In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
data=pd.read_csv('F:\A.S\Working\Material\Machinfy\Sessions\Session 16\Assignments\Exercise\census.csv',sep=',',
data.head()
```

Out[2]:

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	inco
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=5
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=5
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=5
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=5
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=5

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45222 entries, 0 to 45221
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              45222 non-null  int64 
 1   workclass        45222 non-null  object
 2   education_level  45222 non-null  object
 3   education-num    45222 non-null  int64 
 4   marital-status   45222 non-null  object
 5   occupation       45222 non-null  object
 6   relationship     45222 non-null  object
 7   race             45222 non-null  object
 8   sex              45222 non-null  object
 9   capital-gain     45222 non-null  int64 
 10  capital-loss     45222 non-null  int64 
 11  hours-per-week   45222 non-null  int64 
 12  native-country   45222 non-null  object
 13  income           45222 non-null  object
dtypes: int64(5), object(9)
memory usage: 4.8+ MB
```

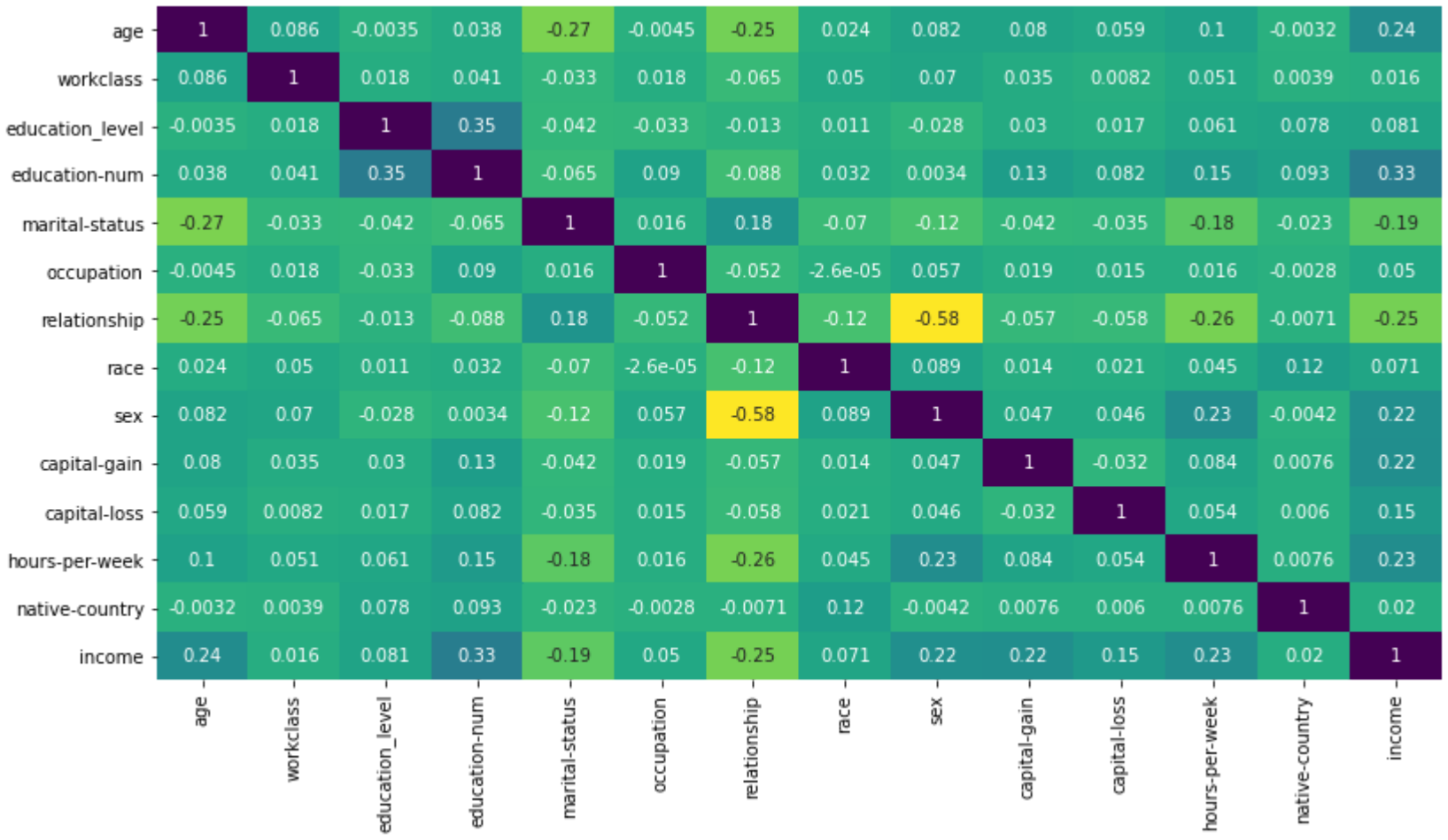
In [4]:
```
data["income"].value_counts()
```

Out[4]:
```
<=50K    34014
>50K     11208
Name: income, dtype: int64
```

In [5]:
```
from sklearn.preprocessing import LabelEncoder
Lb=LabelEncoder()
```

In [6]:
```
data["workclass"]=Lb.fit_transform(data["workclass"])
data["education_level"]=Lb.fit_transform(data["education_level"])
data["marital-status"]=Lb.fit_transform(data["marital-status"])
data["occupation"]=Lb.fit_transform(data["occupation"])
data["relationship"]=Lb.fit_transform(data["relationship"])
data["race"]=Lb.fit_transform(data["race"])
data["sex"]=Lb.fit_transform(data["sex"])
data["native-country"]=Lb.fit_transform(data["native-country"])
data["income"]=Lb.fit_transform(data["income"])
```

In [7]:
```
plt.figure(figsize=(13,7))
sns.heatmap(cbar=False,annot=True,data=data.corr(),cmap='viridis_r')
plt.show()
```



In [8]:
```
data.columns
```

Out[8]:
```
Index(['age', 'workclass', 'education_level', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income'],
      dtype='object')
```

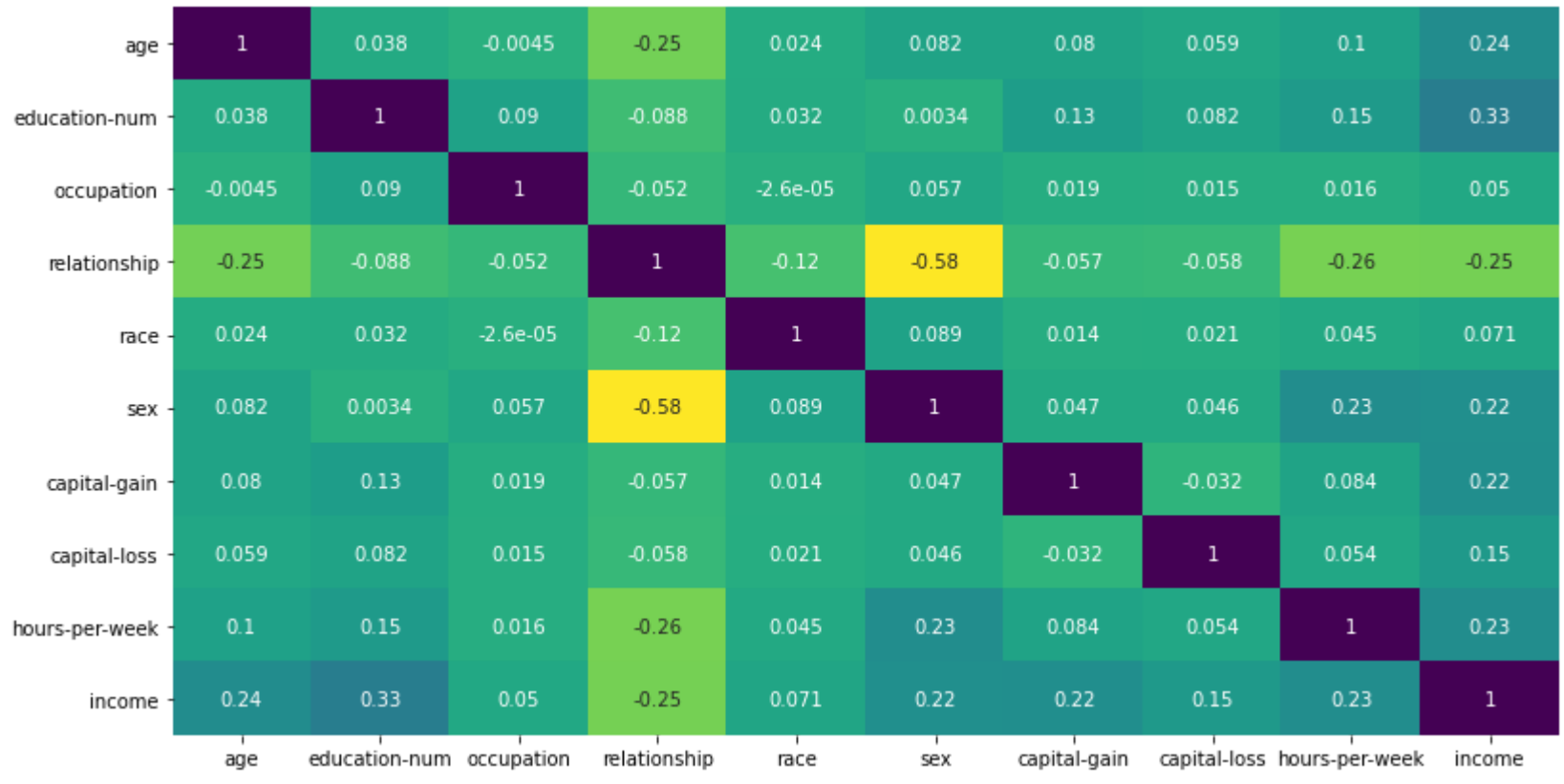
In [9]:
```
columns=['age','education-num', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week','income']
```

In [10]:
```
data=data[columns]
data.head()
```

Out[10]:

	age	education-num	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	income
0	39	13	0	1	4	1	2174	0	40	0
1	50	13	3	0	4	1	0	0	13	0
2	38	9	5	1	4	1	0	0	40	0
3	53	7	5	0	2	1	0	0	40	0
4	28	13	9	5	2	0	0	0	40	0

In [11]:
```
plt.figure(figsize=(13,7))
sns.heatmap(cbar=False,annot=True,data=data.corr(),cmap='viridis_r')
plt.show()
```

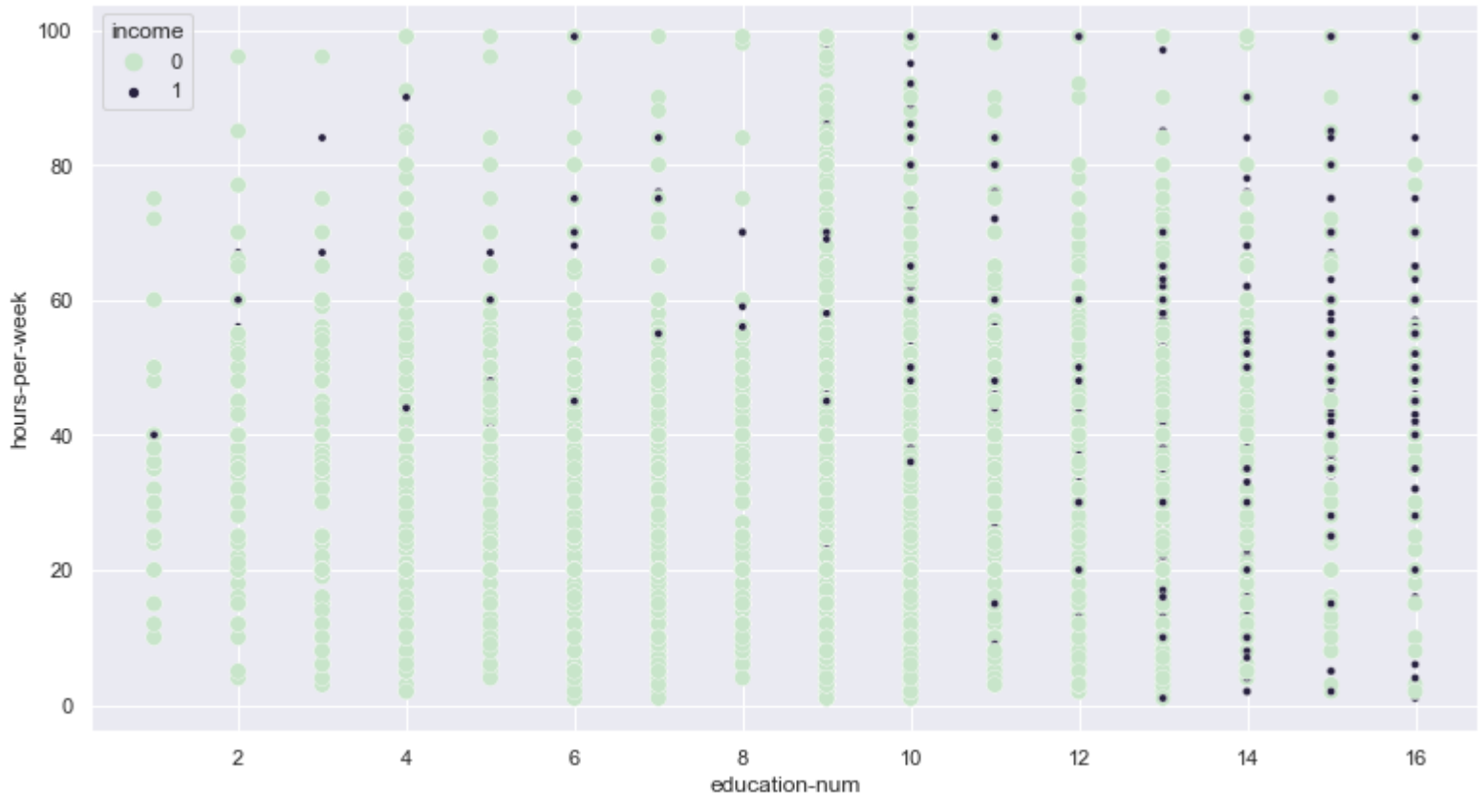


In [12]:
```
data.columns
```

Out[12]:
```
Index(['age', 'education-num', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'income'],
      dtype='object')
```

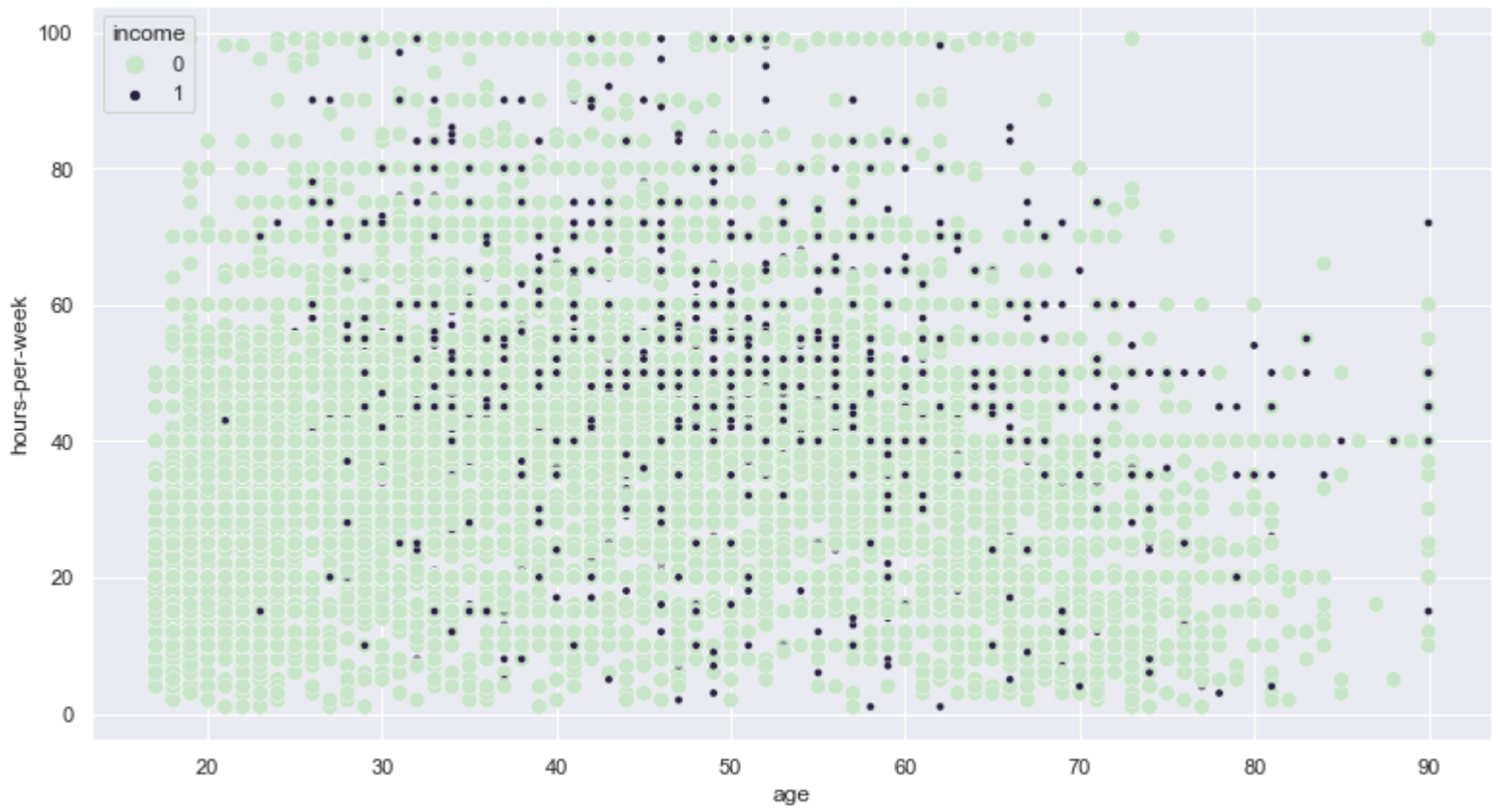
In [13]:
```
sns.set_theme(color_codes=False)
plt.figure(figsize=(13,7))
X=data['education-num']
Y=data["hours-per-week"]
sns.scatterplot(X,Y,hue='income',size='income',data=data,palette="ch:s=10.75,rot=-10.50")
plt.show()
```

```
C:\Users\ElmaDina\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following var
iables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [14]:
```
sns.set_theme(color_codes=False)
plt.figure(figsize=(13,7))
X=data['age']
Y=data["hours-per-week"]
sns.scatterplot(X,Y,hue='income',size='income',data=data,palette="ch:s=10.75,rot=-10.50")
plt.show()
```

```
C:\Users\ElmaDina\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following var
iables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Splitting the dataset into the Training set and Test set

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
data.columns
```

Out[16]:
```
Index(['age', 'education-num', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'income'],
      dtype='object')
```

In [17]:
```
x=data[['age','education-num','occupation','relationship','race','sex','capital-gain','capital-loss','hours-per-
y=data['income'].values
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

Feature Scaling

In [19]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [20]:
```
x_train_scaled=sc.fit_transform(x_train)
x_test_scaled=sc.fit_transform(x_test)
```

Modeling

SVM

Fitting SVM to the Training set

In [21]:
```
from sklearn.svm import SVC()
```

In [22]:
```
svm = SVC(kernel='sigmoid')
svm.fit(x_train,y_train)
```

Out[22]:
```
SVC(kernel='sigmoid')
```

Predicting the Test set

In [23]:
```
y_pred=svm.predict(x_test_scaled)
y_pred
```

Out[23]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Making the Confusion Matrix

In [24]:
```
from sklearn.metrics import confusion_matrix
```

In [25]:
```
cm=confusion_matrix(y_pred,y_test)
cm
```

Out[25]:
```
array([[11213,  3601],
       [    3,   107]], dtype=int64)
```

Accuracy

In [26]:
```
svm.score(x_train_scaled,y_train)
```

Out[26]:
```
0.7587959601293814
```

In [27]:
```
svm.score(x_test_scaled,y_test)
```

Out[27]:
```
0.7585097829000268
```

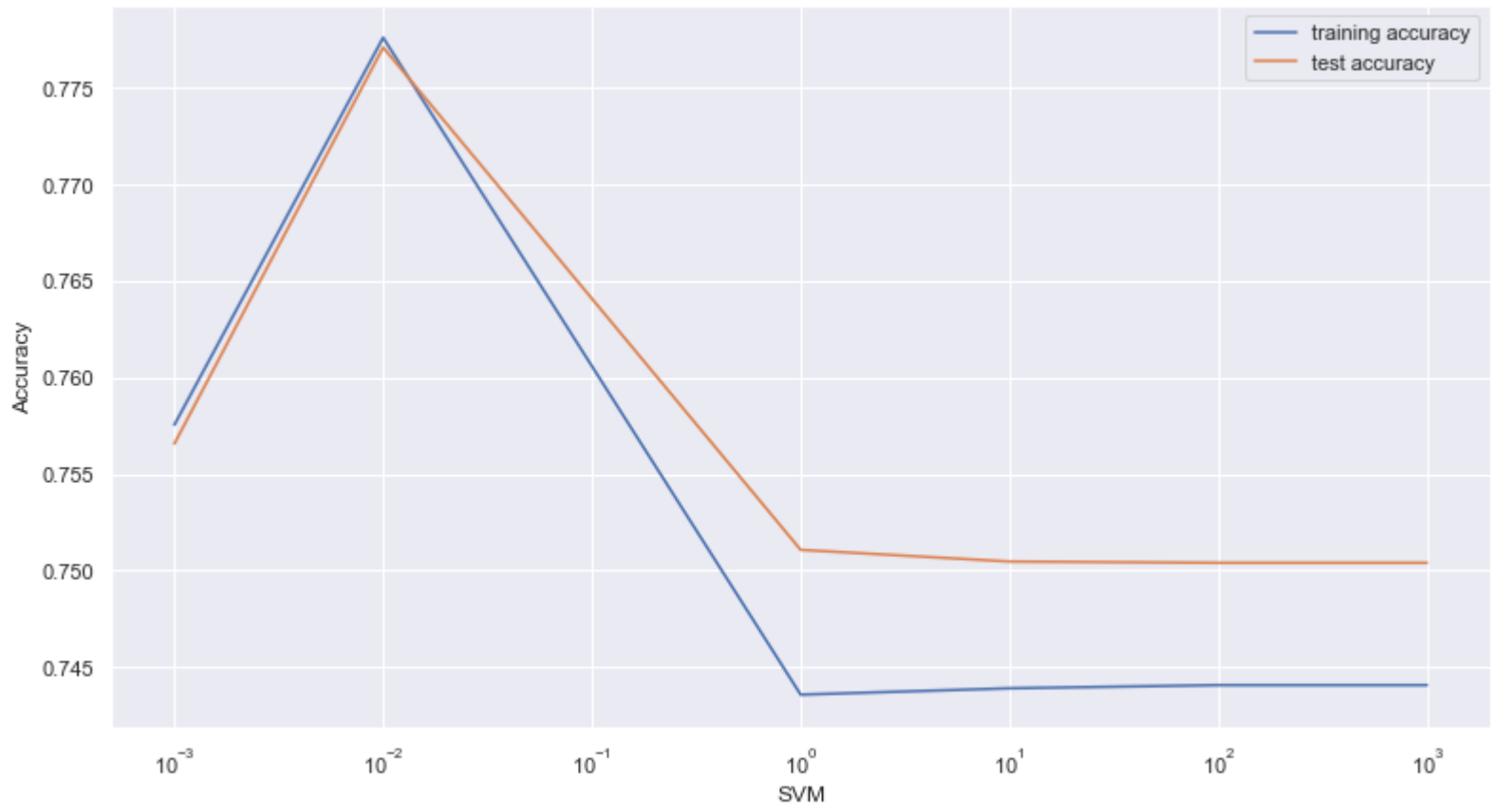
Performance

In [29]:
```
training_accuracy = []
test_accuracy = []
for c_value in [0.001,0.01,1,10,100,1000]:
    svm=SVC(kernel='sigmoid',C=c_value)
    svm.fit(x_train_scaled,y_train)
    training_accuracy.append(svm.score(x_train_scaled,y_train))
    test_accuracy.append(svm.score(x_test_scaled,y_test))

c_values=np.array(([0.001,0.01,1,10,100,1000]))
```

In [30]:
```
plt.figure(figsize=(13,7))
plt.semilogx(c_values, training_accuracy, label="training accuracy")
plt.semilogx(c_values, test_accuracy,label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("SVM")
plt.legend()
```

Out[30]:
```
<matplotlib.legend.Legend at 0x2141412d100>
```



Attributes

In [124]:
```
svm.support_
```

Out[124]:
```
array([    2,    11,    12, ..., 30218, 30240, 30257])
```

In [125]:
```
svm.support_vectors_
```

Out[125]:
```
array([[ 1.24387238,  0.34126385,  0.74968066, ...,  0.76777956,
        -0.21780005, -0.0782174 ],
       [-0.87141382,  1.50966961,  0.74968066, ..., -0.14507373,
        -0.21780005,  2.00364956],
       [ 1.62160206, -2.38501626, -0.74190225, ..., -0.14507373,
        -0.21780005,  1.17090277],
       ...,
       [ 1.54605613,  0.34126385,  0.74968066, ..., -0.14507373,
        -0.21780005,  0.75452938],
       [ 1.24387238, -0.43767333, -0.99049941, ..., -0.14507373,
        -0.21780005, -0.0782174 ],
       [-0.72032194,  0.34126385, -0.4933051 , ..., -0.14507373,
        -0.21780005,  0.25488131]])
```

In [127]:
```
svm.n_support_
```

Out[127]:
```
array([3881, 3880])
```

Modeling

Naive Bayes

Fitting Naive Bayes to the Training set

In [89]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

In [90]:
```
gnb.fit(x_train,y_train)
```

Out[90]:
```
GaussianNB()
```

Predicting the Test set

In [91]:
```
y_pred1=gnb.predict(x_test_scaled)
y_pred1
```

Out[91]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Making the Confusion Matrix

In [92]:
```
cm1=confusion_matrix(y_pred1,y_test)
cm1
```

Out[92]:
```
array([[11216,  3708],
       [    0,     0]], dtype=int64)
```

Accuracy

In [93]:
```
gnb.score(x_train_scaled,y_train)
```

Out[93]:
```
0.7524589081787577
```

In [94]:
```
gnb.score(x_test_scaled,y_test)
```

Out[94]:
```
0.7515411417850443
```

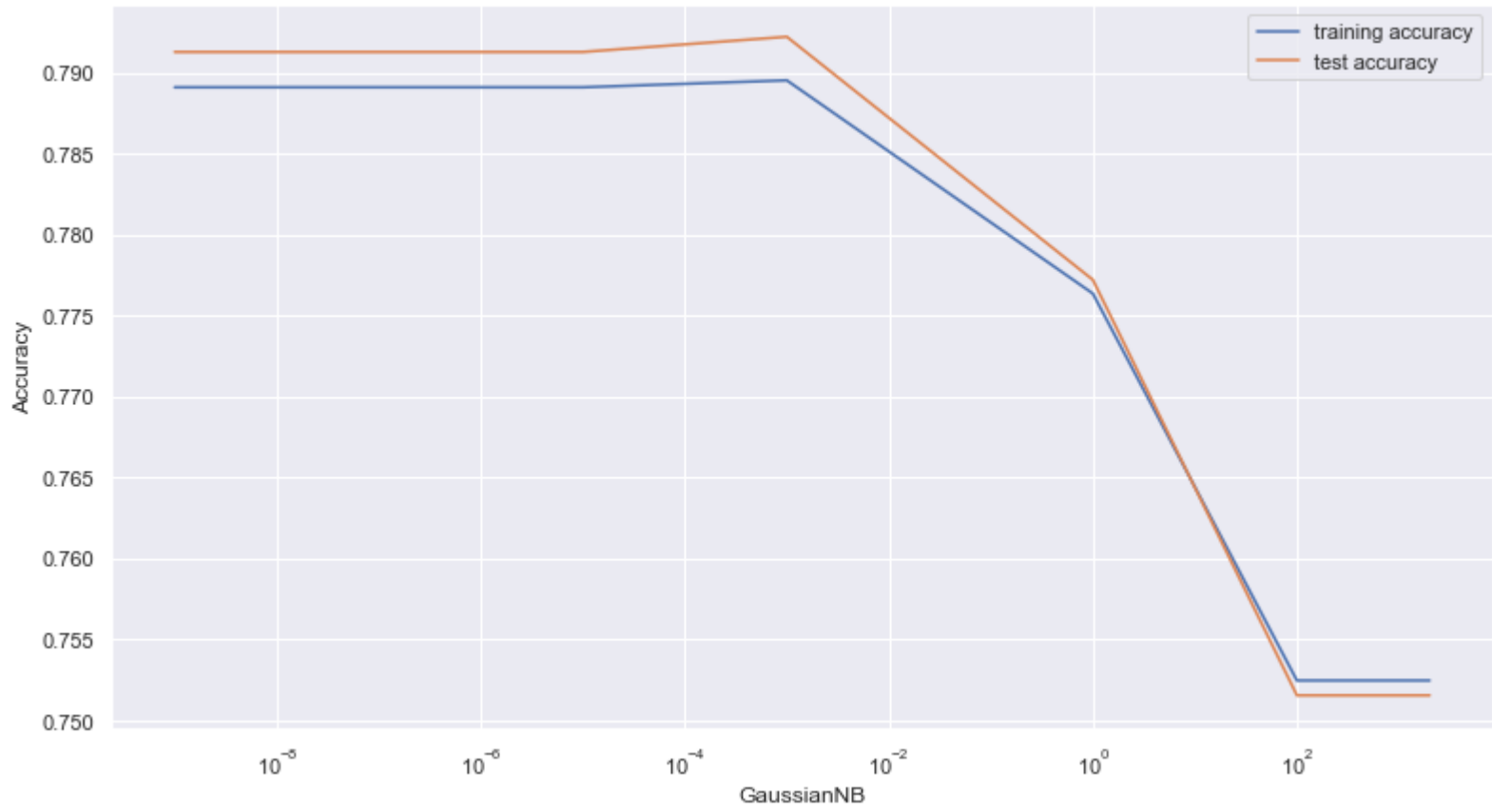
Performance

In [118]:
```
training_accuracy = []
test_accuracy = []
for var_value in [0.000000001,0.00001,0.001,1,100,1000,2000]:
    gnb= GaussianNB(var_smoothing=var_value)
    gnb.fit(x_train_scaled,y_train)
    training_accuracy.append(gnb.score(x_train_scaled,y_train))
    test_accuracy.append(gnb.score(x_test_scaled,y_test))

var_value=np.array(([0.000000001,0.00001,0.001,1,100,1000,2000]))
```

In [119]:
```
plt.figure(figsize=(13,7))
plt.semilogx(var_value, training_accuracy, label="training accuracy")
plt.semilogx(var_value, test_accuracy,label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("GaussianNB")
plt.legend()
```

Out[119]: <matplotlib.legend.Legend at 0x21425547520>



Attributes

In [120]:
```
gnb.class_count_
```

Out[120]: array([22798., 7500.])

In [121]:
```
gnb.class_prior_
```

Out[121]: array([0.75245891, 0.24754109])

In [123]:
```
gnb.epsilon_
```

Out[123]: 2000.000000001045

Modeling

Logistic Regression

Fitting Logistic Regression to the Training set

In [160]:
```
from sklearn.linear_model import LogisticRegression
logisticregr=LogisticRegression()
```

In [161]:
```
logisticregr.fit(x_train_scaled,y_train)
```

Out[161]: LogisticRegression()

Predicting the Test set

In [162]:
```
y_pred2=logisticregr.predict(x_test_scaled)
y_pred2
```

Out[162]: array([0, 0, 0, ..., 0, 1, 0])

Making the Confusion Matrix

In [163]:
```
cm2=confusion_matrix(y_pred2,y_test)
cm2
```

Out[163]:
```
array([[10622,  2044],
       [  594,  1664]], dtype=int64)
```

Accuracy

In [164]:
```
logisticregr.score(x_train_scaled,y_train)
```

Out[164]:
```
0.8190639646181266
```

In [165]:
```
logisticregr.score(x_test_scaled,y_test)
```

Out[165]:
```
0.8232377378718843
```

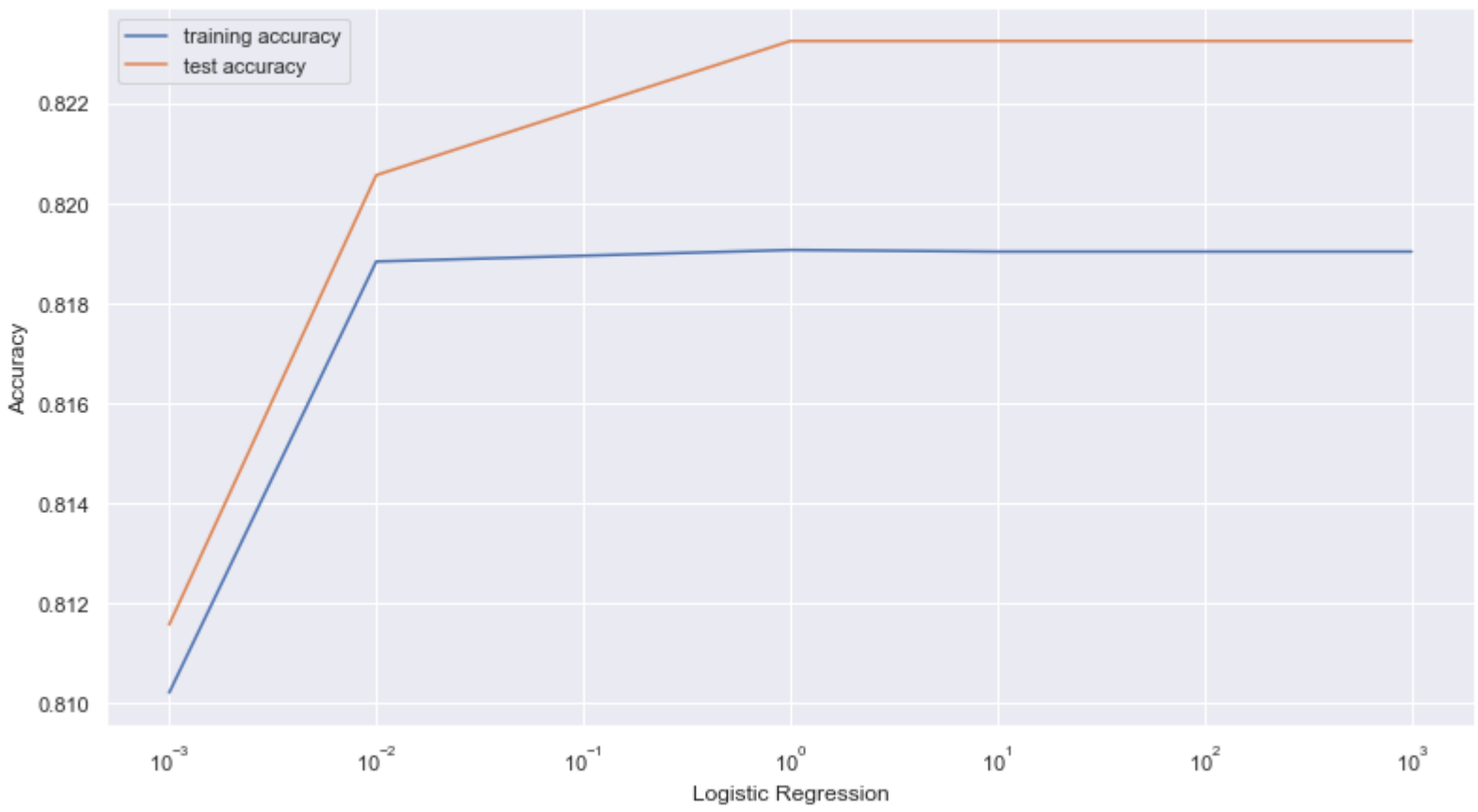
Performance

In [166]:
```
training_accuracy = []
test_accuracy = []
for c_value in [0.001,0.01,1,10,100,1000]:
    logisticregr=LogisticRegression(C=c_value)
    logisticregr.fit(x_train_scaled,y_train)
    training_accuracy.append(logisticregr.score(x_train_scaled,y_train))
    test_accuracy.append(logisticregr.score(x_test_scaled,y_test))

c_values=np.array(([0.001,0.01,1,10,100,1000]))
```

In [167]:
```
plt.figure(figsize=(13,7))
plt.semilogx(c_values, training_accuracy, label="training accuracy")
plt.semilogx(c_values, test_accuracy,label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Logistic Regression")
plt.legend()
```

Out[167]:
```
<matplotlib.legend.Legend at 0x2142602b460>
```



Modeling

KNN CLF

Fitting KNN CLF to the Training set

In [178]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_c = KNeighborsClassifier(n_neighbors=20)
```

In [179]:
```
knn_c.fit(x_train_scaled,y_train)
```

Out[179]:
```
KNeighborsClassifier(n_neighbors=20)
```

Predicting the Test set

In [180]:
```
y_pred3=knn_c.predict(x_test_scaled)
y_pred3
```

Out[180]:
```
array([0, 0, 0, ..., 1, 1, 0])
```

Making the Confusion Matrix

In [181]:
```
cm3=confusion_matrix(y_pred3,y_test)
cm3
```

Out[181]:
```
array([[10387,  1559],
       [  829,  2149]], dtype=int64)
```

Accuracy

In [182]:
```
knn_c.score(x_train_scaled,y_train)
```

Out[182]:
```
0.852828569542544
```

In [183]:
```
knn_c.score(x_test_scaled,y_test)
```

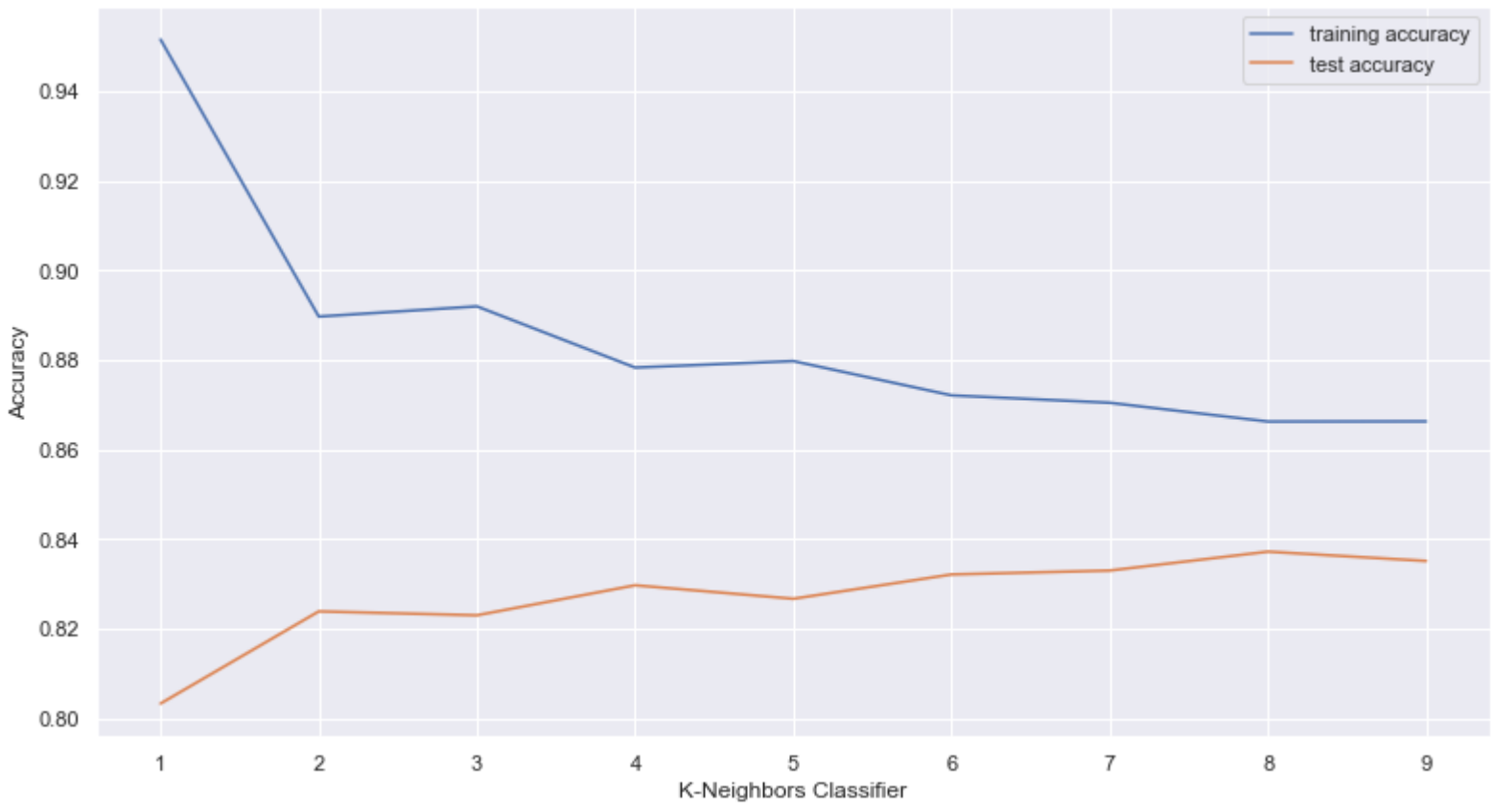
Out[183]:
```
0.8399892790136693
```

Performance

In [184]:
```
training_accuracy = []
test_accuracy = []
n=range(1, 10)
for n_neighbors in n:
    knn_c=KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_c.fit(x_train_scaled,y_train)
    training_accuracy.append(knn_c.score(x_train_scaled,y_train))
    test_accuracy.append(knn_c.score(x_test_scaled,y_test))
```

In [185]:
```
plt.figure(figsize=(13,7))
plt.plot(n, training_accuracy, label="training accuracy")
plt.plot(n, test_accuracy,label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("K-Neighbors Classifier")
plt.legend()
```

Out[185]:
```
<matplotlib.legend.Legend at 0x214268a1910>
```



In [ ]:
```
```