Lending Environment Simulation Model & Lender Evaluation Tool

Data Exploration

In [5]:
```
import glob
import os
import numpy as np
import pandas as pd
import category_maps as maps
import dictionaries as dc
import data_processor as preproc


pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

input_path = "../../data/processed/clean_data.csv"
paths = glob.glob(input_path)
len(paths)
```

Out[5]: 1

In [6]:
```
# read all *.csv files from the input path
# drop the headers and replace them with the sequential numbers
#df = pd.concat([pd.read_csv(f,names=range(0,38),header=0,low_memory=False) for f in pat
hs],ignore_index = True)
#df.shape,df.columns
```

In [7]:
```
df = pd.read_csv(input_path)
df.describe()
```

Out[7]:

	2	3	4	5	6	7	8	
count	29383.000000	29383.000000	29383.000000	29383.000000	29383.000000	29383.000000	29383.000000	2
mean	1.690195	2.068577	1.901201	2.156791	1.799136	1.822653	1.827349	
std	0.617315	0.613887	0.642582	0.604722	0.637219	0.630624	0.639422	
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	
25%	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000	1.000000	
50%	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	
75%	2.000000	2.000000	2.000000	3.000000	2.000000	2.000000	2.000000	
max	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	

In [8]:
```
df.head()
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	18	19	20	21	22	23	24	25	2
0	6-23-2019 5:42:51	N24S64b12	1	1	1	2	1	1	1	1	5	2	2	2	1	3	2	2	3	3	3	2	3	4	
1	6-23-2019 10:57:52	AC9S64a1	1	1	2	1	1	1	1	1	3	3	5	2	2	3	2	2	2	2	4	5	2	3	
2	6-23-2019 11:59:37	D16S64a26	2	2	2	2	2	2	2	2	2	2	9	5	2	3	2	2	2	2	2	2	2	3	
3	6-23-2019 20:40:41	AC9S64a2	2	1	3	1	3	2	2	2	2	6	5	4	2	3	2	2	1	2	3	3	3	3	
4	6-23-2019 20:41:27	AC9S64a3	2	1	3	2	3	2	3	2	2	5	5	3	2	3	2	1	3	3	2	3	2	3	

Feature Selection

In [9]:
```
from sklearn.feature_selection import SelectKBest, chi2

data = df.iloc[:,2:36]  #do not include date and location
c = pd.DataFrame.from_dict([dc.columns]).T
columns = pd.concat([c[2:15],c[16:17],c[18:39]])
credit_score = df['credit_score_category']
lender_score = df['lender_score_category']
```

Univariate Selection

Credit Score

In [10]:
```
bestfeatures = SelectKBest(chi2, k="all")
fit = bestfeatures.fit(data,credit_score)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(data.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Feature','Score']
print(featureScores.nlargest(30,'Score'))
```

```
    Feature        Score
20       24  4749.095968
14       18  1077.694415
22       26   727.163027
18       22   536.273308
15       19   512.080280
16       20   441.736339
29       33   360.285654
19       23   351.023573
12       14   301.560537
17       21   218.304598
13       16   197.034503
21       25   180.847240
9        11   103.303475
10       12    75.244389
25       29    63.378532
0         2    55.152635
8        10    39.545178
27       31    35.681715
26       30    31.481848
28       32    25.524501
33       37    24.675374
4         6    21.780553
31       35    19.450626
23       27    16.123902
11       13    15.055737
3         5    14.081732
6         8    13.702533
24       28    11.492451
1         3    11.432025
5         7    10.991739
```

Lender Score

In [11]:
```python
bestfeatures = SelectKBest(chi2, k="all")
fit = bestfeatures.fit(data,lender_score)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(data.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Feature','Score']
print(featureScores.nlargest(30,'Score'))
```

```
   Feature        Score
20      24  6667.831316
18      22  3588.478658
19      23  3339.377271
14      18  2014.330110
13      16   595.871957
21      25   555.595727
15      19   460.156830
22      26   444.852329
16      20   415.174349
12      14   174.179424
17      21   112.948018
33      37    91.901937
24      28    78.772595
23      27    76.693398
2        4    56.490698
9       11    44.332558
6        8    42.945995
28      32    39.221475
0        2    38.570787
31      35    37.263106
29      33    28.177739
4        6    26.136030
5        7    24.223802
1        3    22.600101
27      31    11.417786
11      13    11.361916
7        9     9.809677
26      30     9.362339
25      29     9.164333
3        5     7.823124
```

Feature Importance

Credit Score

In [12]:
```python
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier(n_estimators=100)
model.fit(data,credit_score)

#print(model.feature_importances_) #use inbuilt class feature_importances of tree based
 classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=data.columns)
_ = plt.figure(figsize=(23,10))
_ = feat_importances.nlargest(30).plot(kind='barh')
```

Credit Score Important Features

24,26,22,18,20,23,19,21,25,16

```
24    6684.282031
22    3579.344815
23    3308.826808
18    2021.701786
16     568.578714
25     555.864421
19     472.875751
26     447.615584
20     411.577992
36     196.022728
14     172.512950
30     164.043637
31     141.347621
21     114.004620
```

24,26,22,18,20,23,19,16,36,25

Lender Score

In [13]:
```
model = ExtraTreesClassifier(n_estimators=100)
model.fit(data,lender_score)

feat_importances = pd.Series(model.feature_importances_, index=data.columns)
_ = plt.figure(figsize=(23,10))
_ = feat_importances.nlargest(30).plot(kind='barh')
```

Lender Score Most Important Features

24,18,22,20,23,26,19,25,16

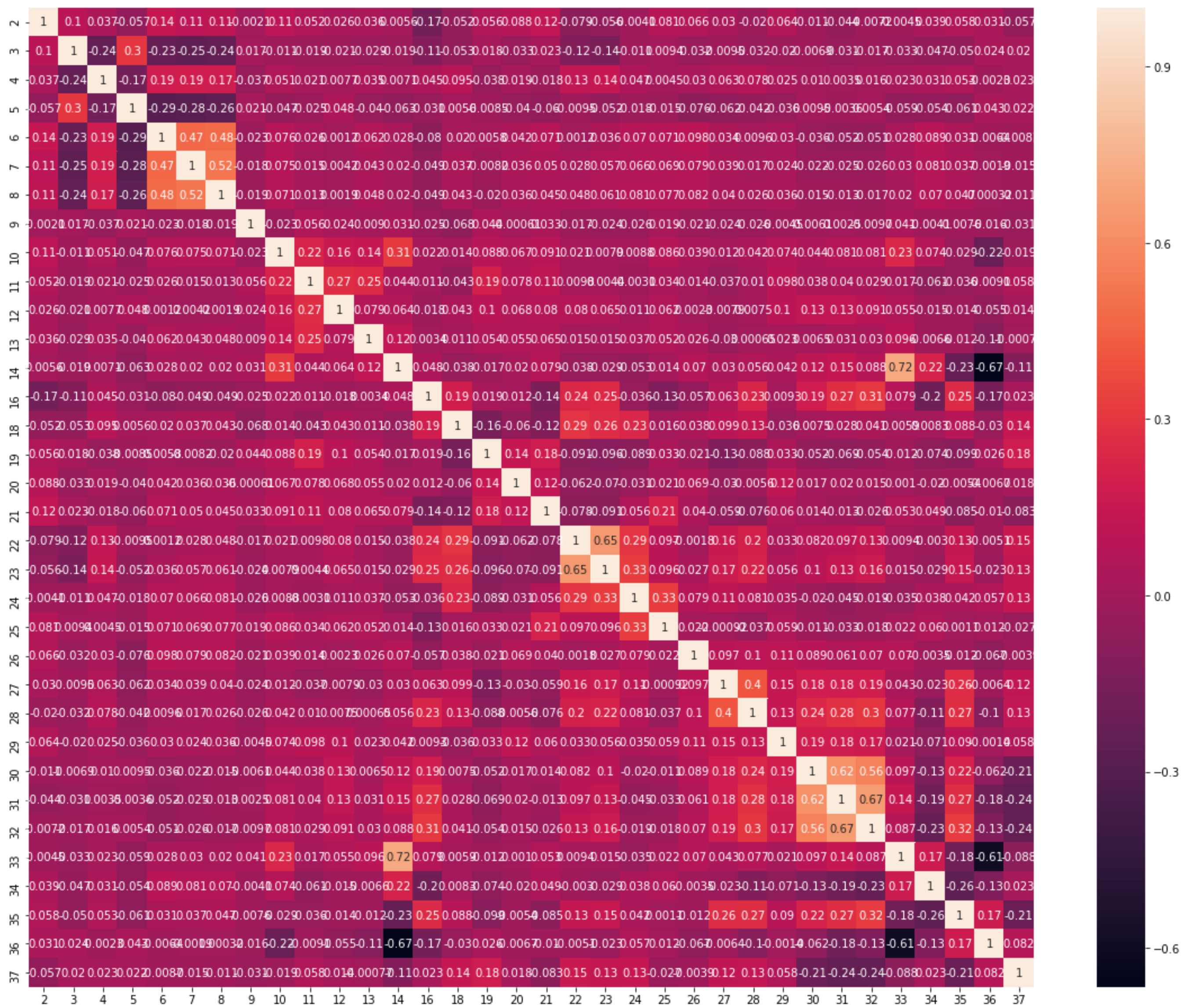
```
    24  6667.831316
    22  3588.478658
    23  3339.377271
    18  2014.330110
    16   595.871957
    25   555.595727
    19   460.156830
    26   444.852329
    20   415.174349
    14   174.179424
    21   112.948018
```

24,18,22,23,20,26,19,25,16,14,21

Feature Correlation Matrix

In [14]:

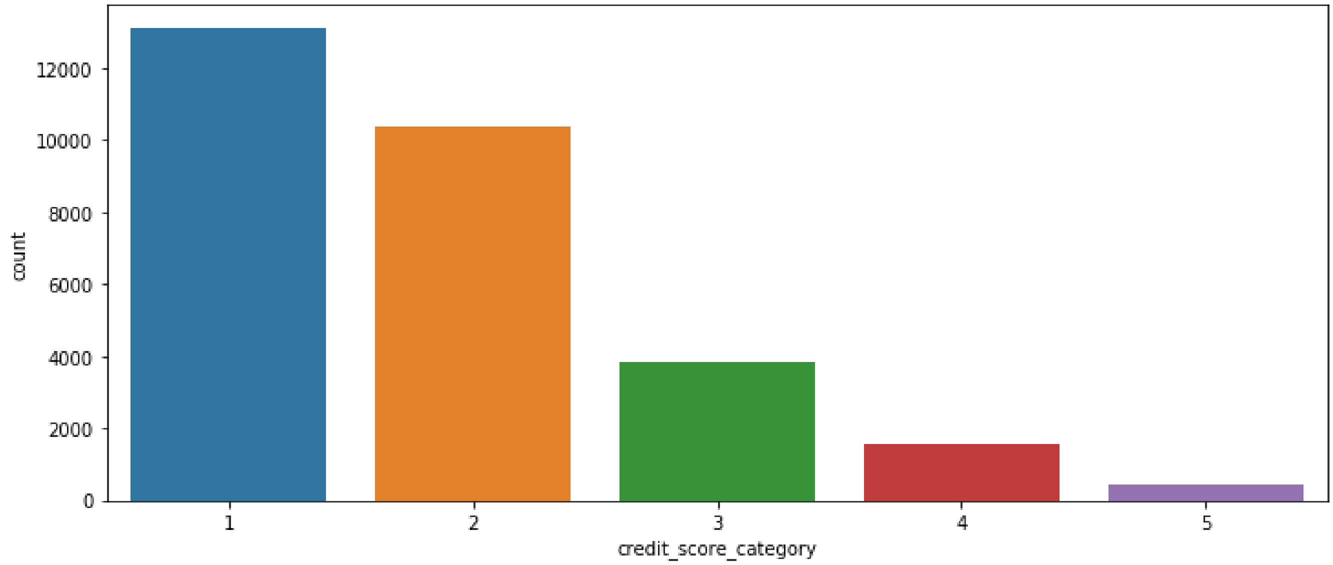
```python
import seaborn as sns

#get correlations of each features in dataset
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,16))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True)
```



Dataset Balance

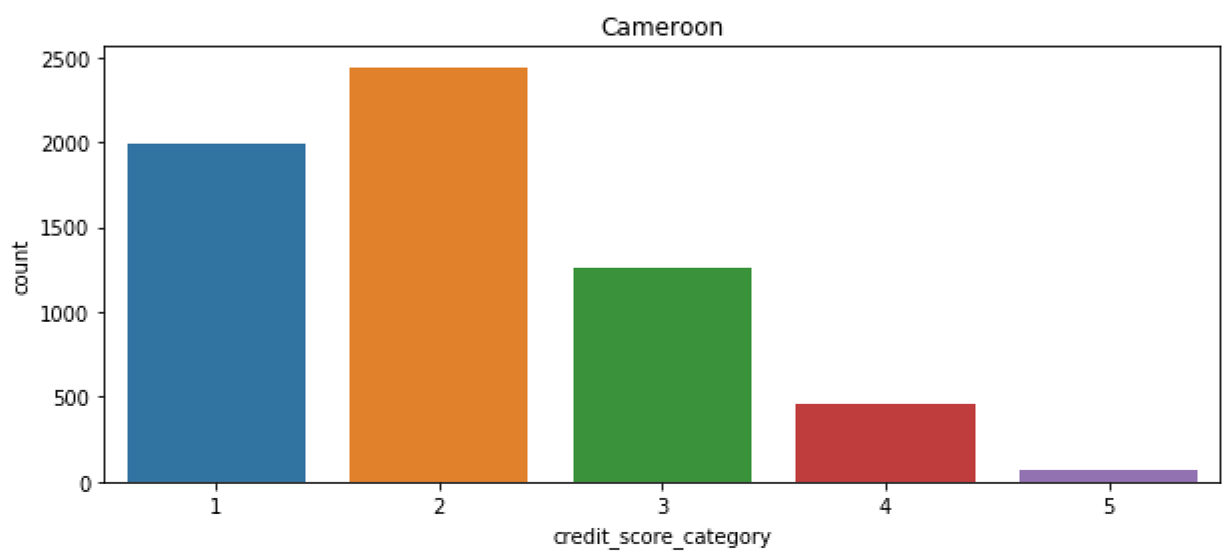
Credit Score Categories

In [15]:
```
plt.figure(figsize=(12,5))
sns.countplot(x=df['credit_score_category'], log=False);
```

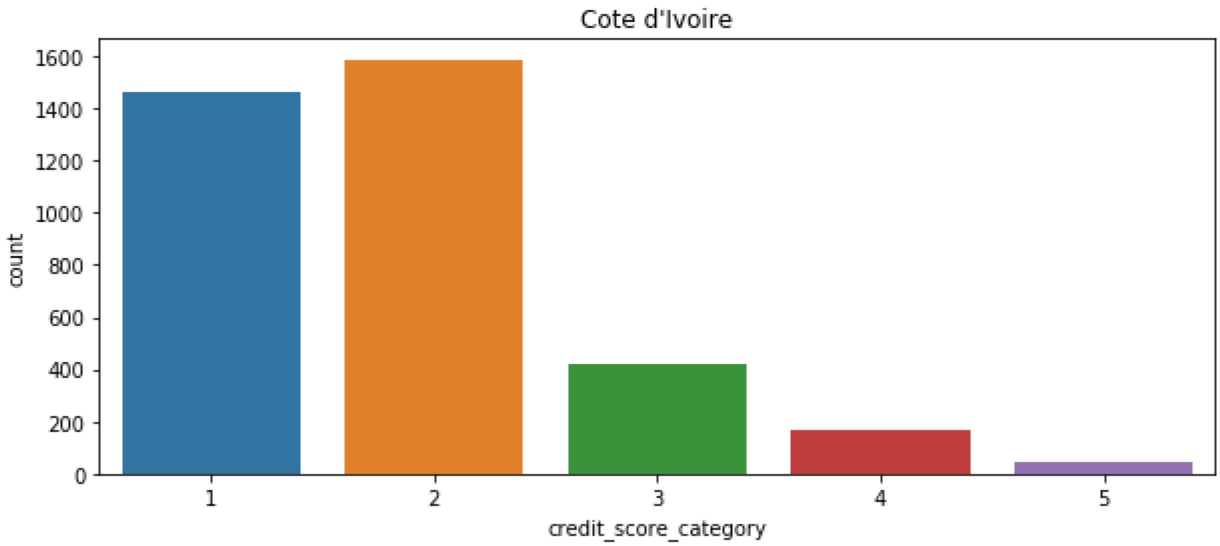


In [16]:
```
cameroon = df[df['16'] == 1]
cote = df[df['16'] == 2]
ghana = df[df['16'] == 3]
kenya = df[df['16'] == 4]
nigeria = df[df['16'] == 5]
south_africa = df[df['16'] == 6]
tanzania = df[df['16'] == 7]
```

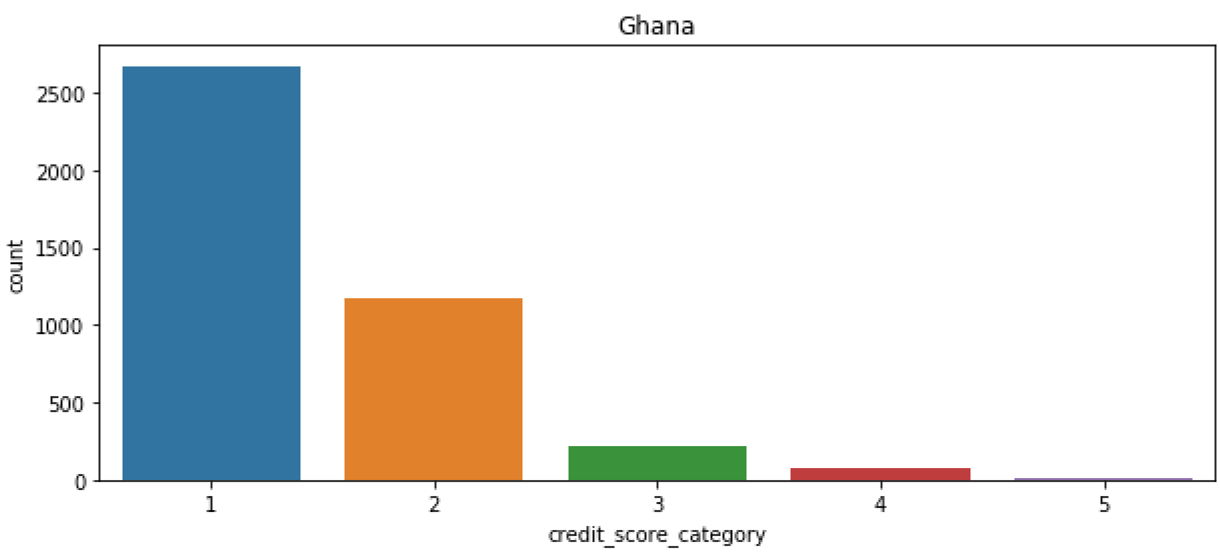
In [17]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=cameroon['credit_score_category']);
_ = plt.title("Cameroon")
```



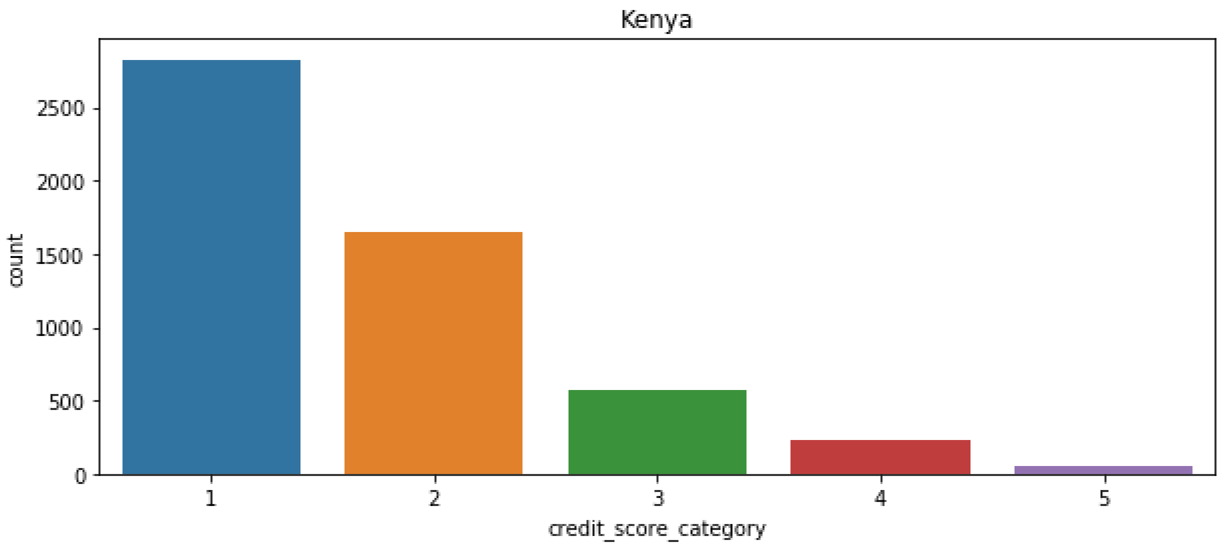
In [18]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=cote['credit_score_category']);
_ = plt.title("Cote d'Ivoire")
```



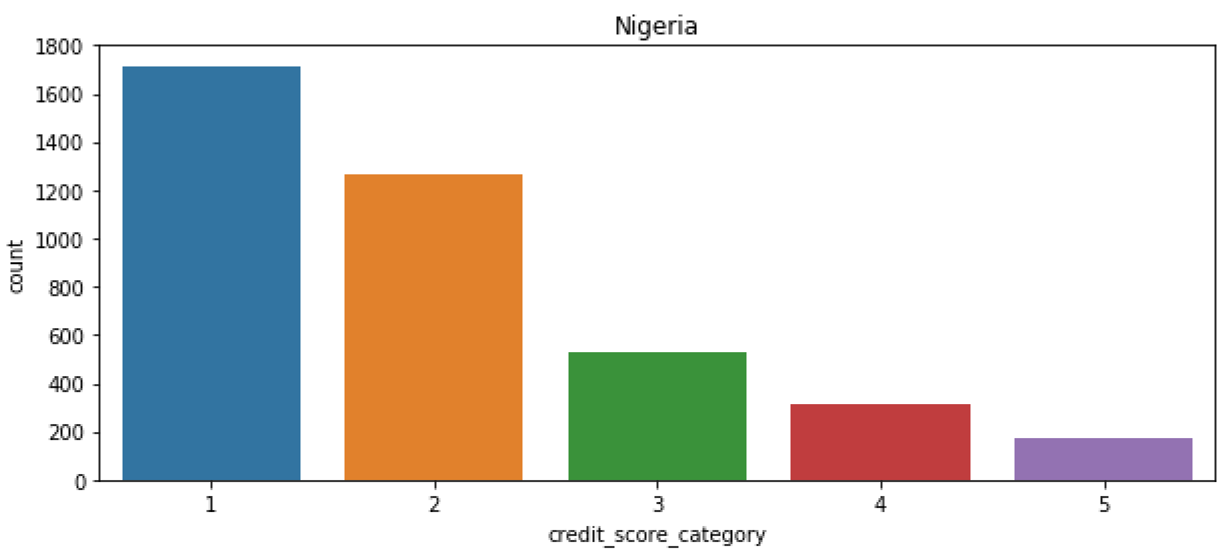
In [19]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=ghana['credit_score_category']);
_ = plt.title("Ghana")
```



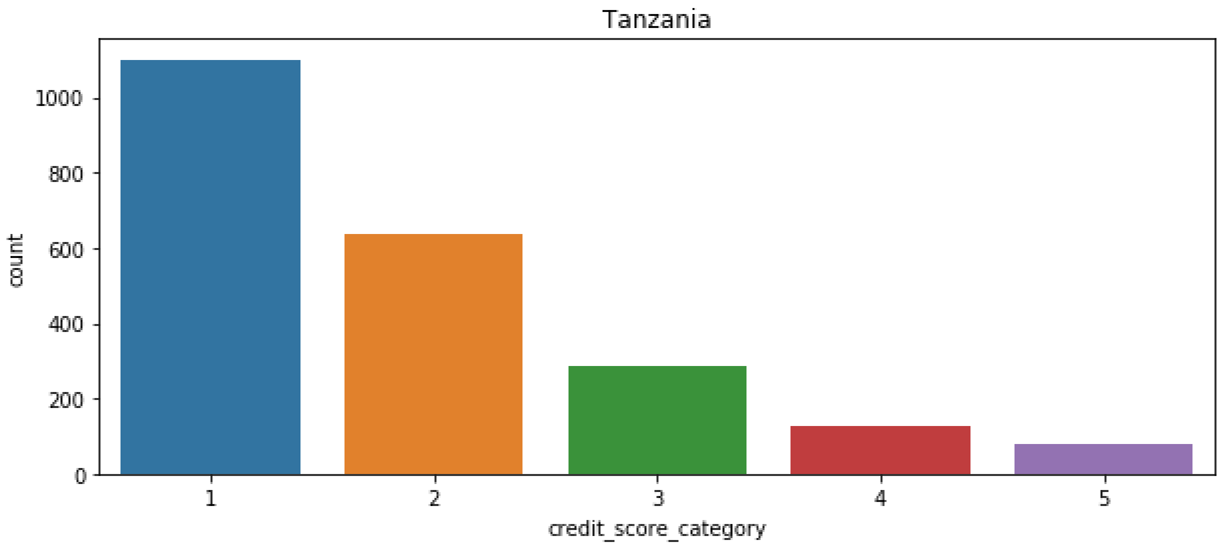
In [20]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=kenya['credit_score_category']);
_ = plt.title("Kenya")
```



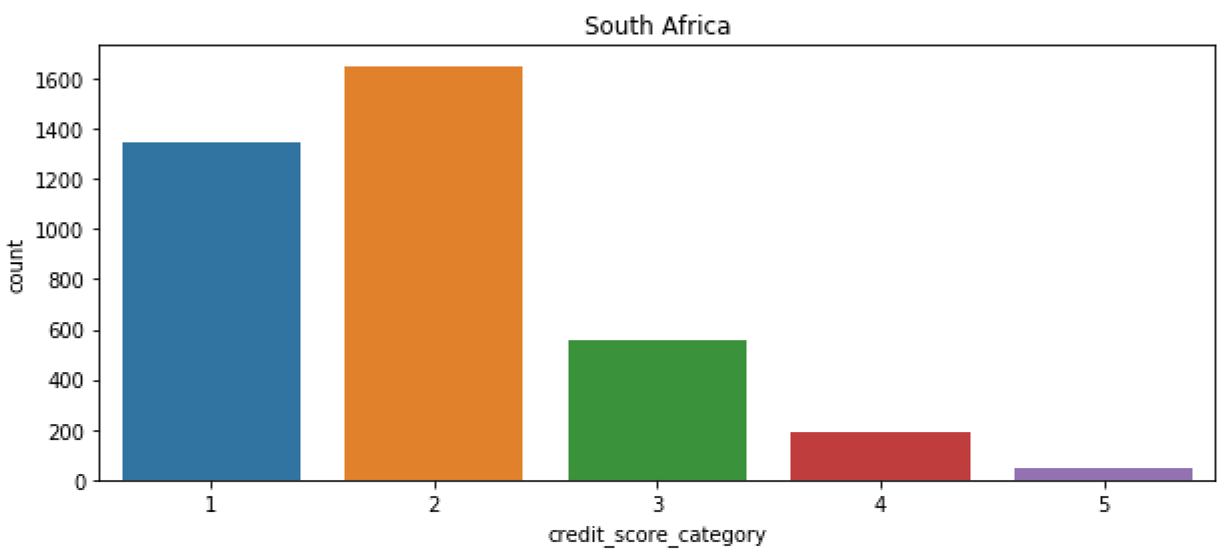
In [21]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=nigeria['credit_score_category']);
_ = plt.title("Nigeria")
```



In [22]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=tanzania['credit_score_category']);
_ = plt.title("Tanzania")
```



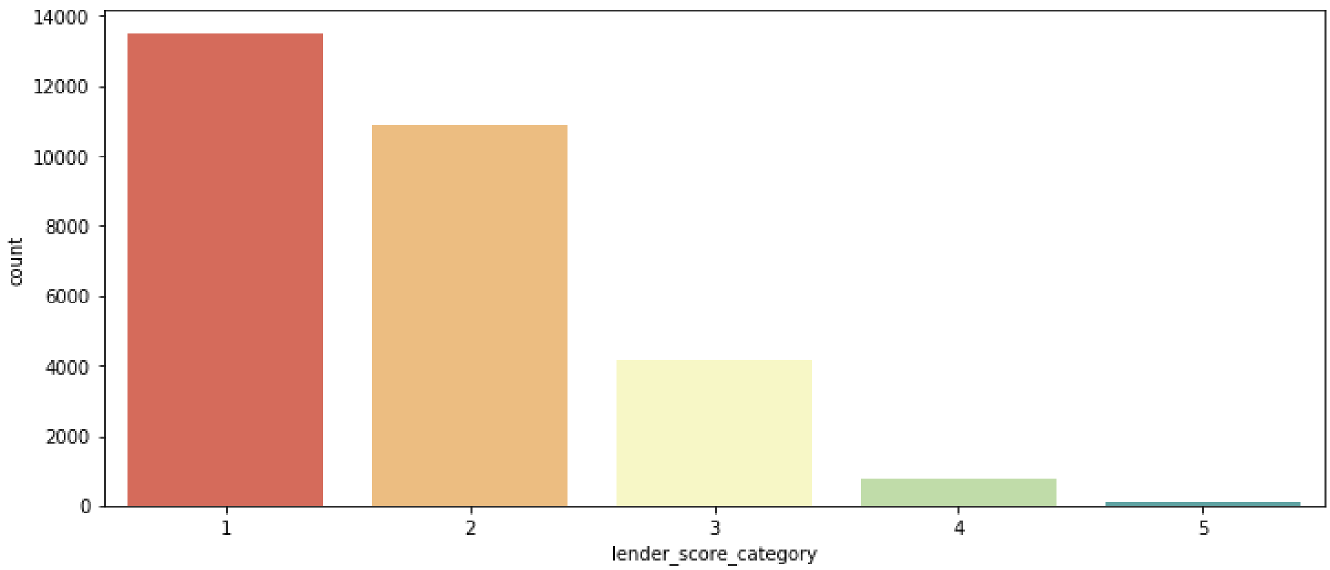
In [23]:
```
plt.figure(figsize=(10,4))
_ = sns.countplot(x=south_africa['credit_score_category']);
_ = plt.title("South Africa")
```



Lender Score Categories

In [24]:
```
plt.figure(figsize=(12,5))
sns.countplot(x=df['lender_score_category'], log=False, palette='Spectral')
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x261e6c360b8>



This is for information olny. Do not include into the report

In [25]:
```
negative_score = df[df['credit_score'] <0]
negative_score.shape[0]/df.shape[0]
```

Out[25]: 0.1022359867950856

In [26]:
```
negative_score = df[df['lender_score'] <0]
negative_score.shape[0]/df.shape[0]
```

Out[26]: 0.1555661436885274

Final data preparation. Take important features. Do usampling.

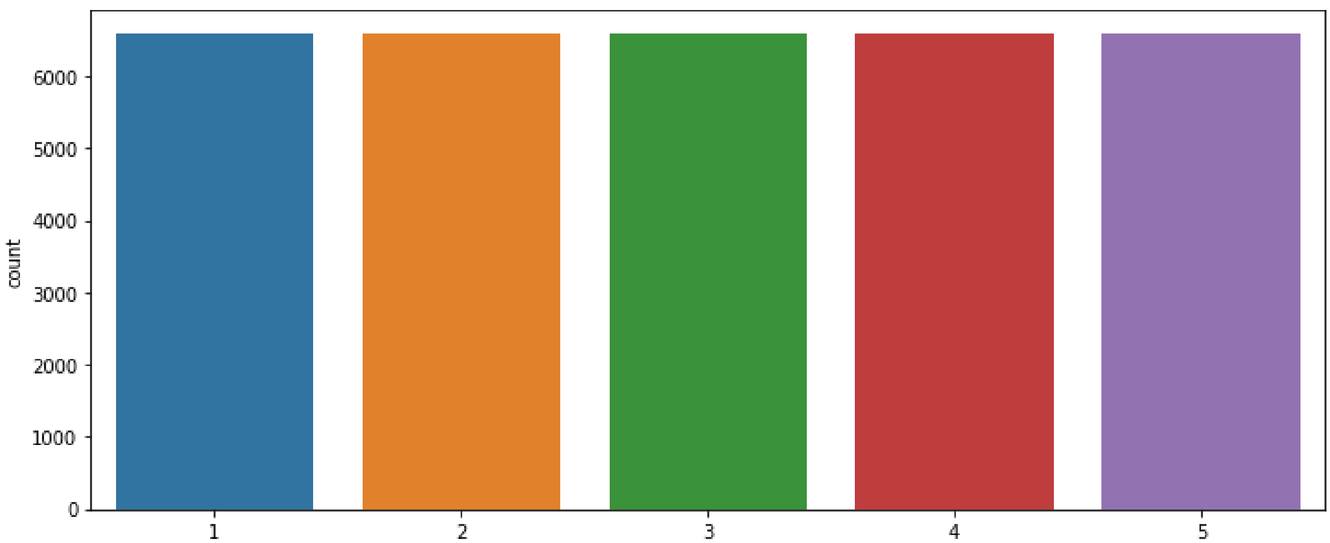
In [27]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
## credit score datasets
cs_data = df.iloc[:,[24,26,22,18,20,23,19,16,36,25]]
credit_score = df['credit_score_category']

# lender score dataset
ls_data = df.iloc[:,[24,18,22,23,20,26,19,25,16,12,11,21]]
lender_score = df['lender_score_category']
cs_data.shape, credit_score.shape,ls_data.shape, lender_score.shape
```

Out[27]: ((29383, 10), (29383,), (29383, 12), (29383,))

```
In [28]: # split to train/test 75/25

train_cs, test_cs, train_cs_class, test_cs_class = train_test_split(cs_data, credit_score, test_size=0.5)
train_ls, test_ls, train_ls_class, test_ls_class = train_test_split(ls_data, lender_score, test_size=0.3)

# usample both training sets
train_cs, train_cs_class = SMOTE().fit_resample(train_cs, train_cs_class)
train_ls, train_ls_class = SMOTE().fit_resample(train_ls, train_ls_class)
```

Upsampled Credit Score Categories of Training Set

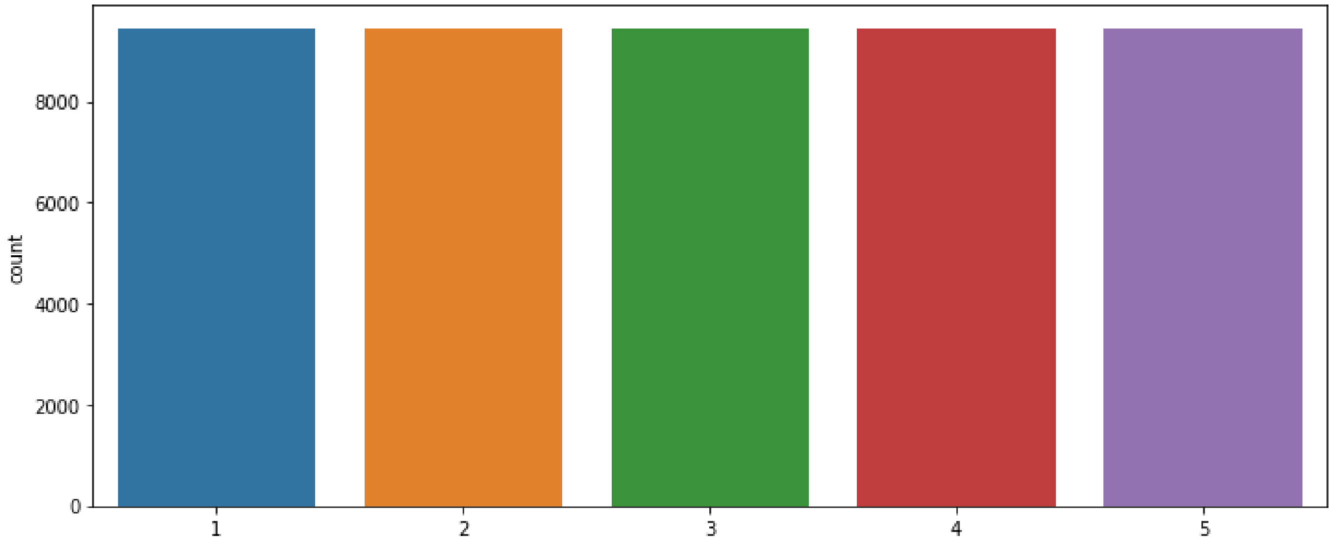
```
In [29]: import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,5))
sns.countplot(x=train_cs_class, log=False);
```




```
In [30]: train_cs.shape,train_cs_class.shape

Out[30]: ((32925, 10), (32925,))
```

Upsmapled Lender Score Categories of Training set

In [31]:
```
plt.figure(figsize=(12,5))
sns.countplot(x=train_ls_class, log=False);
```



In [32]:
```
train_ls.shape,train_ls_class.shape
```

Out[32]: ((47230, 12), (47230,))

Model Training

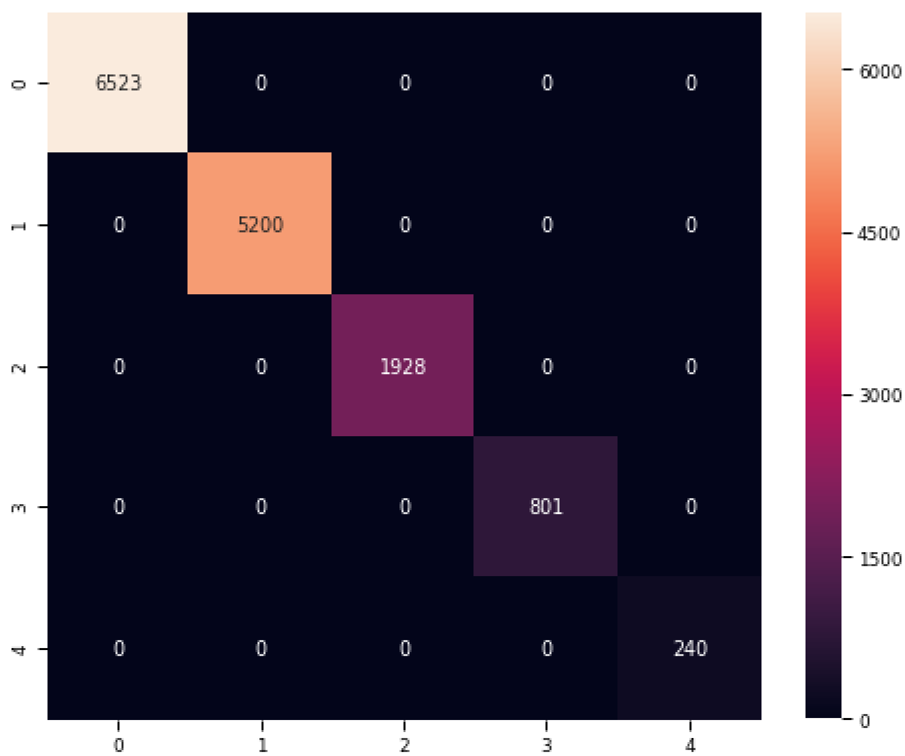
In [33]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import learning_curve, GridSearchCV
width = 8
height = 6.5
```

Credit Score Model Training

Random Forest

In [34]:
```python
rf = RandomForestClassifier(n_estimators=100, min_samples_split=20)
_ = rf.fit(train_cs,train_cs_class)
rfp = rf.predict(test_cs)
print("Random Forest  model score: {:.4}".format( rf.score(test_cs,test_cs_class)))
rfcm = confusion_matrix(test_cs_class, rfp)
sns.set_context("paper", rc={"lines.linewidth": 1})
_ =plt.figure(figsize=(width, height))
rfr = classification_report(test_cs_class, rfp)
print(rfr)
_= sns.heatmap(pd.DataFrame(rfcm), annot=True, fmt="d")
```

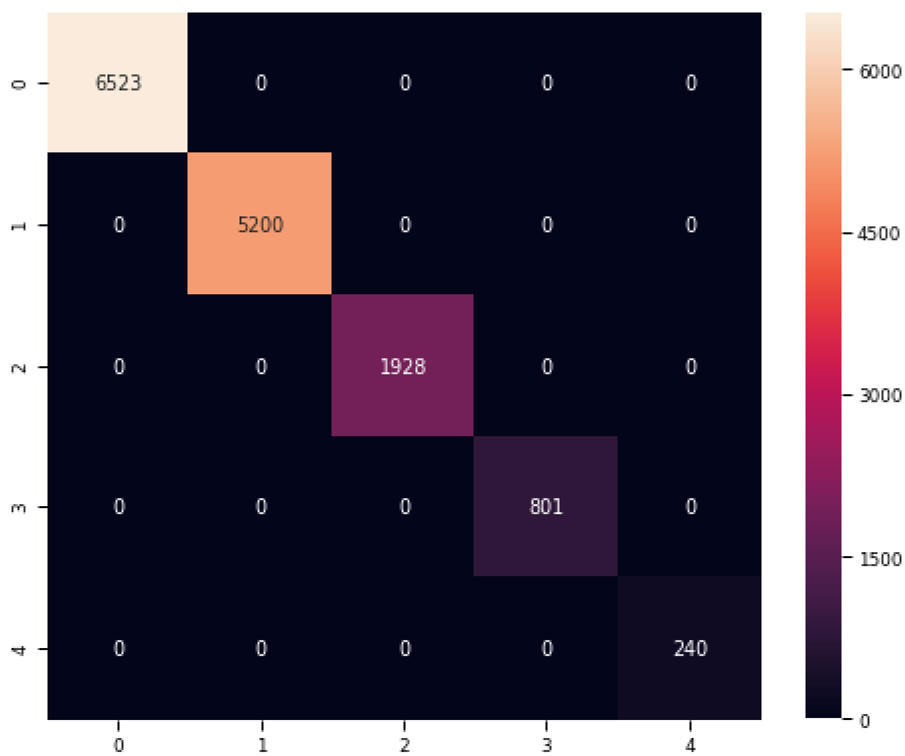
```
Random Forest  model score: 1.0
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      6523
           2       1.00      1.00      1.00      5200
           3       1.00      1.00      1.00      1928
           4       1.00      1.00      1.00       801
           5       1.00      1.00      1.00       240

   micro avg       1.00      1.00      1.00     14692
   macro avg       1.00      1.00      1.00     14692
weighted avg       1.00      1.00      1.00     14692
```



SVM

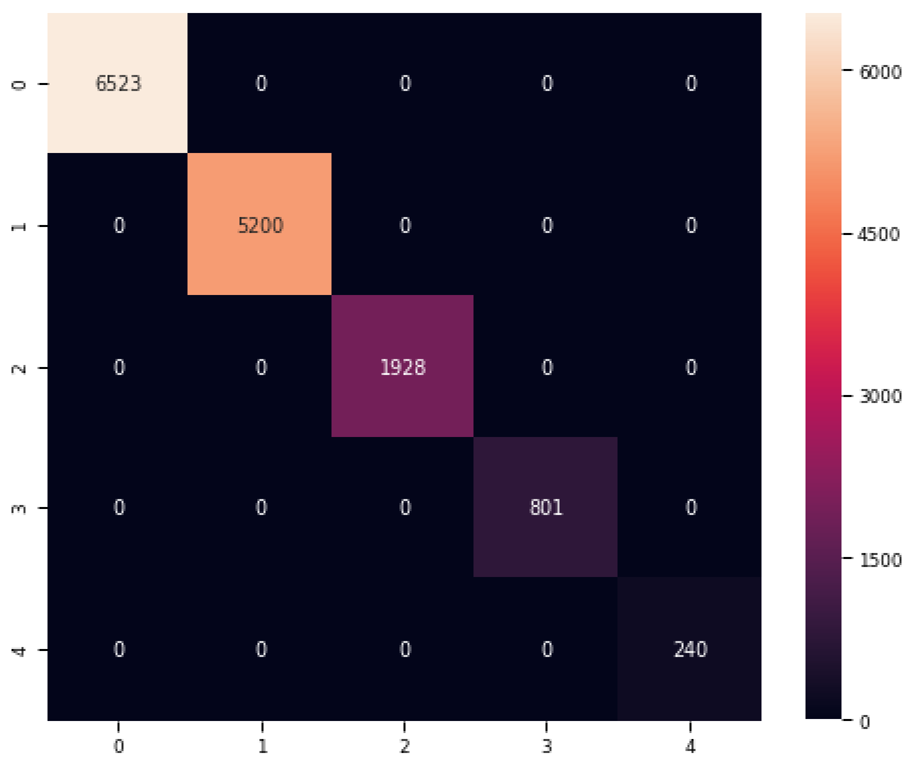
In [35]:
```
svc = SVC(gamma='scale')
_ = svc.fit(train_cs,train_cs_class)
svcp = svc.predict(test_cs)
print("SVC  model score: {:.4}".format( svc.score(test_cs,test_cs_class)))
svccm = confusion_matrix(test_cs_class, svcp)
sns.set_context("paper", rc={"lines.linewidth": 1})
_ =plt.figure(figsize=(width, height))
svcr = classification_report(test_cs_class, svcp)
print(svcr)
_= sns.heatmap(pd.DataFrame(svccm), annot=True, fmt="d")
```

```
SVC  model score: 1.0
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      6523
           2       1.00      1.00      1.00      5200
           3       1.00      1.00      1.00      1928
           4       1.00      1.00      1.00       801
           5       1.00      1.00      1.00       240

   micro avg       1.00      1.00      1.00     14692
   macro avg       1.00      1.00      1.00     14692
weighted avg       1.00      1.00      1.00     14692
```



In [36]:
```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random
_state=0 )
_ = gb.fit(train_cs,train_cs_class)
gbp = gb.predict(test_cs)
print("Random Forest  model score: {:.4}".format( gb.score(test_cs,test_cs_class)))
gbcm = confusion_matrix(test_cs_class, gbp)
sns.set_context("paper", rc={"lines.linewidth": 1})
_ =plt.figure(figsize=(width, height))
gbr = classification_report(test_cs_class, gbp)
print(gbr)
_= sns.heatmap(pd.DataFrame(gbcm ), annot=True, fmt="d")
```

```
Random Forest  model score: 1.0
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      6523
           2       1.00      1.00      1.00      5200
           3       1.00      1.00      1.00      1928
           4       1.00      1.00      1.00       801
           5       1.00      1.00      1.00       240

   micro avg       1.00      1.00      1.00     14692
   macro avg       1.00      1.00      1.00     14692
weighted avg       1.00      1.00      1.00     14692
```



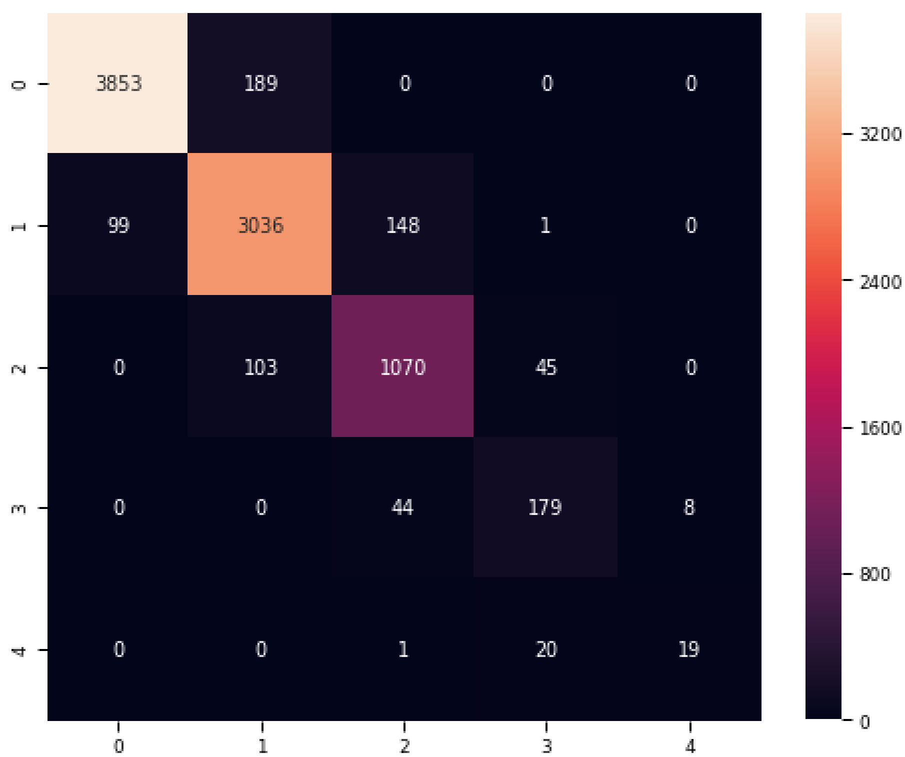
Lender Score Model Training

Random Forest

In [37]:

```
rf = RandomForestClassifier(n_estimators=200, min_samples_split=20)
_ = rf.fit(train_ls,train_ls_class)
rfp = rf.predict(test_ls)
print("Random Forest  model score: {:.4}".format( rf.score(test_ls,test_ls_class)))
rfcm = confusion_matrix(test_ls_class, rfp)
sns.set_context("paper", rc={"lines.linewidth": 1})
_ =plt.figure(figsize=(width, height))
rfr = classification_report(test_ls_class, rfp)
print(rfr)
_= sns.heatmap(pd.DataFrame(rfcm), annot=True, fmt="d")
```

```
Random Forest  model score: 0.9254
              precision    recall  f1-score   support

           1       0.97      0.95      0.96      4042
           2       0.91      0.92      0.92      3284
           3       0.85      0.88      0.86      1218
           4       0.73      0.77      0.75       231
           5       0.70      0.47      0.57        40

   micro avg       0.93      0.93      0.93      8815
   macro avg       0.83      0.80      0.81      8815
weighted avg       0.93      0.93      0.93      8815
```



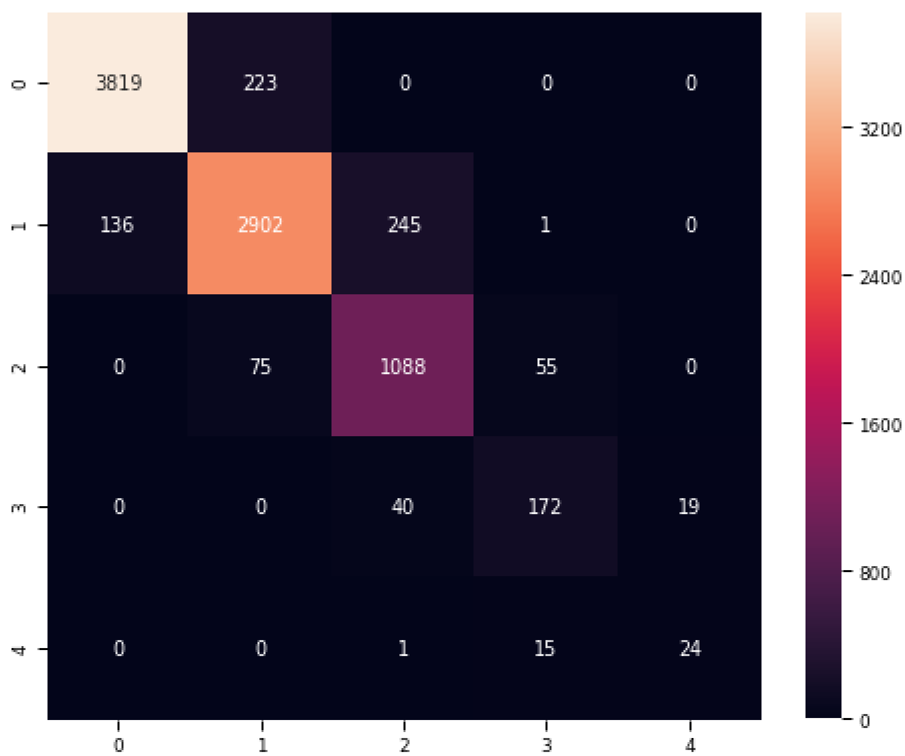
In [130]:
```
params = {'n_estimators':[200, 300, 400], 'min_samples_split':[20,30,40,50]}
gs = GridSearchCV( RandomForestClassifier(),params, cv=3)
_ = gs.fit(train_ls,train_ls_class)
print("Best Random Forest model score: %0.3f" % gs.best_score_)
print("Best model hyperparameters:")
bestParams = gs.best_estimator_.get_params()
for paramName in sorted(params.keys()):
   print("\t%s: %r" % (paramName, bestParams[paramName]))
```

```
Best Random Forest model score: 0.951
Best model hyperparameters:
        min_samples_split: 20
        n_estimators: 200
```

SVM

In [39]:
```
svc = SVC(gamma='scale')
_ = svc.fit(train_ls,train_ls_class)
svcp = svc.predict(test_ls)
print("SVC  model score: {:.4}".format( svc.score(test_ls,test_ls_class)))
svccm = confusion_matrix(test_ls_class, svcp)
sns.set_context("paper", rc={"lines.linewidth": 1})
_ =plt.figure(figsize=(width, height))
svcr = classification_report(test_ls_class, svcp)
print(svcr)
_= sns.heatmap(pd.DataFrame(svccm), annot=True, fmt="d")
```

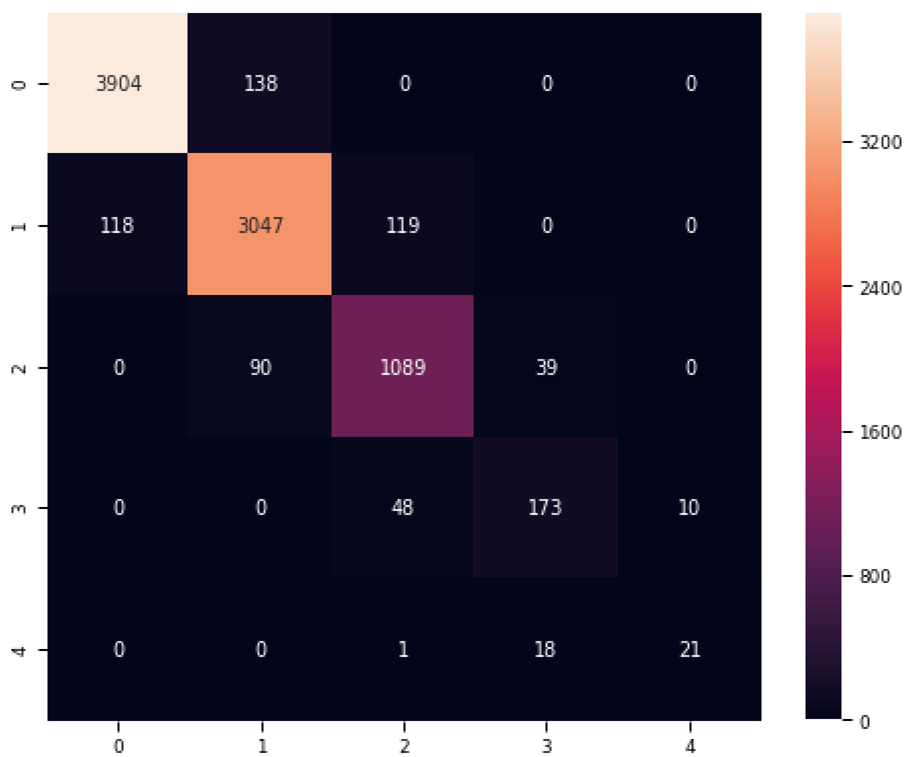
```
SVC  model score: 0.9081
              precision    recall  f1-score   support

           1       0.97      0.94      0.96      4042
           2       0.91      0.88      0.90      3284
           3       0.79      0.89      0.84      1218
           4       0.71      0.74      0.73       231
           5       0.56      0.60      0.58        40

   micro avg       0.91      0.91      0.91      8815
   macro avg       0.79      0.81      0.80      8815
weighted avg       0.91      0.91      0.91      8815
```



Gradient Boost

In [40]:
```
gb = GradientBoostingClassifier(n_estimators=300, learning_rate=0.1, max_depth=6, random
_state=0, min_samples_split=30 )
gb_model = gb.fit(train_ls,train_ls_class)
gbp = gb.predict(test_ls)
print("Random Forest  model score: {:.4}".format( gb.score(test_ls,test_ls_class)))
gbcm = confusion_matrix(test_ls_class, gbp)
sns.set_context("paper", rc={"lines.linewidth": 1})
_ =plt.figure(figsize=(width, height))
gbr = classification_report(test_ls_class, gbp)
print(gbr)
_= sns.heatmap(pd.DataFrame(gbcm ), annot=True, fmt="d")
```

```
Random Forest  model score: 0.9341
              precision    recall  f1-score   support

           1       0.97      0.97      0.97      4042
           2       0.93      0.93      0.93      3284
           3       0.87      0.89      0.88      1218
           4       0.75      0.75      0.75       231
           5       0.68      0.53      0.59        40

   micro avg       0.93      0.93      0.93      8815
   macro avg       0.84      0.81      0.82      8815
weighted avg       0.93      0.93      0.93      8815
```



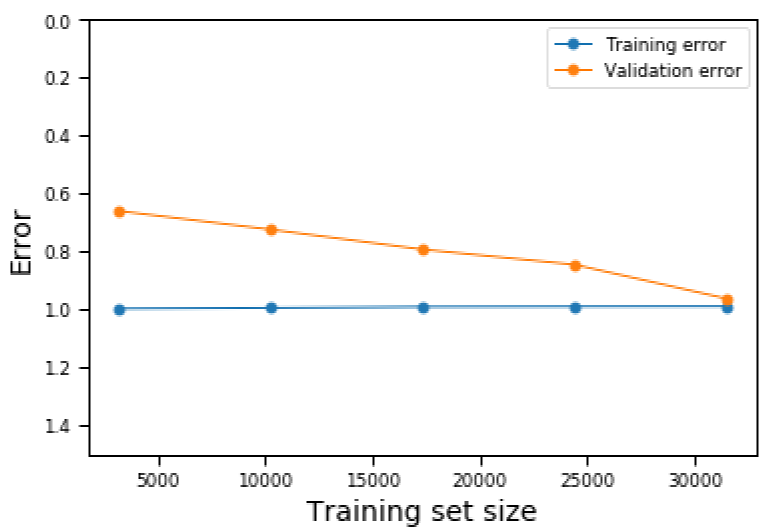
In [42]:
```
params = {'n_estimators':[200, 300 ],'learning_rate':[0.1,1.0,1.2], 'min_samples_split':
[5,20,30],'max_depth':[3,6] }
gs = GridSearchCV( GradientBoostingClassifier(),params, cv=2)
_ = gs.fit(train_ls,train_ls_class)
print("Best Random Forest model score: %0.3f" % gs.best_score_)
print("Best model hyperparameters:")
bestParams = gs.best_estimator_.get_params()
for paramName in sorted(params.keys()):
   print("\t%s: %r" % (paramName, bestParams[paramName]))
```

```
Best Random Forest model score: 0.959
Best model hyperparameters:
        learning_rate: 0.1
        max_depth: 6
        min_samples_split: 30
        n_estimators: 300
```

Learning Curves

In [41]:
```
# takes time
train_sizes, train_scores, validation_scores = learning_curve(
   estimator = gb_model,X = train_ls, y = train_ls_class, cv = 3)
```

In [42]:
```
train_scores_mean = train_scores.mean(axis = 1)
validation_scores_mean = validation_scores.mean(axis = 1)
_ = plt.plot(train_sizes, train_scores_mean,'o-', label = 'Training error')
_ = plt.plot(train_sizes, validation_scores_mean,'o-', label = 'Validation error')

_ = plt.ylabel('Error', fontsize = 14)
_ = plt.xlabel('Training set size', fontsize = 14)
_ = plt.legend()
_ = plt.ylim(1.5,0)
_ =plt.figure(figsize=(width, height))
plt.show()
```




```
<Figure size 576x468 with 0 Axes>
```

Verify Quality of the Data that are Included into the Formula

Risk Columns

Recipients

Raw	Description	Categories	Weights
Family	Family	1	-0.05
Friends	Friends	2	0.1
Business colleagues	Business Colleagues	3	0.1
None of the above	Other	4	-0.05

In [43]:
```
u=df['20'].unique() #20. Who did you lend money to in the past 3 months?
len(u),u
```

Out[43]:
```
(3, array([3, 2, 1], dtype=int64))
```

Interest

Raw	Description	Categories	Weights
Always /Toujours	Always	1	-0.1
Most of the time/ La moitié du temps	Most of the Time	2	-0.05
About half the time/ La moitié du temps	About half the time	3	0.05
Occasionally/ Souvent	Occasionally	4	0.08
Never / Jamais	Never	5	0.12

In [44]:
```
u = df['22'].unique() # 22:"Do you include either interest or a lending fee when you len
d?"
len(u), u
```

Out[44]:
```
(5, array([3, 4, 2, 5, 1], dtype=int64))
```

Collateral

Raw	Description	Categories	Weights
Always /Toujours	Always	1	-0.05
Most of the time/ La moitié du temps	Most of the Time	2	-0.03
About half the time/ La moitié du temps	About half the time	3	0.03
Occasionally/ Souvent	Occasionally	4	0.04
Never / Jamais	Never	5	0.06

In [45]:
```
u = df['23'].unique() #23:"Do you request guarantees when you lend?"
len(u),u
```

Out[45]: (5, array([2, 5, 3, 4, 1], dtype=int64))

Liquidity

Frequency

Raw	Description	Categories	Weights
More than 4 times / Plus de 4 foiss	More than 4 times	1	0.1
Between 2-3 times / Entre 2 et 3 fois	2-3 times	2	0.07
Once / Une fois	Once	3	-0.02
Never / Jamais	Never	4	-0.05

In [46]:
```
u=df['18'].unique() #18:"Over the past 3 months, how many times have you lent someone mo
ney?"
len(u),u
```

Out[46]: (4, array([2, 3, 1, 4], dtype=int64))

Duration

Raw	Description	Categories	Weights
Less than a month / Moins d'un mois	Less tan a month	1	-0.01
At least 1 month but than 3 months / 1 - 2 mois	At least 1 month but less than 3 months	2	0.02
At least 3 months but less than 6 months / 3 - 5 mois	At least 3 months but less than 6 months	3	0.03
At least 6 months but less than 12 months / 6 - 11 mois	At least 6 months but less than 12 months	4	0.04
One year or more / Plus d'un an	One year or more	5	0.05
???	???	6	-0.02

In [47]:
```
u=df['21'].unique() #21:"When you lend money, when do you usually expect to get it repai
d?"
len(u),u
```

Out[47]: (6, array([3, 2, 1, 5, 4, 6], dtype=int64))

Amount

Raw	Description	Categories	Weights
???	Micro	1	-0.05
???	Small	2	0.03
???	Medium	3	0.05
???	Large	4	0.07

In [48]:
```
u=df['19'].unique()
len(u),u
```
Out[48]:
```
(4, array([2, 1, 3, 4], dtype=int64))
```

Default

Raw	Description	Categories	Weights
Always /Toujours	Always	1	0.35
Most of the time/ La moitié du temps	Most of the Time	2	0.25
About half the time/ La moitié du temps	About half the time	3	0.15
Occasionally/ Souvent	Occasionally	4	-0.1
Never / Jamais	Never	5	-0.3

In [49]:
```
u=df['24'].unique() # 24:"Do you receive your money back in time?"
len(u),u
```
Out[49]:
```
(5, array([3, 2, 4, 5, 1], dtype=int64))
```

Usage

Raw	Description	Categories	Weights
Same	To cover regular expenses (Rent, clothing, home appliances, etc.)	1	-0.1
Same	To pay for one-time or sudden expenses (Wedding, medical emergencies, etc.)	2	0.05
Same	To invest or cover business expenses (Merchandise, salaries, etc..)	3	0.15
To pay off other debts	To pay off other debts	4	-0.05
I don't know	I don't know	5	0.05

In [50]:
```
u=df['26'].unique() #26:"What's the most common use of the money you lend?"
len(u),u
```
Out[50]:
```
(5, array([3, 2, 1, 4, 5], dtype=int64))
```