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
ı	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	

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 [8]: df.head()

```
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'))
```

```
Score
   Feature
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
         2
0
              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]: 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]: 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
```

3579.344815 22

23 3308.826808

18 2021.701786

16 568.578714

25 555.864421

19 472.875751

26 447.615584

20 411.577992 196.022728 36

14

172.512950

164.043637

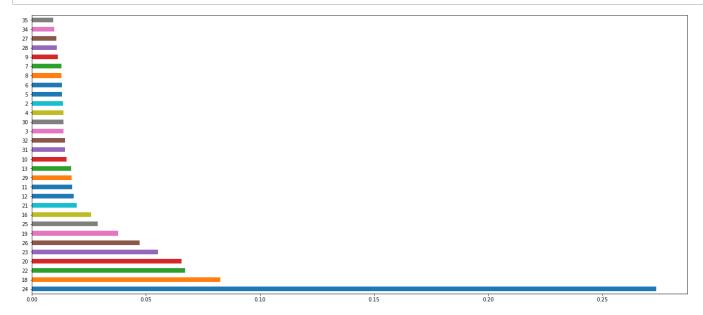
31 141.347621

114.004620 21

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

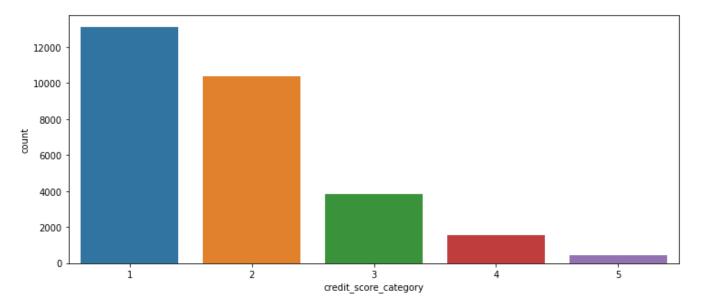
```
N - 1 01 00370.0570.14 0.11 0.110.00210.110.0520.0260.0360.0560.170.0520.0560.088.01.2-0.0790.056.0048.0810.066.003-0.020.0640.0110.044.0072.0049.0390.0580.0310.05
m - 0.1 1 40.24 0.3 -0.23-0.25-0.240.0170.0130.0190.0230.0290.0190.110.0530.0180.0330.023-0.12-0.140.0180.0940.0380.0099.0320.020.00688.0330.0170.0330.0470.050.0240.02
9-0.037-0.24 1 -0.17 019 019 017-0.0370051002D00770035007D04500950.03800190.0180.13 0.14 0.04700450.03 0.0630.0780.025 0.010 0.0350.0160.0230.0310.0580.0028.02
v. -0.057 0.3 -0.17 1 -0.29-0.28-0.260.0210.0470.029.048-0.040.0630.038.0056.0089.04-0.060.0099.0520.0180.0190.0760.0620.0420.036.0098.003500540.0590.0540.0610.0430.02
     0.11 - 0.24 0.17 - 0.26 0.48 0.52 1 0.01 90.07 10.01 30 001 90.048 0.02-0.04 90.043-0.020.0360.0450.0480.0610.0810.0770.082 0.04 0.0260.0360.01 50.01 30.01 70.02 0.07 0.047 0000 32.01
      0021/01/70.0370.0210.0230.0180.019 1 0.0230.0560.0240.0090.0310.0250.0680.044.0006110330.0170.0240.0260.0190.0210.0240.0240.0246.00415.00410.00410.0041.00746.0160.03
      2.11-0.0110.0510.0470.0760.0750.0710.02: 1 0.22 0.16 0.14 0.31 0.0220.0140.0880.0670.0910.0210.070.0088 0860.0390.0120.0420.0740.0440.0810.0810.0310.0230.0740.0290.220.01
      0520.0190.0210.0250.0260.0150.0130.0560.022 1 0.27 0.25 0.0440.0110.0430.19 0.0780.110.0098.0044.0030.0340.0140.0370.01 0.0980.038.0.04.0.0290.0170.0630.036.0098.058
      0260.02800770.04800012004200190.0240.16 027 1 0.0790.0640.0180.043 0.1 0.0680.08 0.080.0650.0110.0620.0028.007900750.1 0.13 0.13 0.0910.0550.0140.0550.014
      0360.0290.035-0.040.0620.0430.0480.0090.14 0.25 0.079 1 0.120.00340.0110.0540.0550.0650.0150.0150.0150.0370.0520.026-0.0180006550.28.00650.0310.03 0.0940.00640.0120.140.000
      005@.01@007£0.0620.028.002.002.0031<mark>0.31</mark>0.0440.064.0.12 1 0.0480.0380.0170.02.00790.0380.0280.0530.014.0.07 0.030.0560.042.0.12 0.150.088<mark>0.72 0.22-0.23-0.67-</mark>0.11
     0.17-0.110.0450.0310.080.0490.0490.0290.0290.0110.0186.00340.045 1 0.19.00190.012-0.14 0.24 0.25-0.0360.130.0570.0630.230.00930.19 0.27 0.31 0.079 0.2 0.25 0.170.02
      0520.0520.0520.0930.00560.02.0.0370.0430.0630.0140.0430.0430.0140.0380.119 1 -0.16-0.06-0.12 0.29 0.26 0.23 0.0160.0380.099.0.13-0.036.00730.0280.0410.00530.0880.088-0.03 0.14
      0560.0180.038.008$0058.00820.02.0.0440.088.019 0.1 0.0540.0170.019-0.16 1 0.14 0.18-0.0910.0990.0360.0330.021-0.130.0880.0330.0520.0690.0540.0120.0740.0990.0260.18
      0880 0330 019-0 040 0420 0360 0360 0061 0670 0780 0680 055 0.02 0 012-0 06 0.14 1 0.12-0 0620 0740 0310 0210 069-0 030 00560 12 0 017 0.02 0 0150 001-0 020 00540060 018
      212 0 0 2 3 0 0 1 8 0 0 6 0 0 7 1 0 0 5 0 0 4 5 0 0 3 3 0 0 9 1 0 1 1 0 0 8 0 0 6 5 0 0 7 9 0 1 4 0 1 2 0 1 8 0 1 2 1 0 0 7 8 0 0 9 1 0 0 5 0 2 1 0 0 4 0 0 5 9 0 0 7 6 0 0 6 0 0 1 4 0 0 1 3 0 0 2 6 0 6 5 3 0 0 4 9 0 0 8 5 0 0 1 0 0 8 0 0 1 0 0 1 0 0 8 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 
      0790.12 0.130.009$500126.0280.0480.0170.0210.00980.080.0150.0380.24 0.29 0.0930.0620.078 1 0.65 0.29 0.0940.00180.16 0.2 0.0330.0820.097 0.130.00940.0030.130.00510.13
      .0560.14.0.14.0.0520.0360.0570.0610.024.0079.0044.0650.0150.0290.25.0.26-0.0960.070.09<mark>1.065</mark> 1 0.330.0960.027.0.17 0.22.0.056 0.1 0.13 0.160.0150.0290.15-0.0230.13
      00440 0120 0470 0180 07 0 0660 0810 026 0086 0030 0110 0370 0530 0360 23-0 0890 0310 0560 29 0 33 1 0 33 0 079 0 11 0 0810 035 0 020 0490 0190 0350 0380 0420 057 0 13
      08D 0094 00450 0150 0710 0690 0770 0190 0860 0340 0620 0520 014-0 130 0160 0330 0210 210 0970 0960 33 1 0 022 00092 0370 0590 0110 0330 0180 022 0 060 001D 0120 02
      0660.0320.03-0.0760.0980.0790.0820.0210.0390.0140.00230.0260.07-0.0570.0380.0210.0690.040.0018.0270.0790.022 1 0.097.01.0.110.0890.0610.07.0.070.00380.0120.0680.003
      .030.0098.0630.0620.0340.039 0.04-0.0240.0120.0370.00790.03 0.03 0.0630.099-0.13-0.030.0590.16 0.17 0.170.000972097 1 0.4 0.15 0.18 0.18 0.19 0.0430.0230.260 0.0640 12
      064-0.020.025-0.0360.03.0.0240.0360.0048.0740.098 0.1 0.0230.0420.00930.0360.0330.012 0.0660.0330.0560.0350.059.0.11 0.15 0.13 1 0.19 0.18 0.17 0.0210.0710.090.0014.05
      0140.00690.010.00990.0360.0220.0149.00610.0440.038.0.130.00650.12 0.190.00790.0520.0170.0140.082 0.1 -0.024.0.0110.089 0.18 0.24 0.19 1 0.62 0.56 0.097-0.13 0.22-0.0620.21
      0440.0340.0036.00360.0520.0250.018.00250.0810.04 0.13 0.0310.15 0.27 0.0280.0690.02-0.0130.097 0.13-0.0450.0330.0610.18 0.28 0.18 0.62 1 0.67 0.14 0.19 0.27 -0.18-0.24
      00778.0170.0160.00540.0530.0260.0170.00970.0810.0290.091.003.0088.0310.0410.0540.0150.0260.13 0.16-0.0190.0180.007 0.19 0.3 0.17 0.56 0.67 1 0.087-0.23 0.32 -0.13-0.24
      0048,0330.0230.0590.0280.03 0.02 0.041<mark>0.23</mark> 0.0170.0550.096<mark>0.72</mark>0.0790.00590.0120.0010.0530.00940.0150.0390.022 0.07 0.0430.0770.0210.097 0.14 0.087 1 0.17 -0.18-0.610.08
      0390 0470 0310 0540 0890 081 0 070 0046 0740 0610 015 00660 22 0 20 00820 0740 020 0490 0030 029 038 0 060 0035 0230 110 0710 13 0 19 0 23 017 1 0 26 0 130 02
      0580 050 0530 0610 0310 0370 0470 0076 0290 0360 0140 0120 23 025 0 0880 099 0054 0850 13 0 15 0 0470 00170 0120 26 0 27 0 09 0 22 0 27 0 32 0 18 0 26 1 0 17 0 21
      0310 0240 00270 0470 00674 0010 0900 0390 0160 220 00990 0550 11 0 067 0 17 0 030 0260 00670 010 000570 0230 0570 0120 0670 00640 10 0014 062 0 18 0 13 0 61 0 13 0 61 0 13 0 017 1 0 008
      3 4 5 6 7 8 9 10 11 12 13 14 16 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
```

- 0.6

Dataset Balance

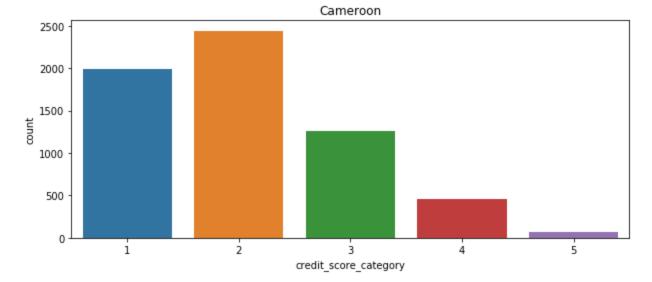
Credit Score Categories

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

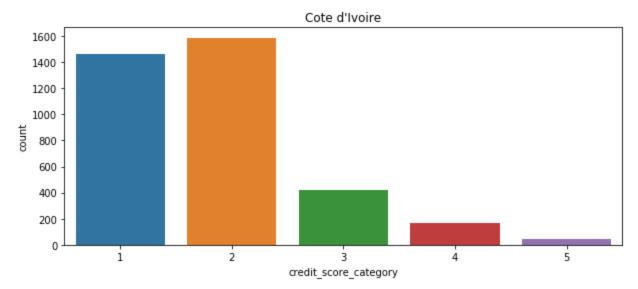


```
In [16]: cameroon = df[df['16'] == 1]
    cote = df[df['16'] == 2]
    ghana = df[df['16'] == 3]
    kenya = df[df['16'] == 4]
    nigeria = df[df['16'] == 5]
    south_africa = df[df['16'] == 6]
    tanzania = df[df['16'] == 7]
```

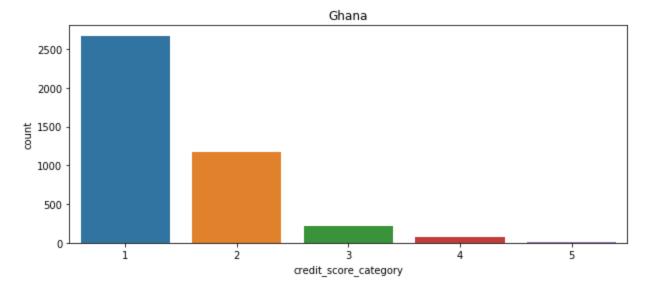
```
In [17]: plt.figure(figsize=(10,4))
    _ = sns.countplot(x=cameroon['credit_score_category']);
    _ = plt.title("Cameroon")
```



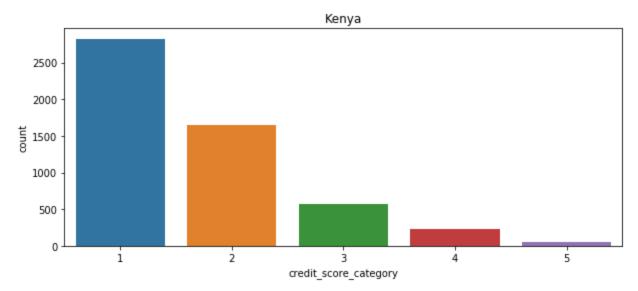
```
In [18]: plt.figure(figsize=(10,4))
    _ = sns.countplot(x=cote['credit_score_category']);
    _ = plt.title("Cote d'Ivoire")
```



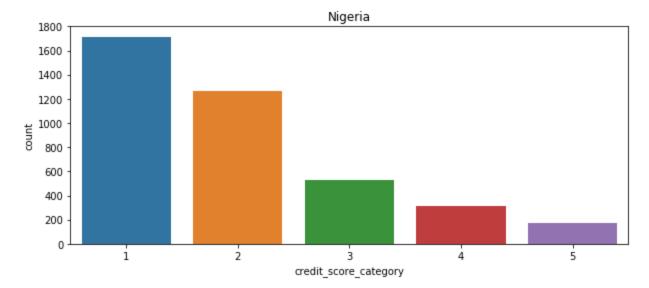
```
In [19]: plt.figure(figsize=(10,4))
    _ = sns.countplot(x=ghana['credit_score_category']);
    _ = plt.title("Ghana")
```



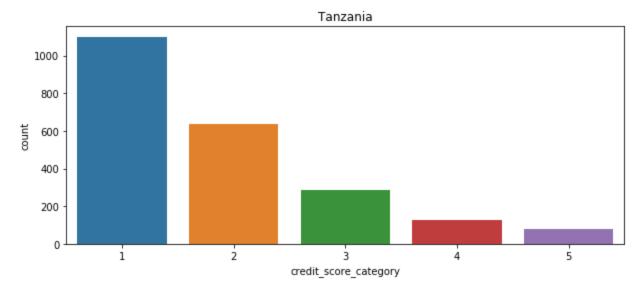
```
In [20]: plt.figure(figsize=(10,4))
    _ = sns.countplot(x=kenya['credit_score_category']);
    _ = plt.title("Kenya")
```



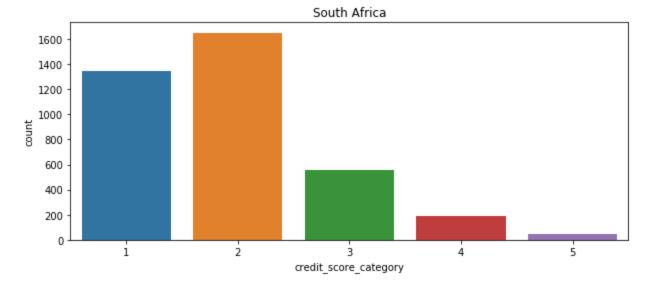
```
In [21]: plt.figure(figsize=(10,4))
    _ = sns.countplot(x=nigeria['credit_score_category']);
    _ = plt.title("Nigeria")
```



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



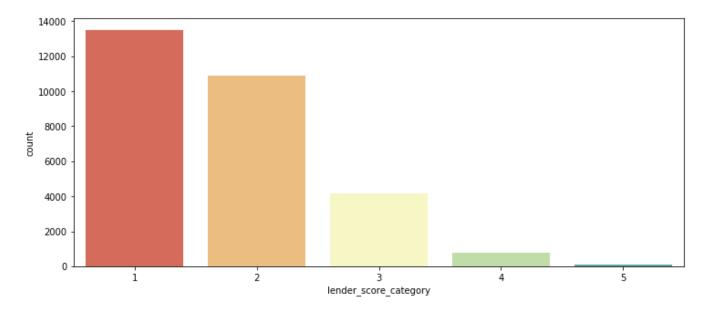
```
In [23]: plt.figure(figsize=(10,4))
    _ = sns.countplot(x=south_africa['credit_score_category']);
    _ = plt.title("South Africa")
```



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

```
negative_score = df[df['credit_score'] <0]</pre>
In [25]:
          negative_score.shape[0]/df.shape[0]
Out[25]: 0.1022359867950856
In [26]: | negative_score = df[df['lender_score'] <0]</pre>
          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

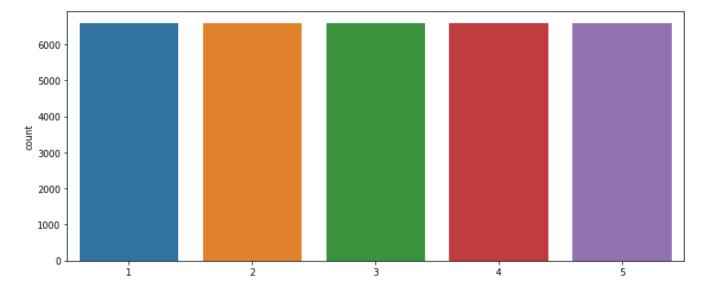
train_cs, test_cs, train_cs_class, test_cs_class = train_test_split(cs_data, credit_scor e, test_size=0.5)
    train_ls, test_ls, train_ls_class, test_ls_class = train_test_split(ls_data, lender_scor e, test_size=0.3)

# 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

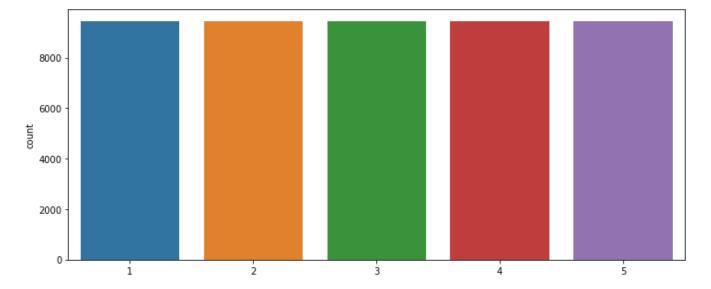
plt.figure(figsize=(12,5))
sns.countplot(x=train_cs_class, log=False);
```



```
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,))
```

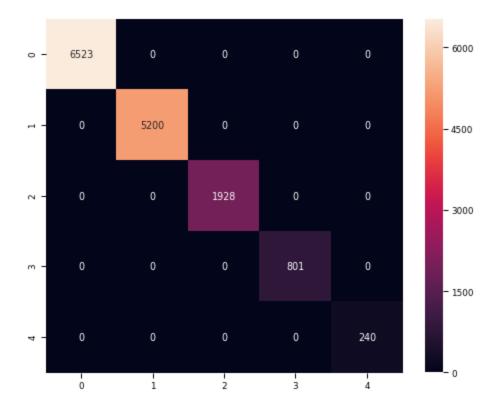
Model Training

```
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

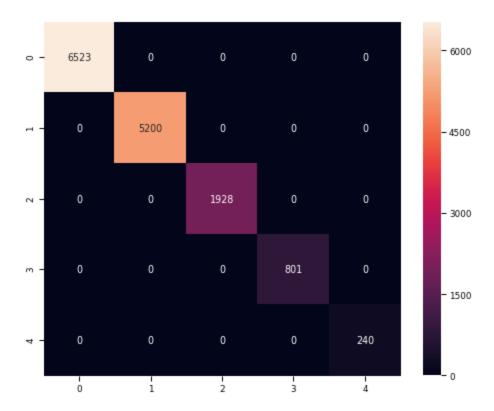
Random Forest

Random Fo	orest	model score	e: 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



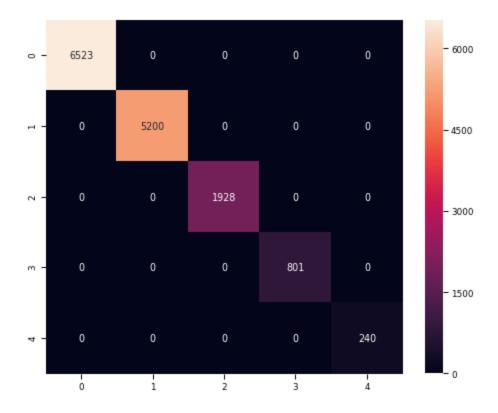
```
In [35]: svc = SVC(gamma='scale')
    _ = svc.fit(train_cs,train_cs_class)
    svcp = svc.predict(test_cs)
    print("SVC model score: {:.4}".format( svc.score(test_cs,test_cs_class)))
    svccm = confusion_matrix(test_cs_class, svcp)
    sns.set_context("paper", rc={"lines.linewidth": 1})
    _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

mode	el sc	ore: 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
icro	avg	1.00	1.00	1.00	14692
acro	avg	1.00	1.00	1.00	14692
nted	avg	1.00	1.00	1.00	14692
	icro acro	1 2 3 4	1 1.00 2 1.00 3 1.00 4 1.00 5 1.00 icro avg 1.00 acro avg 1.00	precision recall 1 1.00 1.00 2 1.00 1.00 3 1.00 1.00 4 1.00 1.00 5 1.00 1.00 icro avg 1.00 1.00 acro avg 1.00 1.00	precision recall f1-score 1 1.00 1.00 1.00 2 1.00 1.00 1.00 3 1.00 1.00 1.00 4 1.00 1.00 1.00 5 1.00 1.00 1.00 icro avg 1.00 1.00 1.00 acro avg 1.00 1.00 1.00



```
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")
```

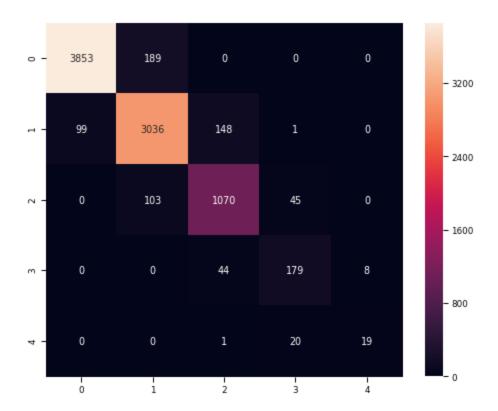
Random Fo	rest	model score	e: 1.0		
		precision	recall	f1-score	support
	1	1.00	1.00	1.00	6523
	2	1.00	1.00	1.00	5200
	3	1.00	1.00	1.00	1928
	4	1.00	1.00	1.00	801
	5	1.00	1.00	1.00	240
•		1 00	1 00	1 00	14602
micro	avg	1.00	1.00	1.00	14692
macro	avg	1.00	1.00	1.00	14692
weighted	avg	1.00	1.00	1.00	14692



Lender Score Model Training

Random Forest

Random Fo	orest	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

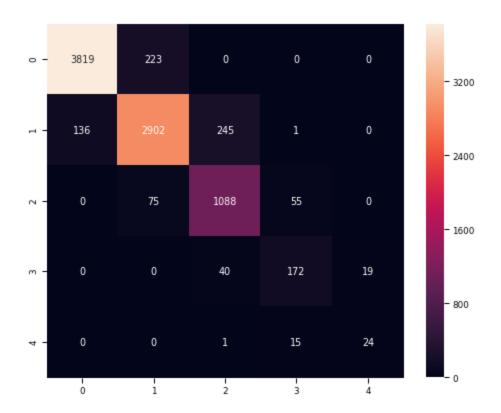


```
In [130]: params = {'n_estimators':[200, 300, 400], 'min_samples_split':[20,30,40,50]}
gs = GridSearchCV( RandomForestClassifier(),params, cv=3)
    _ = gs.fit(train_ls,train_ls_class)
    print("Best Random Forest model score: %0.3f" % gs.best_score_)
    print("Best model hyperparameters:")
  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

SVM

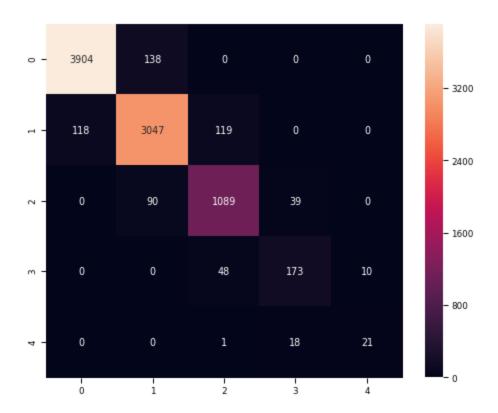
```
In [39]: svc = SVC(gamma='scale')
    _ = svc.fit(train_ls,train_ls_class)
    svcp = svc.predict(test_ls)
    print("SVC model score: {:.4}".format( svc.score(test_ls,test_ls_class)))
    svccm = confusion_matrix(test_ls_class, svcp)
    sns.set_context("paper", rc={"lines.linewidth": 1})
    _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

del			11	C1	
	precision		recall	T1-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
o av	g	0.91	0.91	0.91	8815
o av	'g	0.79	0.81	0.80	8815
d av	g	0.91	0.91	0.91	8815
	o aw	pred 1 2 3 4	precision 1	1 0.97 0.94 2 0.91 0.88 3 0.79 0.89 4 0.71 0.74 5 0.56 0.60 0 avg 0.91 0.91 0 avg 0.79 0.81	precision recall f1-score 1 0.97 0.94 0.96 2 0.91 0.88 0.90 3 0.79 0.89 0.84 4 0.71 0.74 0.73 5 0.56 0.60 0.58 0 avg 0.91 0.91 0.91 0 avg 0.79 0.81 0.80



Gradient Boost

Random Fo	orest	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]))
```

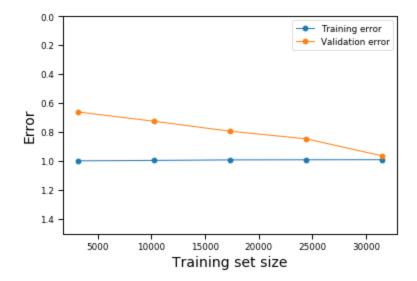
```
Best Random Forest model score: 0.959
Best model hyperparameters:
    learning_rate: 0.1
    max_depth: 6
    min_samples_split: 30
    n_estimators: 300
```

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.plot('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
Bus	iness colleagues	Business Colleagues	3	0.1
N	one 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	W	<i>l</i> eights
0	ujours	Always		1		-0.1
J	temps	Most of the Time		2		-0.05
J.	temps	About half the time		3		0.05
30	ouvent	Occasionally		4		0.08
J	amais	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

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

```
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
l don't know	I don't know	5	0.05
In [50]: u=df['26 len(u),u].unique() #26:"What's the most common use of the money y	vou Lend?"	

```
Out[50]: (5, array([3, 2, 1, 4, 5], dtype=int64))
```