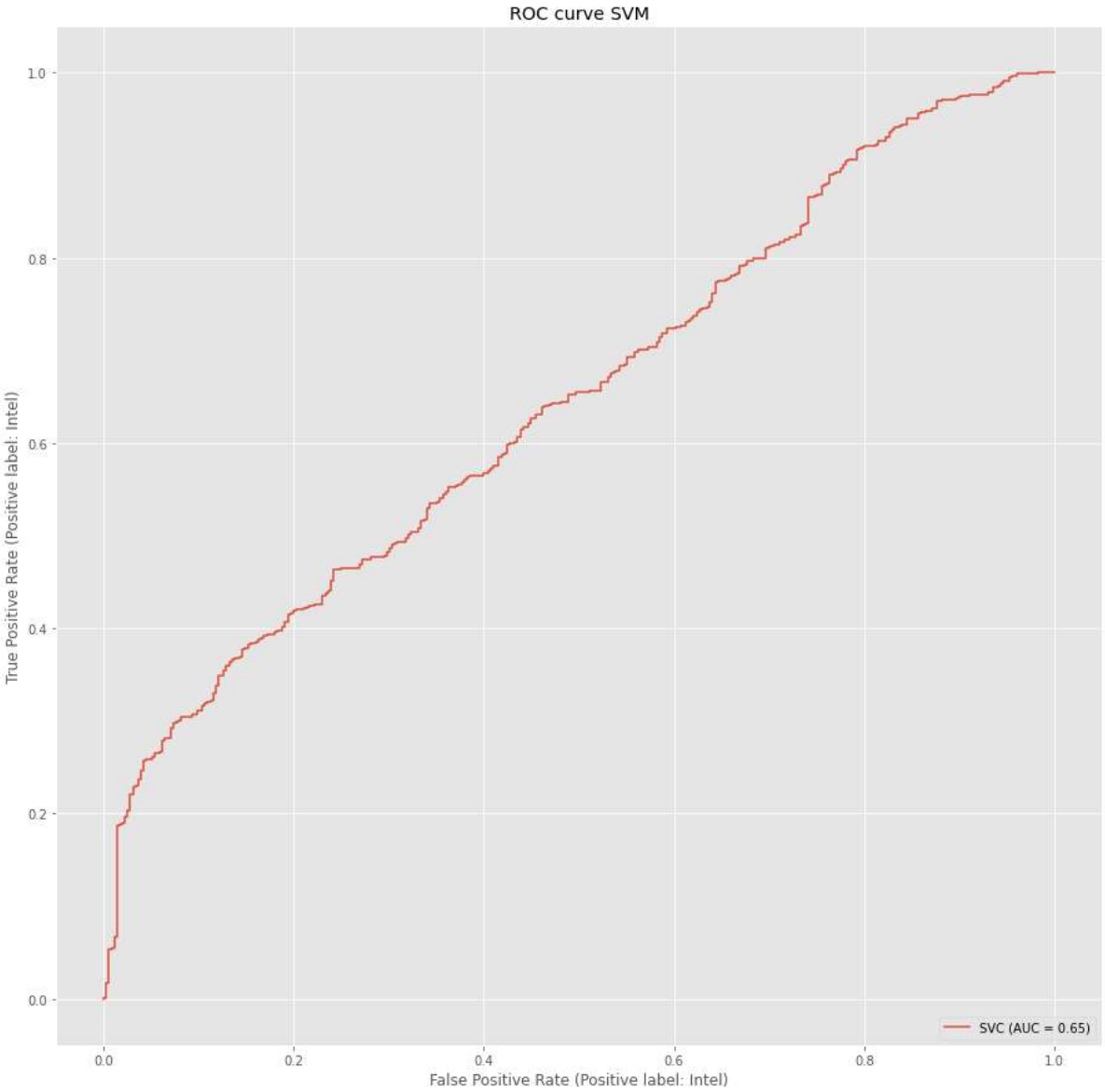


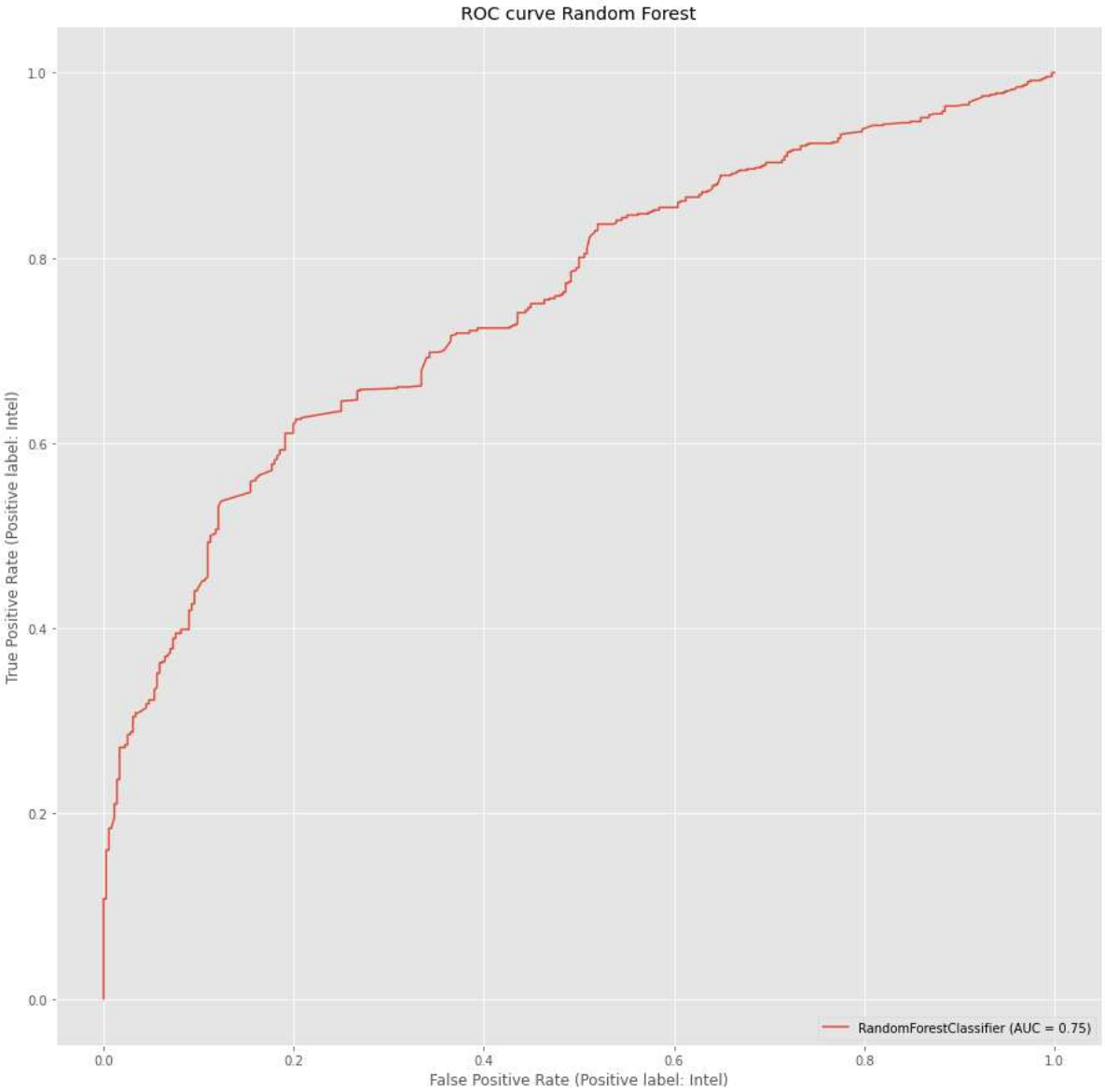
those knn is a clustering algorism, it is said wield to be used for classificaiton, but it as a classifier have a best outcome in this case. accuracy reached 78%.

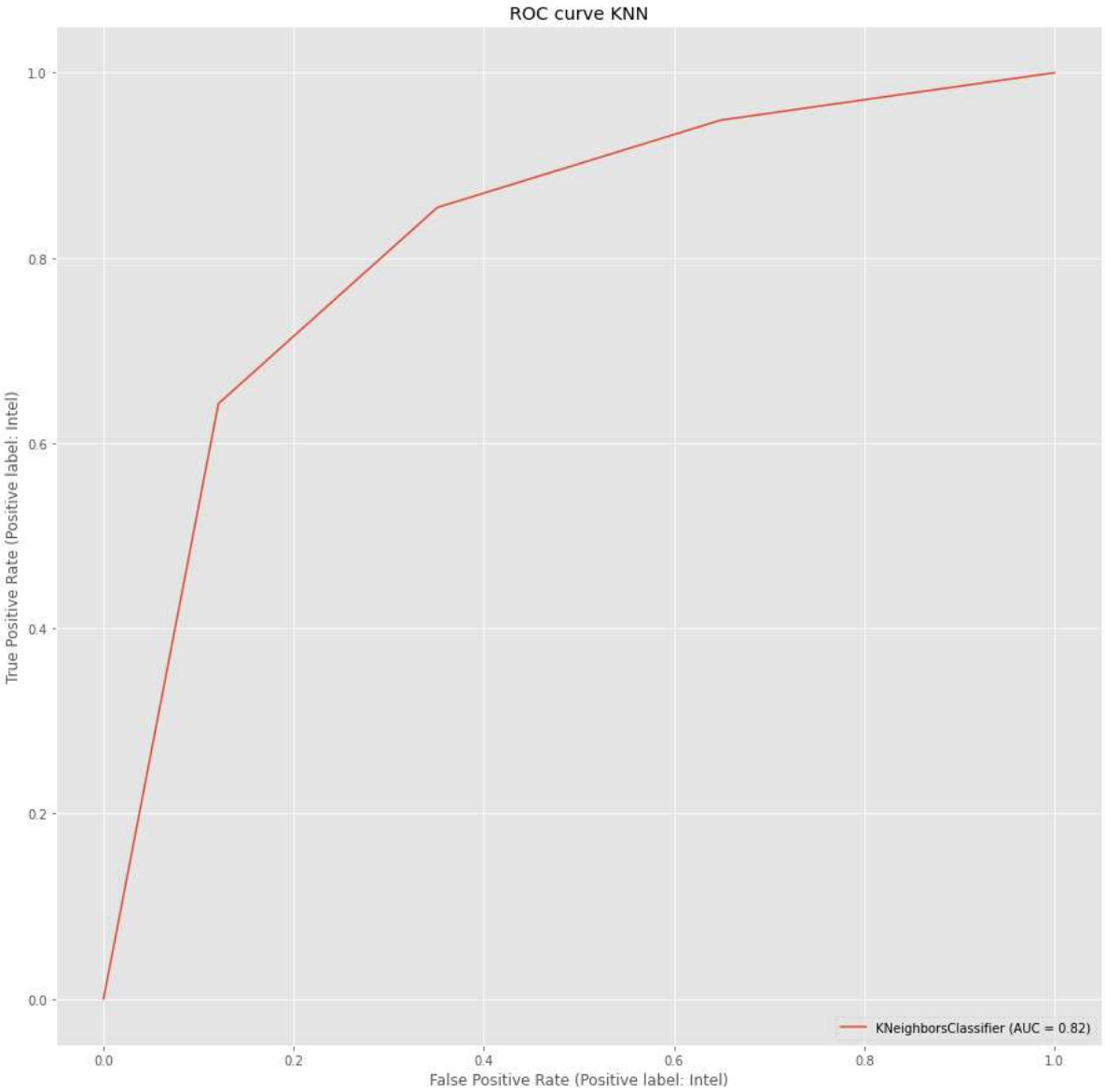
3. model evaluation

to compare the three models, we draw the Receiver Operating Characteristic Curve and the Area Under the curve are: svm:75% random forest: 75% knn: 82% This result also explain the accuracy we got above.

In [ ]:
```
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import roc_auc_score
RocCurveDisplay.from_estimator(clf, X_test, y_test)

plt.title('ROC curve SVM')
RocCurveDisplay.from_estimator(rf, X_test, y_test)
plt.title('ROC curve Random Forest')
RocCurveDisplay.from_estimator(neigh, X_test, y_test)
plt.title('ROC curve KNN')
plt.show()
```


ROC curve SVM
True Positive Rate (Positive label: Intel)
False Positive Rate (Positive label: Intel)
0.0
0.2
0.4
0.6
0.8
1.0
SVC (AUC = 0.65)


ROC curve Random Forest
True Positive Rate (Positive label: Intel)
False Positive Rate (Positive label: Intel)
0.0
0.2
0.4
0.6
0.8
1.0
RandomForestClassifier (AUC = 0.75)



after all the experiments, the result shows it is possible but difficult to pridict which brand a cpu is base on given data, from the following figures,we can also see the reasons.

In [ ]:
```
import seaborn as sns
sns.pairplot(df,hue='brand')
```

<seaborn.axisgrid.PairGrid at 0x244a36135e0>


price
cpuMark
cpuValue
threadMark
threadValue
TDP
powerPerf
cores
testDate
brand
Intel
AMD