DataScienceForBusiness-I

October 26, 2022

1 Data Science for business- Harkkatyö

Kirjoitetaan tähän pythonilla suoritettavat tutkimukset

2 Preparation

[ ]:
```
#Install required libraries
!pip install sklearn
!pip install shap
!pip install xgboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting sklearn
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-
packages (from sklearn) (1.0.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (1.2.0)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (1.21.6)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (1.7.3)
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) … done
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1310
sha256=1689ba4901c47435e47a80f5e246f20ff37e2e9b05d845269aad306ad9adf999
  Stored in directory: /root/.cache/pip/wheels/46/ef/c3/157e41f5ee1372d1be90b09f
74f82b10e391eaacca8f22d33e
Successfully built sklearn
Installing collected packages: sklearn
Successfully installed sklearn-0.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting shap
```

```
  Downloading
shap-0.41.0-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (569 kB)
     |                  | 569 kB 12.7 MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-
packages (from shap) (1.7.3)
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages
(from shap) (0.56.3)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.7/dist-
packages (from shap) (1.5.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from shap) (1.21.6)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.7/dist-
packages (from shap) (21.3)
Collecting slicer==0.0.7
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
(from shap) (1.3.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-
packages (from shap) (1.0.2)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.7/dist-
packages (from shap) (4.64.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging>20.9->shap) (3.0.9)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from numba->shap) (4.13.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
packages (from numba->shap) (57.4.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in
/usr/local/lib/python3.7/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: typing-extensions>=3.6.4 in
/usr/local/lib/python3.7/dist-packages (from importlib-metadata->numba->shap)
(4.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata->numba->shap) (3.9.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-
packages (from pandas->shap) (2022.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.7.3->pandas->shap) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn->shap) (3.1.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.41.0 slicer-0.0.7
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages
(0.90)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from xgboost) (1.21.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages
(from xgboost) (1.7.3)
```

[ ]:
```
import numpy as np
import pandas as pd
import itertools
import shap
import scipy.stats as stats

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.ticker as mtick
import pydotplus

from IPython.display import Image

import sklearn
from sklearn.model_selection import train_test_split  #Data split function
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV

import xgboost as xgb
from xgboost import XGBClassifier
from xgboost import plot_importance

from imblearn.under_sampling import RandomUnderSampler

from collections import Counter

from datetime import datetime, timedelta

from wordcloud import WordCloud
```

[ ]:
```
from google.colab import drive
drive.mount("/content/drive")

kickstarter_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/
 ↪kickstarter_projects.csv")
```

```
Mounted at /content/drive
```

[ ]:
```
data = kickstarter_data.copy(deep=True)
data.head(20)
```

[ ]:
```
            ID                                               Name  \
0   1860890148  Grace Jones Does Not Give A F$#% T-Shirt (limi…
1    709707365                        CRYSTAL ANTLERS UNTITLED MOVIE
2   1703704063                                   drawing for dollars
3       727286                          Offline Wikipedia iPhone app
4   1622952265                                            Pantshirts
5   2089078683                                New York Makes a Book!!
6    830477146                        Web Site for Short Horror Film
7    266044220                        Help me write my second novel.
8   1502297238                             Produce a Play (Canceled)
9    813230527  Sponsor Dereck Blackburn (Lostwars) Artist in …
10   469734648                                      kicey to iceland
11   515267366                                     Crossword Puzzles!
12  1167151653                           Smogr Alert Field Recording
13   177921463                            Icons for your iPhone apps
14  1099226462              Logical Guess Pictures' 2nd Horror Movie!
15  2147219671  You Are Among Friends: a book for the little s…
16  1147015301  "All We Had" Gets Into Cannes -- $10 or More G…
17  1304906577             Accidental to Edinburgh - PHASE 1: AIRFARE
18  1801448924  Accidental to Edinburgh - PHASE 1: REBUILDING …
19   901991585                                        Produce My Play

       Category     Subcategory        Country             Launched  \
0       Fashion         Fashion  United States  2009-04-21 21:02:48
1   Film & Video         Shorts  United States  2009-04-23 00:07:53
2           Art    Illustration  United States  2009-04-24 21:52:03
3    Technology        Software  United States  2009-04-25 17:36:21
4       Fashion         Fashion  United States  2009-04-27 14:10:39
5    Journalism      Journalism  United States  2009-04-28 13:55:41
6   Film & Video         Shorts  United States  2009-04-29 02:04:21
7    Publishing         Fiction  United States  2009-04-29 02:58:50
8       Theater         Theater  United States  2009-04-29 04:37:37
9         Music            Rock  United States  2009-04-29 05:26:32
10  Photography     Photography  United States  2009-04-29 06:43:44
11        Games         Puzzles  United States  2009-04-29 13:52:03
12       Design  Graphic Design  United States  2009-04-29 22:08:13
```

```
13    Technology        Software  United States  2009-04-29 23:11:15
14  Film & Video    Film & Video  United States  2009-04-30 01:32:55
15    Publishing      Publishing  United States  2009-04-30 07:14:06
16  Film & Video     Documentary  United States  2009-04-30 22:10:30
17       Theater         Theater  United States  2009-04-30 22:22:43
18       Theater         Theater  United States  2009-04-30 22:23:22
19       Theater         Theater  United States  2009-05-01 05:06:19

      Deadline   Goal  Pledged  Backers       State
0   2009-05-31   1000      625       30      Failed
1   2009-07-20  80000       22        3      Failed
2   2009-05-03     20       35        3  Successful
3   2009-07-14     99      145       25  Successful
4   2009-05-26   1900      387       10      Failed
5   2009-05-16   3000     3329      110  Successful
6   2009-05-29    200       41        3      Failed
7   2009-05-29    500      563       18  Successful
8   2009-06-01    500        0        0    Canceled
9   2009-05-16    300       15        2      Failed
10  2009-06-17    350     1630       31  Successful
11  2009-06-30   1500     2265      163  Successful
12  2009-07-04    640       41        3      Failed
13  2009-06-15    500     1820       98  Successful
14  2009-06-06    500      502       22  Successful
15  2009-07-01    350      750       41  Successful
16  2009-05-20    300       40        4      Failed
17  2009-06-05   6000     6575       24  Successful
18  2009-07-15  10000    10145       27  Successful
19  2009-06-01    500      575       21  Successful
```

[ ]:
```python
print(f"Rows: {data.shape[0]}")
print(f"Columns {data.shape[1]}\n")

print("Dataframe information")
data.info()

print("\nDescriptive statistics for the variables")
data.loc[:,'Goal':'Backers'].describe()
```

```
Rows: 374853
Columns 11

Dataframe information
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374853 entries, 0 to 374852
Data columns (total 11 columns):
 #   Column       Non-Null Count   Dtype
```

```
---  ------        --------------  -----
 0   ID           374853 non-null  int64
 1   Name         374853 non-null  object
 2   Category     374853 non-null  object
 3   Subcategory  374853 non-null  object
 4   Country      374853 non-null  object
 5   Launched     374853 non-null  object
 6   Deadline     374853 non-null  object
 7   Goal         374853 non-null  int64
 8   Pledged      374853 non-null  int64
 9   Backers      374853 non-null  int64
 10  State        374853 non-null  object
dtypes: int64(4), object(7)
memory usage: 31.5+ MB

Descriptive statistics for the variables
```

[ ]:

	Goal	Pledged	Backers
count	3.748530e+05	3.748530e+05	374853.000000
mean	4.586378e+04	9.121073e+03	106.690359
std	1.158778e+06	9.132054e+04	911.718520
min	0.000000e+00	0.000000e+00	0.000000
25%	2.000000e+03	3.100000e+01	2.000000
50%	5.500000e+03	6.250000e+02	12.000000
75%	1.600000e+04	4.051000e+03	57.000000
max	1.663614e+08	2.033899e+07	219382.000000

3 Data modelling

[ ]:

```
data.loc[data["Goal"] > 0, 'goal_percent'] = (data["Pledged"] / data["Goal"]).
 ↪round(2)
data.loc[data["Goal"] <= 0, 'goal_percent'] = 1 #How much of the goal was␣
 ↪funded, if goal > 0, result, else 1

data.loc[data["Backers"] > 0, "Pledged_per_Backer"] = (data["Pledged"] /␣
 ↪data["Backers"]).round(2)
data.loc[data["Backers"] <= 0, "Pledged_per_Backer"] = 0 # Money pledged per␣
 ↪backer, if backers > 0, function, else return 0

data["Launched"] = pd.to_datetime(data["Launched"], dayfirst=True)
data["Deadline"] = pd.to_datetime(data["Deadline"], dayfirst=True)
data["Duration"] = data["Deadline"] - data["Launched"]
data["Duration_float"] = (data["Duration"].dt.total_seconds().astype(float))/
 ↪86400
data["name_length"] = data["Name"].str.len()
```

```python
#data["Launchtime"] = pd.DatetimeIndex.dayofyear(data["Launched"]) pd.
↪DatetimeIndex(dates).year
#data["Endtime"] = pd.DatetimeIndex.dayofyear(data["Deadline"])

data.head(10)
```

[ ]:

```
           ID                                               Name  \
0  1860890148  Grace Jones Does Not Give A F$#% T-Shirt (limi…
1   709707365                         CRYSTAL ANTLERS UNTITLED MOVIE
2  1703704063                                    drawing for dollars
3      727286                           Offline Wikipedia iPhone app
4  1622952265                                            Pantshirts
5  2089078683                                  New York Makes a Book!!
6   830477146                        Web Site for Short Horror Film
7   266044220                        Help me write my second novel.
8  1502297238                              Produce a Play (Canceled)
9   813230527  Sponsor Dereck Blackburn (Lostwars) Artist in …

       Category   Subcategory        Country             Launched   Deadline  \
0       Fashion       Fashion  United States 2009-04-21 21:02:48 2009-05-31
1  Film & Video        Shorts  United States 2009-04-23 00:07:53 2009-07-20
2           Art  Illustration  United States 2009-04-24 21:52:03 2009-05-03
3    Technology      Software  United States 2009-04-25 17:36:21 2009-07-14
4       Fashion       Fashion  United States 2009-04-27 14:10:39 2009-05-26
5    Journalism    Journalism  United States 2009-04-28 13:55:41 2009-05-16
6  Film & Video        Shorts  United States 2009-04-29 02:04:21 2009-05-29
7    Publishing       Fiction  United States 2009-04-29 02:58:50 2009-05-29
8       Theater       Theater  United States 2009-04-29 04:37:37 2009-06-01
9         Music          Rock  United States 2009-04-29 05:26:32 2009-05-16

    Goal  Pledged  Backers       State  goal_percent  Pledged_per_Backer  \
0   1000      625       30      Failed          0.62               20.83
1  80000       22        3      Failed          0.00                7.33
2     20       35        3  Successful          1.75               11.67
3     99      145       25  Successful          1.46                5.80
4   1900      387       10      Failed          0.20               38.70
5   3000     3329      110  Successful          1.11               30.26
6    200       41        3      Failed          0.20               13.67
7    500      563       18  Successful          1.13               31.28
8    500        0        0    Canceled          0.00                0.00
9    300       15        2      Failed          0.05                7.50

          Duration  Duration_float  name_length
0 39 days 02:57:12       39.123056           59
1 87 days 23:52:07       87.994525           30
2  8 days 02:07:57        8.088854           19
3 79 days 06:23:39       79.266424           28
```

```
4 28 days 09:49:21       28.409271          10
5 17 days 10:04:19       17.419664          23
6 29 days 21:55:39       29.913646          30
7 29 days 21:01:10       29.875810          30
8 32 days 19:22:23       32.807211          25
9 16 days 18:33:28       16.773241          76
```

Lets check the outcome states of the project

4 Data preparation

Preparing the data, drop nulls, and rows where goal = 0

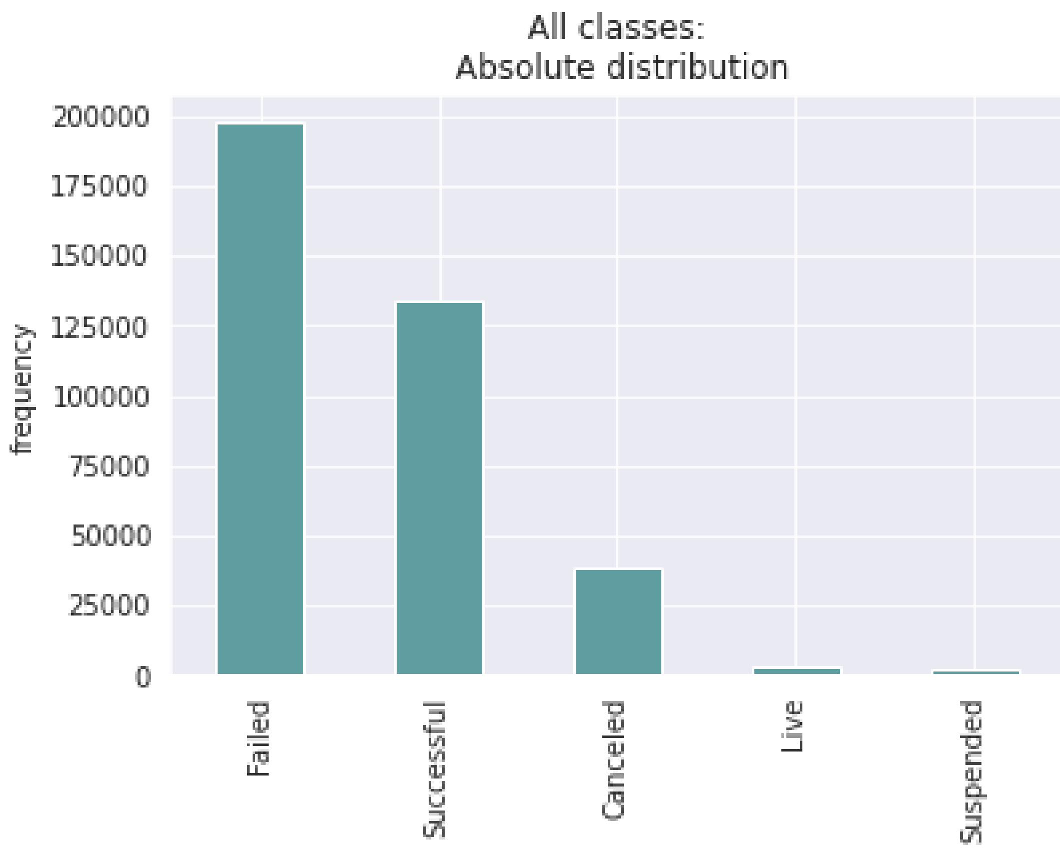
Looking at the variables: * ID can be dropped * Variables 1-7 are predicting variables (features?) * New predicting variables can be formed: length of the crowdfunding period, length of name etc. * Variables 8-10 are outcomes, of which 10 (State) is the most crucial * We could take from days the time of year (what month should you release your project -> remove year and clock) and general time (if the success has increased/decreased recently)

[ ]:
```
df = data #df is data with filtering

#dropataan seuraavat:
#3. NAN arvot syyniin --> poistettu rivit, joilla vain NULL
#4. Live ja suspended poistetaan --> poistettu rivit, joilla state == Live tai␣
↪Suspended
#5. tarkistetaan duplikaatit --> Tehty, ei duplikaatteja

df.drop(['ID', 'Subcategory', 'Launched', 'Deadline', 'Duration', 'Name'], axis␣
↪= 1, inplace=True)
df.drop_duplicates(inplace=True)
df.head(10)
```

[ ]:
```
        Category         Country   Goal  Pledged  Backers       State  \
0        Fashion   United States   1000      625       30      Failed
1   Film & Video   United States  80000       22        3      Failed
2            Art   United States     20       35        3  Successful
3     Technology   United States     99      145       25  Successful
4        Fashion   United States   1900      387       10      Failed
5     Journalism   United States   3000     3329      110  Successful
6   Film & Video   United States    200       41        3      Failed
7     Publishing   United States    500      563       18  Successful
8        Theater   United States    500        0        0    Canceled
9          Music   United States    300       15        2      Failed

   goal_percent  Pledged_per_Backer  Duration_float  name_length
0          0.62               20.83       39.123056           59
1          0.00                7.33       87.994525           30
```

2	1.75	11.67	8.088854	19
3	1.46	5.80	79.266424	28
4	0.20	38.70	28.409271	10
5	1.11	30.26	17.419664	23
6	0.20	13.67	29.913646	30
7	1.13	31.28	29.875810	30
8	0.00	0.00	32.807211	25
9	0.05	7.50	16.773241	76

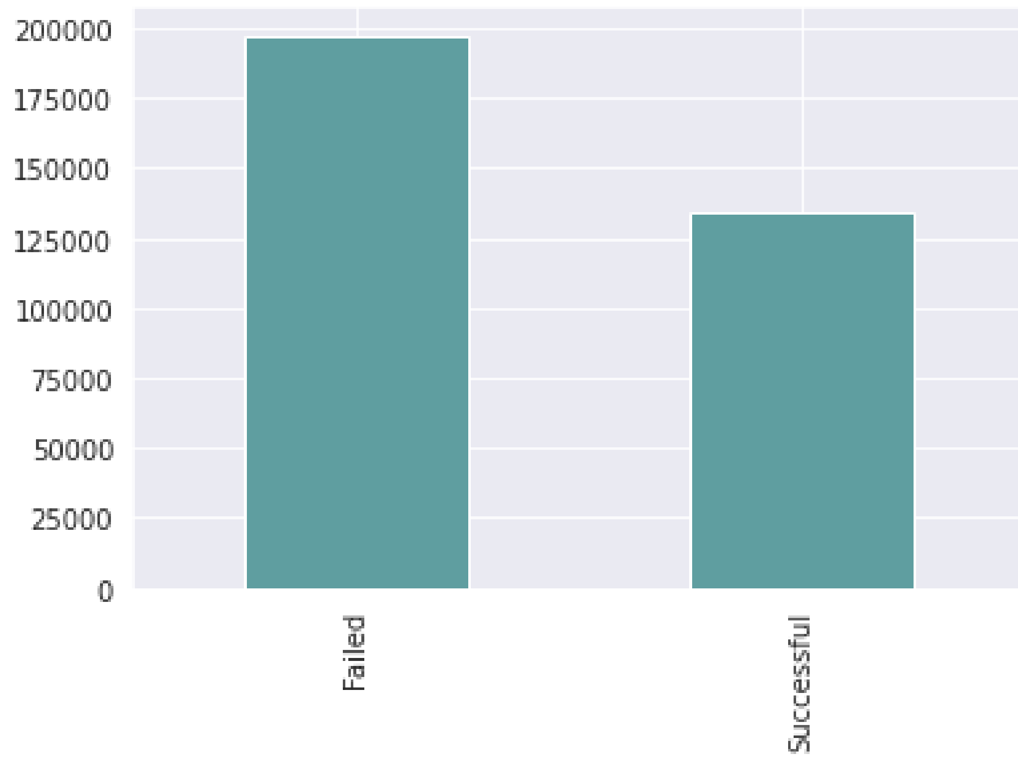
[ ]:
```
# Create here graphs to see if data is balanced or imbalanced
ax = df['State'].value_counts().plot(kind='bar', color='#5F9EA0')

ax0 = plt.title('All classes:\n Absolute distribution')
ax0 = plt.ylabel('frequency')
```



[ ]:
```
#dataCloud = WordCloud(background_color="white").generate(' '.join(df['Name']))
# Create a figure of the generated cloud
#plt.imshow(dataCloud, interpolation='bilinear')
#plt.axis('off')
# Display the figure
```

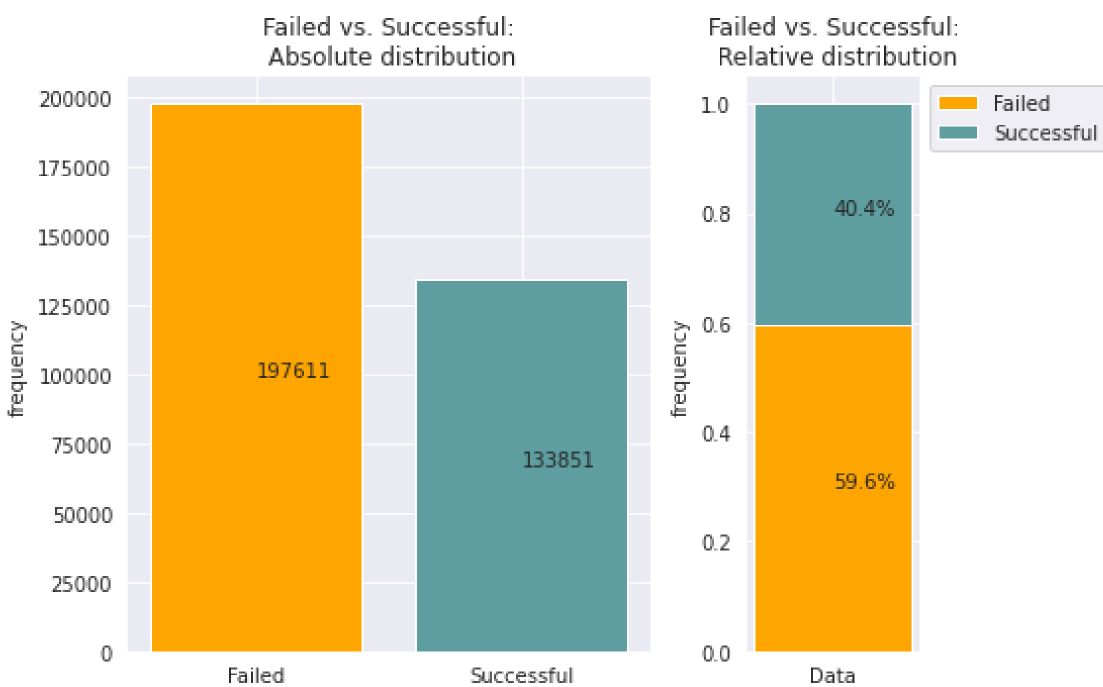
```
#plt.show()
```

[ ]:
```
#Deleting the irrelevant states
df = df[ df['State'].isin(['Failed', 'Successful'])]
ax = df['State'].value_counts().plot(kind='bar', color='#5F9EA0')
```



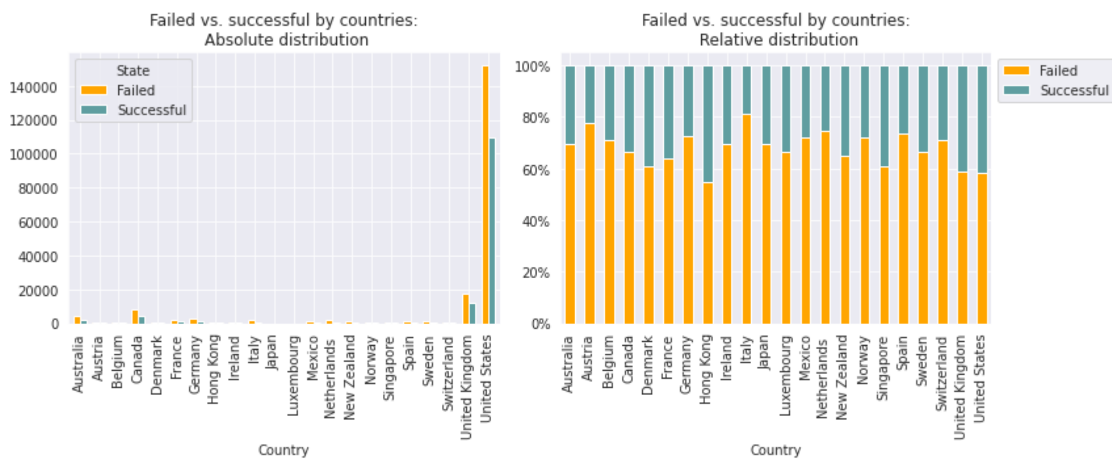
[ ]:
```
keys = np.unique(str(df.State))
counts = df['State'].value_counts()
counts_norm = counts/counts.sum()

fig = plt.figure(figsize=(8, 5)) #specify figure size
gs = gridspec.GridSpec(1, 2, width_ratios=[3,1]) #specify relative size of left␣
 ↪and right plot

ax0 = plt.subplot(gs[0])
ax0 = plt.bar(['Failed', 'Successful'], counts, color=['#FFA500','#5F9EA0'])␣
 ↪#left bar plot
ax0 = plt.title('Failed vs. Successful:\n Absolute distribution')
ax0 = plt.ylabel('frequency')
ax0 = plt.text(['Failed'], counts[0]/2, counts[0])
ax0 = plt.text(['Successful'], counts[1]/2, counts[1])
```

```
#Normalized values
ax1 = plt.subplot(gs[1])
ax1 = plt.bar(['Data'], [counts_norm[0]], label='Failed',  color=['#FFA500'])
ax1 = plt.bar(['Data'], [counts_norm[1]], bottom=counts_norm[0],␣
↪label='Successful', color=['#5F9EA0'])
ax1 = plt.legend(bbox_to_anchor=(1, 1))
ax1 = plt.title('Failed vs. Successful:\n Relative distribution')
ax1 = plt.ylabel('frequency')
ax1 = plt.text(['Data'],counts_norm[0]/2, '{}%'.format((counts_norm[0]*100).
↪round(1)))
ax1 = plt.text(['Data'],(counts_norm[1]/2)+counts_norm[0], '{}%'.
↪format((counts_norm[1]*100).round(1)))


plt.tight_layout()
plt.show()
```



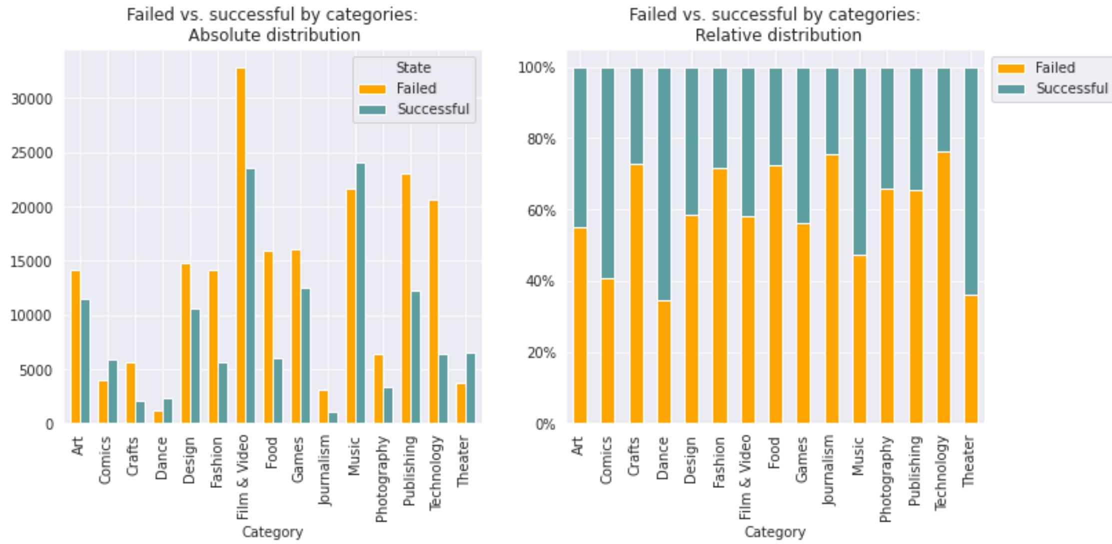
[ ]:
```
#frauds by incident cause
fig = plt.figure(figsize=(12, 5)) #specify figure size

#Absolute distribution
plt.subplot(1, 2, 1)
ax1 = df.groupby(['Country', 'State'])['State'].count().unstack().plot.
↪bar(rot=90, ax=plt.gca(), width=0.7, color=['#FFA500','#5F9EA0'])
```

```
plt.title('Failed vs. successful by countries:\n Absolute distribution')

#Relative distribution
plt.subplot(1, 2, 2)
ax2 = df.groupby(['Country','State'])['State'].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()).unstack().plot(kind='bar',stacked=True,␣
 ↪rot=90, ax=plt.gca(), color=['#FFA500','#5F9EA0'])
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.legend(bbox_to_anchor=(1, 1))
plt.title('Failed vs. successful by countries:\n Relative distribution')

plt.tight_layout()
plt.show()
```

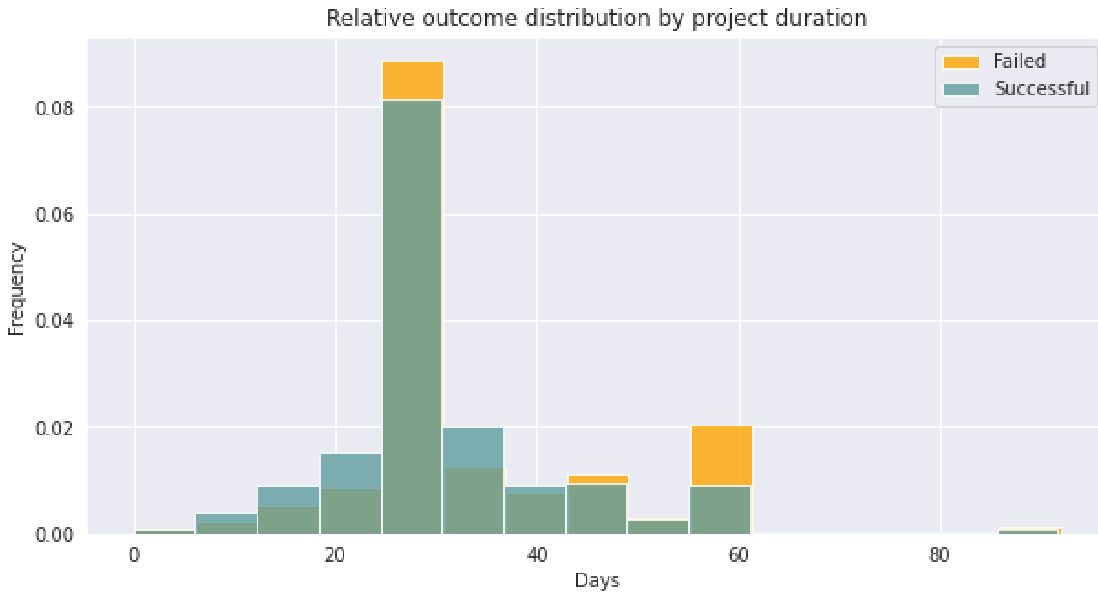


[ ]:
```
#frauds by incident cause
fig = plt.figure(figsize=(12, 5)) #specify figure size

#Absolute distribution
plt.subplot(1, 2, 1)
ax1 = df.groupby(['Category', 'State'])['State'].count().unstack().plot.
 ↪bar(rot=90, ax=plt.gca(), width=0.7, color=['#FFA500','#5F9EA0'])
plt.title('Failed vs. successful by categories:\n Absolute distribution')
#plot bar labels
#for p in ax1.patches:
#    ax1.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() * 0.
 ↪605))

#Relative distribution
plt.subplot(1, 2, 2)
ax2 = df.groupby(['Category','State'])['State'].size().groupby(level=0).apply(
```

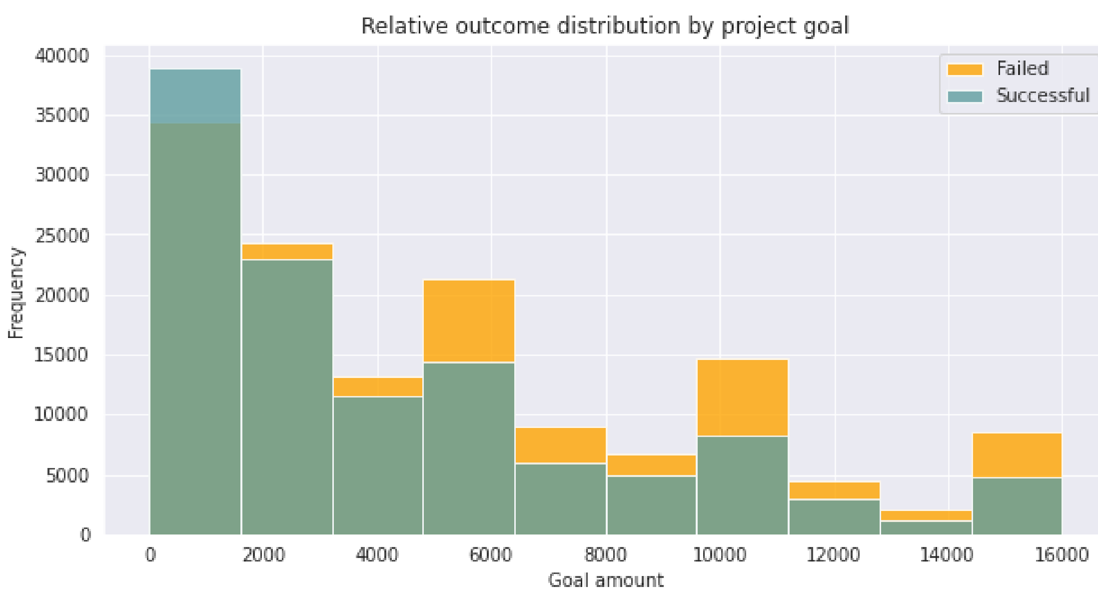
```
    lambda x: 100 * x / x.sum()).unstack().plot(kind='bar',stacked=True,␣
 ↪rot=90, ax=plt.gca(), color=['#FFA500','#5F9EA0'])
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.legend(bbox_to_anchor=(1, 1))
plt.title('Failed vs. successful by categories:\n Relative distribution')
#plot bar labels
#for p, q in zip(ax2.patches[0:5], ax2.patches[5:10]):
#    ax2.annotate(str(round(p.get_height(),1)) + '%', (p.get_x(), p.
 ↪get_height()/2))
#    ax2.annotate(str(round(q.get_height(),1)) + '%', (q.get_x(), q.
 ↪get_height()/2+p.get_height()))

#plt.tight_layout()
plt.show()
```



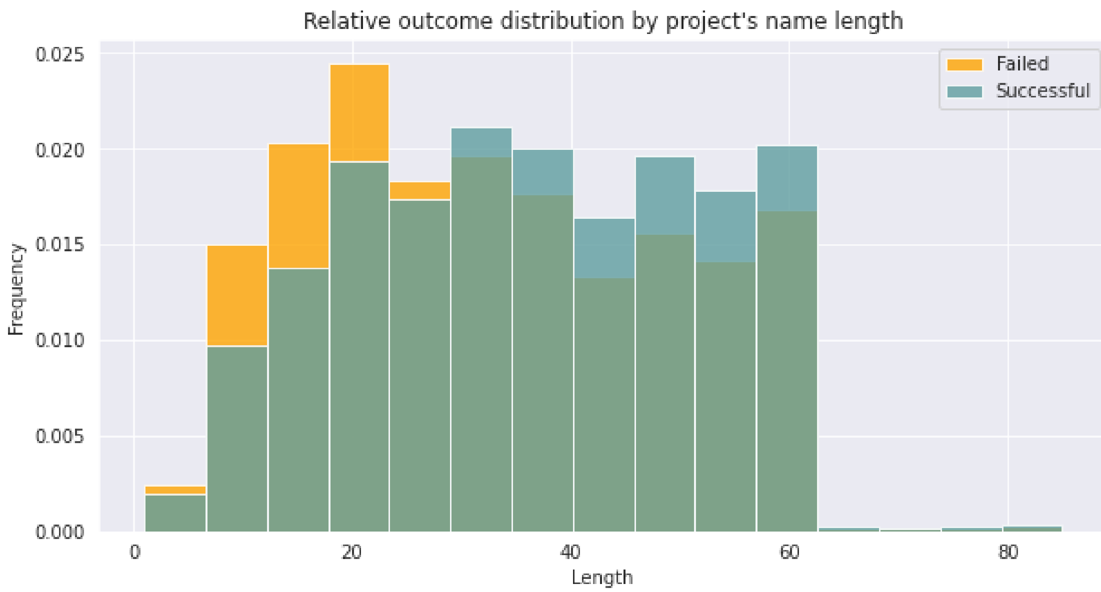
[ ]:
```
ax = df['Duration_float'].loc[df['State'] == 'Failed'].plot.hist(bins=15,␣
 ↪density = True, alpha=0.8, label='Failed', figsize = (10,5),␣
 ↪color=['#FFA500'])
ax = df['Duration_float'].loc[df['State'] == 'Successful'].plot.hist(bins=15,␣
 ↪density = True, alpha=0.8, label='Successful', figsize = (10,5),␣
 ↪color=['#5F9EA0'])
ax.set_xlabel('Days')
ax.set_title('Relative outcome distribution by project duration')
ax.legend();
```




```
[ ]: ax = df['Goal'].loc[df['State'] == 'Failed'].plot.hist(bins=10, alpha=0.8,␣
      ↪label='Failed', range=[0,16000], figsize = (10,5), color=['#FFA500'])
     ax = df['Goal'].loc[df['State'] == 'Successful'].plot.hist(bins=10, alpha=0.8,␣
      ↪label='Successful', range=[0,16000], color=['#5F9EA0'])
     ax.set_xlabel('Goal amount')
     ax.set_title('Relative outcome distribution by project goal')
     ax.legend();
```



[ ]:
```
ax = df['name_length'].loc[df['State'] == 'Failed'].plot.hist(bins=15, density␣
↪= True, alpha=0.8, label='Failed', figsize = (10,5), color=['#FFA500'])
ax = df['name_length'].loc[df['State'] == 'Successful'].plot.hist(bins=15,␣
↪density = True, alpha=0.8, label='Successful', figsize = (10,5),␣
↪color=['#5F9EA0'])
ax.set_xlabel('Length')
ax.set_title("Relative outcome distribution by project's name length")
ax.legend();
```

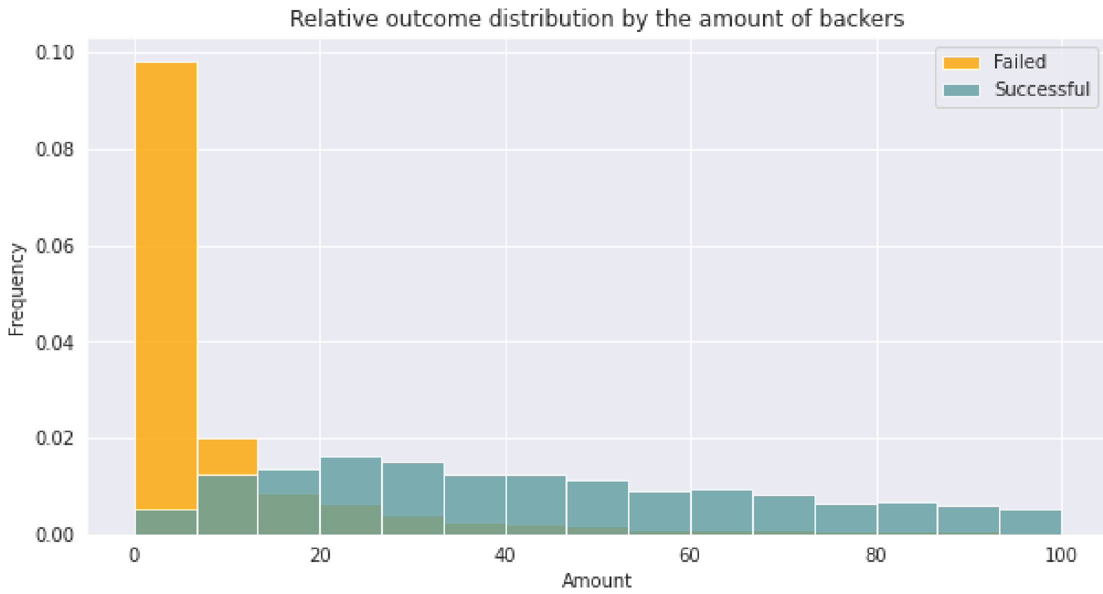


[ ]:
```
df.head()
```

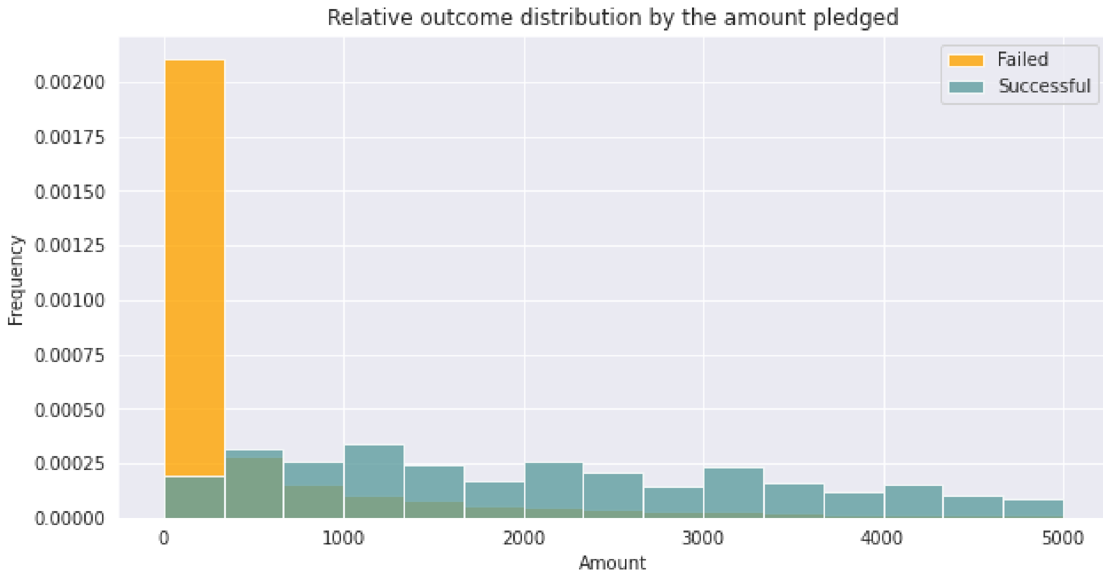
[ ]:
```
        Category        Country   Goal  Pledged  Backers       State  \
0        Fashion  United States   1000      625       30      Failed
1   Film & Video  United States  80000       22        3      Failed
2            Art  United States     20       35        3  Successful
3     Technology  United States     99      145       25  Successful
4        Fashion  United States   1900      387       10      Failed

   goal_percent  Pledged_per_Backer  Duration_float  name_length
0          0.62               20.83       39.123056           59
1          0.00                7.33       87.994525           30
2          1.75               11.67        8.088854           19
3          1.46                5.80       79.266424           28
4          0.20               38.70       28.409271           10
```

[ ]:

```
ax = data['Backers'].loc[data['State'] == 'Failed'].plot.hist(bins=15, density␣
 ↪= True, alpha=0.8, range=[0,100], label='Failed', figsize = (10,5),␣
 ↪color=['#FFA500'])
ax = data['Backers'].loc[data['State'] == 'Successful'].plot.hist(bins=15,␣
 ↪density = True, alpha=0.8, range=[0,100], label='Successful', figsize =␣
 ↪(10,5), color=['#5F9EA0'])
ax.set_xlabel('Amount')
ax.set_title("Relative outcome distribution by the amount of backers")
ax.legend();
```



[ ]:
```
ax = data['Pledged'].loc[data['State'] == 'Failed'].plot.hist(bins=15, density␣
 ↪= True, alpha=0.8, range=[0,5000], label='Failed', figsize = (10,5),␣
 ↪color=['#FFA500'])
ax = data['Pledged'].loc[data['State'] == 'Successful'].plot.hist(bins=15,␣
 ↪density = True, alpha=0.8, range=[0,5000], label='Successful', figsize =␣
 ↪(10,5), color=['#5F9EA0'])
ax.set_xlabel('Amount')
ax.set_title("Relative outcome distribution by the amount pledged")
ax.legend();
```



[ ]:
```
#dataCloud = WordCloud(background_color="white").generate(' '.join(df['Name']))
#Create a figure of the generated cloud
#plt.imshow(dataCloud, interpolation='bilinear')
#plt.axis('off')
#Display the figure
#plt.show()
```

[ ]:
```
df = df.copy(deep=True)
```

[ ]:
```
#Find null values
print(df.isnull().sum())
#delete null values where name_length null
df =df[~df['name_length'].isnull()]

print(df.isnull().sum())
```

```
Category              0
Country               0
Goal                  0
Pledged               0
Backers               0
State                 0
goal_percent          0
Pledged_per_Backer    0
Duration_float        0
name_length           0
dtype: int64
Category              0
```

```
Country               0
Goal                  0
Pledged               0
Backers               0
State                 0
goal_percent          0
Pledged_per_Backer    0
Duration_float        0
name_length           0
dtype: int64
```

All variables except ID and Name seems to have Null values. However it seems like all has as many Nulls. Further investigating there exists 861 project that we dont have any information about. All the other projects has all values and no Nulls. –> Maybe we could drop these projects that we dont have any information except the name.

[ ]:
```python
#Finding projects with no goal and removing them
negative_goal = (df['Goal'] <= 0).sum()
print(f'Rows with goal <= 0 and before cleaning: {negative_goal}')

df = df[df['Goal'] > 0 ]

negative_goal = (df['Goal'] <= 0).sum()
print(f'Rows with goal <= 0 and after cleaning: {negative_goal}')
```

```
Rows with goal <= 0 and before cleaning: 3
Rows with goal <= 0 and after cleaning: 0
```

4.1 Feature selection

[ ]:
```python
df.info()
continuous_features = df.loc[:
 ↪,['Goal','Pledged','Backers','goal_percent','Pledged_per_Backer', 'State']]
categorical_features = df.loc[:,['Category','Country','State']]
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 331459 entries, 0 to 374605
Data columns (total 10 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   Category            331459 non-null  object
 1   Country             331459 non-null  object
 2   Goal                331459 non-null  int64
 3   Pledged             331459 non-null  int64
 4   Backers             331459 non-null  int64
 5   State               331459 non-null  object
 6   goal_percent        331459 non-null  float64
 7   Pledged_per_Backer  331459 non-null  float64
 8   Duration_float      331459 non-null  float64
```

```
 9   name_length         331459 non-null  int64
dtypes: float64(3), int64(4), object(3)
memory usage: 27.8+ MB
```

4.2 Categorical features

[ ]:
```
category_state= pd.crosstab(categorical_features['Category'],␣
↪categorical_features['State'])
country_state= pd.
↪crosstab(categorical_features['Country'],categorical_features['State'])
```

[ ]:
```
category_state
```

[ ]:
```
State          Failed  Successful
Category
Art             14130       11508
Comics           4036        5842
Crafts           5703        2115
Dance            1235        2338
Design          14814       10549
Fashion         14181        5593
Film & Video    32890       23612
Food            15969        6085
Games           16002       12518
Journalism       3136        1012
Music           21696       24105
Photography      6384        3305
Publishing      23113       12300
Technology      20613        6433
Theater          3708        6534
```

[ ]:
```
country_state
```

[ ]:
```
State            Failed  Successful
Country
Australia          4606        2010
Austria             378         107
Belgium             371         152
Canada             8236        4134
Denmark             566         360
France             1612         908
Germany            2499         937
Hong Kong           261         216
Ireland             476         207
Italy              1930         439
Japan                16           7
Luxembourg           38          19
```

```
Mexico             1015      395
Netherlands        1794      617
New Zealand         826      448
Norway              420      162
Singapore           276      178
Spain              1381      492
Sweden             1000      509
Switzerland         465      187
United Kingdom    17386    12067
United States    152058   109298
```

[ ]:
```
def printChi2(chi2_result, title):
 ␣
 ↪print(f"---------------------------------------------------{title}-------------------------
  print(f"Chi2 value: {chi2_result[0]}")
  print(f"P-value: {chi2_result[1]}")
  print(f"Degrees of freedom: {chi2_result[2]}")
 ␣
 ↪print("------------------------------------------------------------------------------------
```

[ ]:
```
#Run chi2 tests
category_state_chi2= stats.chi2_contingency(category_state)
country_state_chi2= stats.chi2_contingency(country_state)

printChi2(category_state_chi2, "Category vs State Chi2 test")
print("\n")
printChi2(country_state_chi2, "Country vs State Chi2 test")
```

```
---------------------------------------------------Category vs State Chi2
test-----------------------------------------
Chi2 value: 15425.470013936694
P-value: 0.0
Degrees of freedom: 14
--------------------------------------------------------------------------------
--------------------------------------



---------------------------------------------------Country vs State Chi2
test-----------------------------------------
Chi2 value: 2200.7204390623438
P-value: 0.0
Degrees of freedom: 21
--------------------------------------------------------------------------------
--------------------------------------
```

From the CHI2 test results we can conclude that both categorical variables are valuable to the prediction of state

4.3 Continuous selection

#Data split

[ ]:
```
df.drop(["Backers", "Pledged_per_Backer", "Pledged", "goal_percent"], axis = 1,␣
 ↪inplace=True)
```

[ ]:
```
#Muuttaa Staten 0 - 1 muotoon --> ainakin ROC tarvitsi tätä
cleanup_nums = {"State": {"Failed": 0, "Successful": 1}}
df.replace(cleanup_nums, inplace=True)
df.head()
```

[ ]:
```
       Category        Country   Goal  State  Duration_float  name_length
0       Fashion  United States   1000      0       39.123056           59
1  Film & Video  United States  80000      0       87.994525           30
2           Art  United States     20      1        8.088854           19
3    Technology  United States     99      1       79.266424           28
4       Fashion  United States   1900      0       28.409271           10
```

[ ]:
```
X, y = df.loc[:, df.columns != 'State'], df['State']
X.head()
```

[ ]:
```
       Category        Country   Goal  Duration_float  name_length
0       Fashion  United States   1000       39.123056           59
1  Film & Video  United States  80000       87.994525           30
2           Art  United States     20        8.088854           19
3    Technology  United States     99       79.266424           28
4       Fashion  United States   1900       28.409271           10
```

[ ]:
```
X = pd.get_dummies(X, columns=["Category", "Country"],
                    prefix=["Category", "Country"], drop_first=True) #Myös␣
 ↪Subcategory:n vois ehkä lisätä
X.head()
```

[ ]:
```
    Goal  Duration_float  name_length  Category_Comics  Category_Crafts  \
0   1000       39.123056           59                0                0
1  80000       87.994525           30                0                0
2     20        8.088854           19                0                0
3     99       79.266424           28                0                0
4   1900       28.409271           10                0                0

   Category_Dance  Category_Design  Category_Fashion  Category_Film & Video  \
0               0                0                 1                      0
1               0                0                 0                      1
2               0                0                 0                      0
3               0                0                 0                      0
4               0                0                 1                      0
```

```
   Category_Food  …  Country_Mexico  Country_Netherlands   \
0              0  …               0                    0
1              0  …               0                    0
2              0  …               0                    0
3              0  …               0                    0
4              0  …               0                    0

   Country_New Zealand  Country_Norway  Country_Singapore  Country_Spain  \
0                    0               0                  0              0
1                    0               0                  0              0
2                    0               0                  0              0
3                    0               0                  0              0
4                    0               0                  0              0

   Country_Sweden  Country_Switzerland  Country_United Kingdom  \
0               0                    0                       0
1               0                    0                       0
2               0                    0                       0
3               0                    0                       0
4               0                    0                       0

   Country_United States
0                      1
1                      1
2                      1
3                      1
4                      1

[5 rows x 38 columns]
```

[ ]:
```
X.info()
```

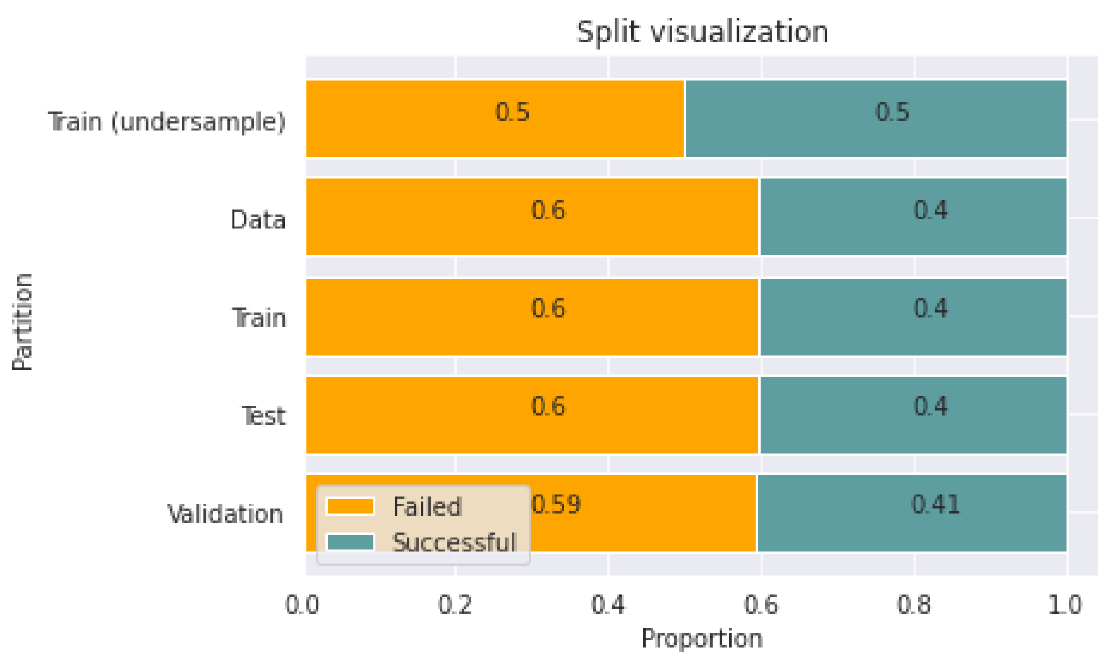
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 331459 entries, 0 to 374605
Data columns (total 38 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Goal                    331459 non-null  int64
 1   Duration_float          331459 non-null  float64
 2   name_length             331459 non-null  int64
 3   Category_Comics         331459 non-null  uint8
 4   Category_Crafts         331459 non-null  uint8
 5   Category_Dance          331459 non-null  uint8
 6   Category_Design         331459 non-null  uint8
 7   Category_Fashion        331459 non-null  uint8
 8   Category_Film & Video   331459 non-null  uint8
 9   Category_Food           331459 non-null  uint8
```

```
 10  Category_Games          331459 non-null  uint8
 11  Category_Journalism     331459 non-null  uint8
 12  Category_Music          331459 non-null  uint8
 13  Category_Photography    331459 non-null  uint8
 14  Category_Publishing     331459 non-null  uint8
 15  Category_Technology     331459 non-null  uint8
 16  Category_Theater        331459 non-null  uint8
 17  Country_Austria         331459 non-null  uint8
 18  Country_Belgium         331459 non-null  uint8
 19  Country_Canada          331459 non-null  uint8
 20  Country_Denmark         331459 non-null  uint8
 21  Country_France          331459 non-null  uint8
 22  Country_Germany         331459 non-null  uint8
 23  Country_Hong Kong       331459 non-null  uint8
 24  Country_Ireland         331459 non-null  uint8
 25  Country_Italy           331459 non-null  uint8
 26  Country_Japan           331459 non-null  uint8
 27  Country_Luxembourg      331459 non-null  uint8
 28  Country_Mexico          331459 non-null  uint8
 29  Country_Netherlands     331459 non-null  uint8
 30  Country_New Zealand     331459 non-null  uint8
 31  Country_Norway          331459 non-null  uint8
 32  Country_Singapore       331459 non-null  uint8
 33  Country_Spain           331459 non-null  uint8
 34  Country_Sweden          331459 non-null  uint8
 35  Country_Switzerland     331459 non-null  uint8
 36  Country_United Kingdom  331459 non-null  uint8
 37  Country_United States   331459 non-null  uint8
dtypes: float64(1), int64(2), uint8(35)
memory usage: 21.2 MB
```

[ ]:
```
# Split into training, VALIDATION and test data!

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15,␣
 ↪random_state=1234567)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size =␣
 ↪0.175, random_state=7654321)

undersample  = RandomUnderSampler(sampling_strategy='majority',␣
 ↪random_state=100)
X_und, y_und = undersample.fit_resample(X_train, y_train) #applied only to␣
 ↪training
```

[ ]:
```
train_dist = y_train.value_counts() / len(y_train) #normalize absolute count␣
 ↪values for plotting
test_dist = y_test.value_counts() / len(y_test)
```

```
df_dist = df['State'].value_counts() / len(df)
valid_dist = y_val.value_counts() / len(y_val)
und_dist = pd.Series(y_und).value_counts() / len(pd.Series(y_und))

fig, ax = plt.subplots()

ax.barh(['Validation','Test','Train','Data', 'Train (undersample)'],␣
↪[valid_dist[0], test_dist[0], train_dist[0], df_dist[0], und_dist[0]],␣
↪color='#FFA500', label='Failed')
ax.barh(['Validation','Test','Train','Data', 'Train (undersample)'],␣
↪[valid_dist[1], test_dist[1], train_dist[1], df_dist[1], und_dist[1]],␣
↪left=[valid_dist[0], test_dist[0], train_dist[0], df_dist[0], und_dist[0]],␣
↪color='#5F9EA0', label='Successful')
ax.set_title('Split visualization')
ax.legend(loc='lower left')
plt.xlabel('Proportion')
plt.ylabel('Partition')

#plot bar values
for part, a, b in zip(['Validation', 'Test', 'Train','Data', 'Train␣
↪(undersample)'], [valid_dist[0], test_dist[0], train_dist[0], df_dist[0],␣
↪und_dist[0]], [valid_dist[1],test_dist[1], train_dist[1], df_dist[1],␣
↪und_dist[1]]):
    plt.text(a/2, part, str(np.round(a, 2)))
    plt.text(b/2+a, part, str(np.round(b, 2)));
```



5 Modeling

5.1 Baseline model - Logistic Regression

[ ]:
```
Xb, yb = df[['Goal', 'Duration_float', 'name_length']], df['State']
Xb.head()
```

[ ]:
```
    Goal  Duration_float  name_length
0   1000       39.123056           59
1  80000       87.994525           30
2     20        8.088854           19
3     99       79.266424           28
4   1900       28.409271           10
```

[ ]:
```
Xb_train, Xb_test, yb_train, yb_test = train_test_split(Xb, yb, test_size = 0.
 ↪15, random_state=1234567)
Xb_train, Xb_val, yb_train, yb_val = train_test_split(Xb_train, yb_train,␣
 ↪test_size = 0.175, random_state=1234567)
```

[ ]:
```
log_reg_baseline = LogisticRegression().fit(Xb_train,yb_train)
```

[ ]:
```
yb_reg_probs = log_reg_baseline.predict_proba(Xb_val)
yb_reg_predicted = log_reg_baseline.predict(Xb_val)
print ("Accuracy is: ", (accuracy_score(yb_val,yb_reg_predicted)*100).round(2))
```

```
Accuracy is:  61.08
```

[ ]:
```
print(classification_report(y_val, yb_reg_predicted))
```

```
              precision    recall  f1-score   support

           0       0.59      0.80      0.68     29238
           1       0.39      0.19      0.26     20067

    accuracy                           0.55     49305
   macro avg       0.49      0.49      0.47     49305
weighted avg       0.51      0.55      0.51     49305
```

[ ]:
```
fpr, tpr, thresholds = roc_curve(yb_val, yb_reg_probs[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Validation: " + str(roc_auc))
```

```
AUC score on Validation: 0.6396080203167409
```

5.2 Decision tree

[ ]:
```
clf = tree.DecisionTreeClassifier(max_depth=5).fit(X_train, y_train)
```

[ ]:
```
clf_bal = tree.DecisionTreeClassifier(max_depth=5).fit(X_und, y_und)
```

5.3 Random forest

[ ]:
```
rf = RandomForestClassifier().fit(X_train, y_train)
```

[ ]:
```
rf_bal = RandomForestClassifier().fit(X_und, y_und)
```

5.4 Logistic regression

[ ]:
```
log_reg = LogisticRegression().fit(X_train,y_train)
```

[ ]:
```
log_reg_bal = LogisticRegression().fit(X_und,y_und)
```

5.5 XGBoost

[ ]:
```
xg = XGBClassifier().fit(X_train, y_train)
```

[ ]:
```
xg_bal = XGBClassifier().fit(X_und, y_und)
```

6 Evaluation / analysis

6.1 Imbalanced tree

[ ]:
```
y_pred = clf.predict(X_val)
y_probs = clf.predict_proba(X_val)
print ("Accuracy is: ", (accuracy_score(y_val,y_pred)*100).round(2))
```

```
Accuracy is:  63.22
```

[ ]:
```
y_pred_train = clf.predict(X_train)
y_probs_train = clf.predict_proba(X_train)
print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,clf.
 ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  63.42
```

[ ]:
```
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.89      0.74     29238
           1       0.62      0.25      0.36     20067
```

```
    accuracy                           0.63     49305
   macro avg       0.63      0.57      0.55     49305
weighted avg       0.63      0.63      0.59     49305
```

[ ]:
```python
print(classification_report(y_train, y_pred_train))
```

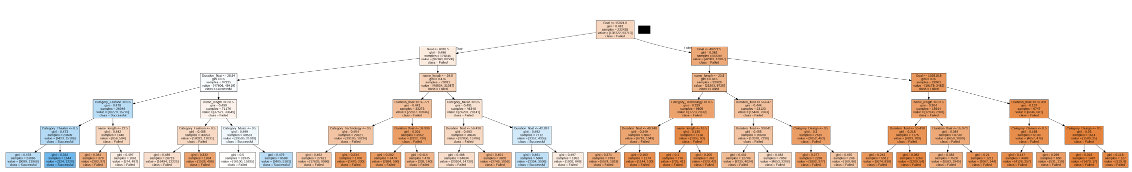
```
              precision    recall  f1-score   support

           0       0.64      0.89      0.74    138722
           1       0.61      0.25      0.36     93713

    accuracy                           0.63    232435
   macro avg       0.63      0.57      0.55    232435
weighted avg       0.63      0.63      0.59    232435
```

[ ]:
```python
fpr, tpr, thresholds = roc_curve(y_val, y_probs[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Validation: " + str(roc_auc))
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_probs_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)
print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.66076644131413 48
AUC score on Training: 0.663352195929738
```

[ ]:
```python
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=X_train.columns,
                                class_names=['Failed', 'Successful'],␣
 ↪filled=True) #or use y_train.unique()

graph = pydotplus.graph_from_dot_data(dot_data)

Image(graph.create_png())
```

[ ]:



6.2 Balanced tree

[ ]:
```
y_pred_bal = clf_bal.predict(X_val)
y_probs_bal = clf_bal.predict_proba(X_val)
print("Accuracy is: ", (accuracy_score(y_val, y_pred_bal)*100).round(2))
```

```
Accuracy is:  58.34
```

[ ]:
```
y_pred_bal_train = clf_bal.predict(X_train)
y_probs_bal_train = clf_bal.predict_proba(X_train)
print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,clf_bal.
 ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  58.55
```

[ ]:
```
print(classification_report(y_val, y_pred_bal))
```

```
              precision    recall  f1-score   support

           0       0.73      0.47      0.57     29238
           1       0.49      0.75      0.60     20067

    accuracy                           0.58     49305
   macro avg       0.61      0.61      0.58     49305
weighted avg       0.64      0.58      0.58     49305
```

[ ]:
```
print(classification_report(y_train, y_pred_bal_train))
```

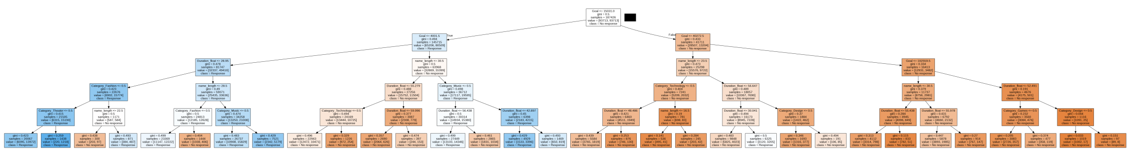
```
              precision    recall  f1-score   support

           0       0.74      0.47      0.58    138722
           1       0.49      0.75      0.59     93713

    accuracy                           0.59    232435
   macro avg       0.62      0.61      0.59    232435
weighted avg       0.64      0.59      0.58    232435
```

[ ]:
```
fpr, tpr, thresholds = roc_curve(y_val, y_probs_bal[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Validation: " + str(roc_auc))
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_probs_bal_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)
print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.6604756197867864
AUC score on Training: 0.6631302503650849
```

[ ]:
```
dot_data = tree.export_graphviz(clf_bal, out_file=None, feature_names=X_und.
 ↪columns, class_names=['No response', 'Response'], filled=True)

graph = pydotplus.graph_from_dot_data(dot_data)

Image(graph.create_png())
```

[ ]:



6.3 Random forest

[ ]:
```
y_rf_probs = rf.predict_proba(X_val)
y_rf_predicted = rf.predict(X_val)
print ("Accuracy is: ", (accuracy_score(y_val,y_rf_predicted)*100).round(2))
```

```
Accuracy is:  64.05
```

[ ]:
```
y_rf_probs_train = rf.predict_proba(X_train)
y_rf_predicted_train = rf.predict(X_train)
print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,rf.
 ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  100.0
```

[ ]:
```
print(classification_report(y_val, y_rf_predicted))
```

```
              precision    recall  f1-score   support

           0       0.68      0.74      0.71     29238
           1       0.57      0.49      0.53     20067

    accuracy                           0.64     49305
   macro avg       0.62      0.62      0.62     49305
weighted avg       0.63      0.64      0.64     49305
```

[ ]:
```
print(classification_report(y_train, y_rf_predicted_train))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    138722
           1       1.00      1.00      1.00     93713
```

```
    accuracy                           1.00    232435
   macro avg       1.00      1.00      1.00    232435
weighted avg       1.00      1.00      1.00    232435
```

```
[ ]: fpr, tpr, thresholds = roc_curve(y_val, y_rf_probs[:,1])
     roc_auc = auc(fpr, tpr)
     print("AUC score on Validation: " + str(roc_auc))
     fpr_train, tpr_train, thresholds = roc_curve(y_train, y_rf_probs_train[:,1])
     roc_auc_train = auc(fpr_train, tpr_train)
     print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.680920675433583
AUC score on Training: 0.9999998461160331
```

6.4 Random forest balanced

```
[ ]: y_rf_probs_bal = rf_bal.predict_proba(X_val)
     y_rf_predicted_bal = rf_bal.predict(X_val)
     print ("Accuracy is: ", (accuracy_score(y_val,y_rf_predicted_bal)*100).round(2))
```

```
Accuracy is:  62.92
```

```
[ ]: y_rf_probs_bal_train = rf_bal.predict_proba(X_train)
     y_rf_predicted_bal_train = rf_bal.predict(X_train)
     print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,rf_bal.
      ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  92.85
```

```
[ ]: print(classification_report(y_val, y_rf_predicted_bal))
```

```
              precision    recall  f1-score   support

           0       0.71      0.63      0.67     29238
           1       0.54      0.63      0.58     20067

    accuracy                           0.63     49305
   macro avg       0.63      0.63      0.62     49305
weighted avg       0.64      0.63      0.63     49305
```

```
[ ]: print(classification_report(y_train, y_rf_predicted_bal_train))
```

```
              precision    recall  f1-score   support

           0       1.00      0.88      0.94    138722
           1       0.85      1.00      0.92     93713
```

```
    accuracy                           0.93    232435
   macro avg       0.92      0.94      0.93    232435
weighted avg       0.94      0.93      0.93    232435
```

[ ]:
```
fpr, tpr, thresholds = roc_curve(y_val, y_rf_probs_bal[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Validation: " + str(roc_auc))
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_rf_probs_bal_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)
print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.681418063496453
AUC score on Training: 0.9802338403776016
```

6.5 Log reg

[ ]:
```
y_reg_probs = log_reg.predict_proba(X_val)
y_reg_predicted = log_reg.predict(X_val)
print ("Accuracy is: ", (accuracy_score(y_val,y_reg_predicted)*100).round(2))
```

```
Accuracy is:  63.69
```

[ ]:
```
y_reg_probs_train = log_reg.predict_proba(X_train)
y_reg_predicted_train = log_reg.predict(X_train)
print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,log_reg.
 ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  63.94
```

[ ]:
```
print(classification_report(y_val, y_reg_predicted))
```

```
              precision    recall  f1-score   support

           0       0.66      0.80      0.72     29238
           1       0.58      0.39      0.47     20067

    accuracy                           0.64     49305
   macro avg       0.62      0.60      0.60     49305
weighted avg       0.63      0.64      0.62     49305
```

[ ]:
```
print(classification_report(y_train, y_reg_predicted_train))
```

```
              precision    recall  f1-score   support

           0       0.66      0.81      0.73    138722
```

```
           1       0.58      0.39      0.47     93713

    accuracy                           0.64    232435
   macro avg       0.62      0.60      0.60    232435
weighted avg       0.63      0.64      0.62    232435
```

```
[ ]: fpr, tpr, thresholds = roc_curve(y_val, y_reg_probs[:,1])
     roc_auc = auc(fpr, tpr)
     print("AUC score on Validation: " + str(roc_auc))
     fpr_train, tpr_train, thresholds = roc_curve(y_train, y_reg_probs_train[:,1])
     roc_auc_train = auc(fpr_train, tpr_train)
     print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.6719139797472979
AUC score on Training: 0.6736927638129417
```

6.6 Log reg balanced

```
[ ]: y_reg_probs_bal = log_reg_bal.predict_proba(X_val)
     y_reg_pred_bal = log_reg_bal.predict(X_val)
     print ("Accuracy is: ", (accuracy_score(y_val,y_reg_pred_bal)*100).round(2))
```

```
Accuracy is:  59.03
```

```
[ ]: y_reg_probs_bal_train = log_reg_bal.predict_proba(X_train)
     y_reg_pred_bal_train = log_reg_bal.predict(X_train)
     print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,log_reg_bal.
      ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  59.25
```

```
[ ]: print(classification_report(y_val, y_reg_pred_bal))
```

```
              precision    recall  f1-score   support

           0       0.69      0.56      0.62     29238
           1       0.50      0.64      0.56     20067

    accuracy                           0.59     49305
   macro avg       0.59      0.60      0.59     49305
weighted avg       0.61      0.59      0.59     49305
```

```
[ ]: print(classification_report(y_train, y_reg_pred_bal_train))
```

```
              precision    recall  f1-score   support
```

```
           0       0.70      0.56      0.62    138722
           1       0.50      0.64      0.56     93713

    accuracy                           0.59    232435
   macro avg       0.60      0.60      0.59    232435
weighted avg       0.62      0.59      0.60    232435
```

[ ]:
```
fpr, tpr, thresholds = roc_curve(y_val, y_reg_probs_bal[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Validation: " + str(roc_auc))
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_reg_probs_bal_train[:
 ↪,1])
roc_auc_train = auc(fpr_train, tpr_train)
print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.6390786091983469
AUC score on Training: 0.639754919375922
```

6.7 XGBoost

[ ]:
```
xg_proba = xg.predict_proba(X_val)
y_pred_xg = xg.predict(X_val)
print ("Accuracy is: ", (accuracy_score(y_val, y_pred_xg)*100).round(2))
```

```
Accuracy is:  66.98
```

[ ]:
```
xg_proba_train = xg.predict_proba(X_train)
y_pred_xg_train = xg.predict(X_train)
print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,xg.
 ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  67.05
```

[ ]:
```
print(classification_report(y_val, y_pred_xg))
```

```
              precision    recall  f1-score   support

           0       0.68      0.82      0.75     29238
           1       0.63      0.45      0.52     20067

    accuracy                           0.67     49305
   macro avg       0.66      0.63      0.64     49305
weighted avg       0.66      0.67      0.66     49305
```

[ ]:
```
print(classification_report(y_train, y_pred_xg_train))
```

```
              precision    recall  f1-score   support

           0       0.69      0.82      0.75    138722
           1       0.63      0.44      0.52     93713

    accuracy                           0.67    232435
   macro avg       0.66      0.63      0.64    232435
weighted avg       0.66      0.67      0.66    232435
```

```
[ ]: fpr, tpr, thresholds = roc_curve(y_val, xg_proba[:,1])
     roc_auc = auc(fpr, tpr)
     print("AUC score on Validation: " + str(roc_auc))
     fpr_train, tpr_train, thresholds = roc_curve(y_train, xg_proba_train[:,1])
     roc_auc_train = auc(fpr_train, tpr_train)
     print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.7202447848002509
AUC score on Training: 0.7223632951234242
```

6.8 XGBoost balanced

```
[ ]: xg_proba_bal = xg_bal.predict_proba(X_val)
     y_pred_xg_bal = xg_bal.predict(X_val)
     print ("Accuracy is: ", (accuracy_score(y_val, y_pred_xg_bal)*100).round(2))
```

```
Accuracy is:  65.26
```

```
[ ]: xg_proba_train_bal = xg_bal.predict_proba(X_train)
     y_pred_xg_train_bal = xg_bal.predict(X_train)
     print ("Training DATA! Accuracy is: ", (accuracy_score(y_train,xg_bal.
      ↪predict(X_train))*100).round(2))
```

```
Training DATA! Accuracy is:  65.51
```

```
[ ]: print(classification_report(y_val, y_pred_xg_bal))
```

```
              precision    recall  f1-score   support

           0       0.75      0.63      0.68     29238
           1       0.56      0.69      0.62     20067

    accuracy                           0.65     49305
   macro avg       0.65      0.66      0.65     49305
weighted avg       0.67      0.65      0.66     49305
```

```
[ ]: print(classification_report(y_train, y_pred_xg_train_bal))
```

```
              precision    recall  f1-score   support

           0       0.75      0.63      0.69    138722
           1       0.56      0.69      0.62     93713

    accuracy                           0.66    232435
   macro avg       0.66      0.66      0.65    232435
weighted avg       0.67      0.66      0.66    232435
```

[ ]:
```
fpr, tpr, thresholds = roc_curve(y_val, xg_proba_bal[:,1])
roc_auc = auc(fpr, tpr)
print("AUC score on Validation: " + str(roc_auc))

fpr_train, tpr_train, thresholds = roc_curve(y_train, xg_proba_train_bal[:,1])
roc_auc_train = auc(fpr_train, tpr_train)
print("AUC score on Training: " + str(roc_auc_train))
```

```
AUC score on Validation: 0.7201833141416913
AUC score on Training: 0.721878466820486
```

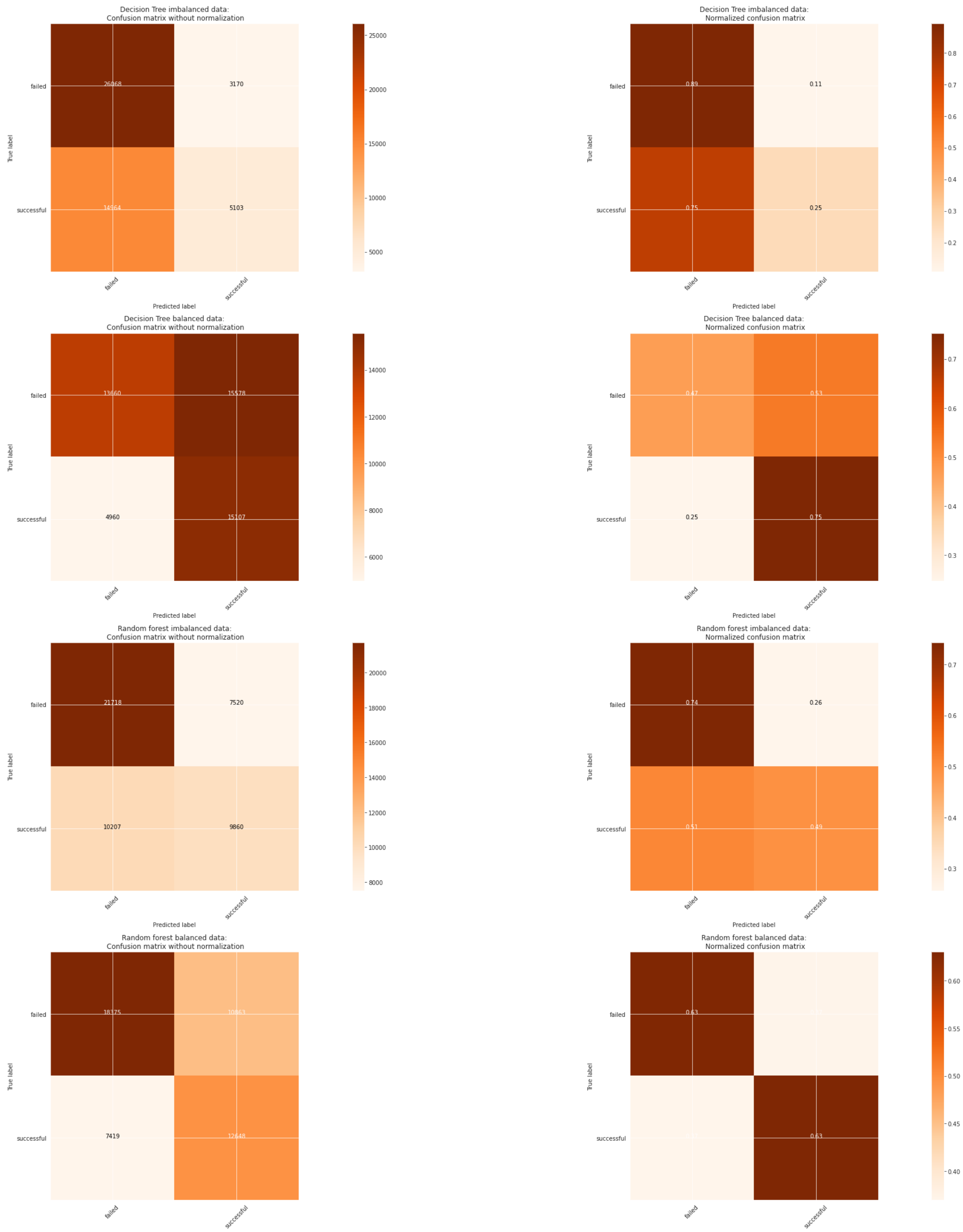
6.9 Confusion matrices

[ ]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Oranges):

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylim([1.5, -0.5]) #added to fix a bug that causes the matrix to be
 ↪squished
```

```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[ ]:
```
class_names = ['failed', 'successful']
DecisionTree_matrix_imb = confusion_matrix(y_val, y_pred)
DecisionTree_matrix_bal = confusion_matrix(y_val, y_pred_bal)
rf_matrix_imb = confusion_matrix(y_val, y_rf_predicted)
rf_matrix_bal = confusion_matrix(y_val, y_rf_predicted_bal)

np.set_printoptions(precision=2)

plt.figure(figsize=(40,30))

plt.subplot(421)
plot_confusion_matrix(DecisionTree_matrix_imb, classes=class_names,␣
 ↪title='Decision Tree imbalanced data:\n Confusion matrix without␣
 ↪normalization')

plt.subplot(422)
plot_confusion_matrix(DecisionTree_matrix_imb, classes=class_names,␣
 ↪normalize=True, title='Decision Tree imbalanced data:\n Normalized confusion␣
 ↪matrix')

plt.subplot(423)
plot_confusion_matrix(DecisionTree_matrix_bal, classes=class_names,␣
 ↪title='Decision Tree balanced data:\n Confusion matrix without␣
 ↪normalization')

plt.subplot(424)
plot_confusion_matrix(DecisionTree_matrix_bal, classes=class_names,␣
 ↪normalize=True, title='Decision Tree balanced data:\n Normalized confusion␣
 ↪matrix')

plt.subplot(425)
plot_confusion_matrix(rf_matrix_imb, classes=class_names, title='Random forest␣
 ↪imbalanced data:\n Confusion matrix without normalization')

plt.subplot(426)
plot_confusion_matrix(rf_matrix_imb, classes=class_names, normalize=True,␣
 ↪title='Random forest imbalanced data:\n Normalized confusion matrix')

plt.subplot(427)
plot_confusion_matrix(rf_matrix_bal, classes=class_names, title='Random forest␣
 ↪balanced data:\n Confusion matrix without normalization')

plt.subplot(428)
```

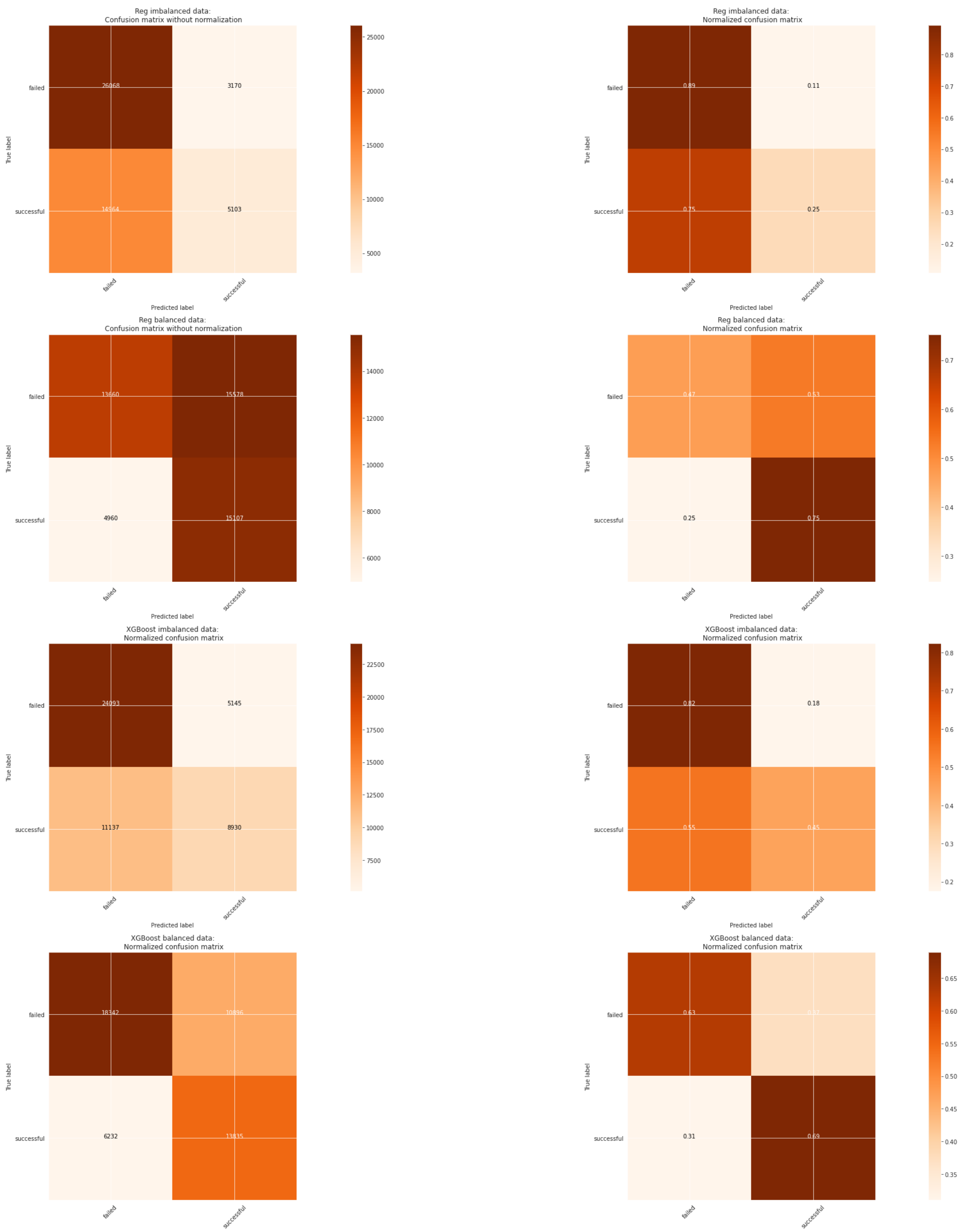
```
plot_confusion_matrix(rf_matrix_bal, classes=class_names, normalize=True,
    title='Random forest balanced data:\n Normalized confusion matrix')

plt.show()
```



[ ]:

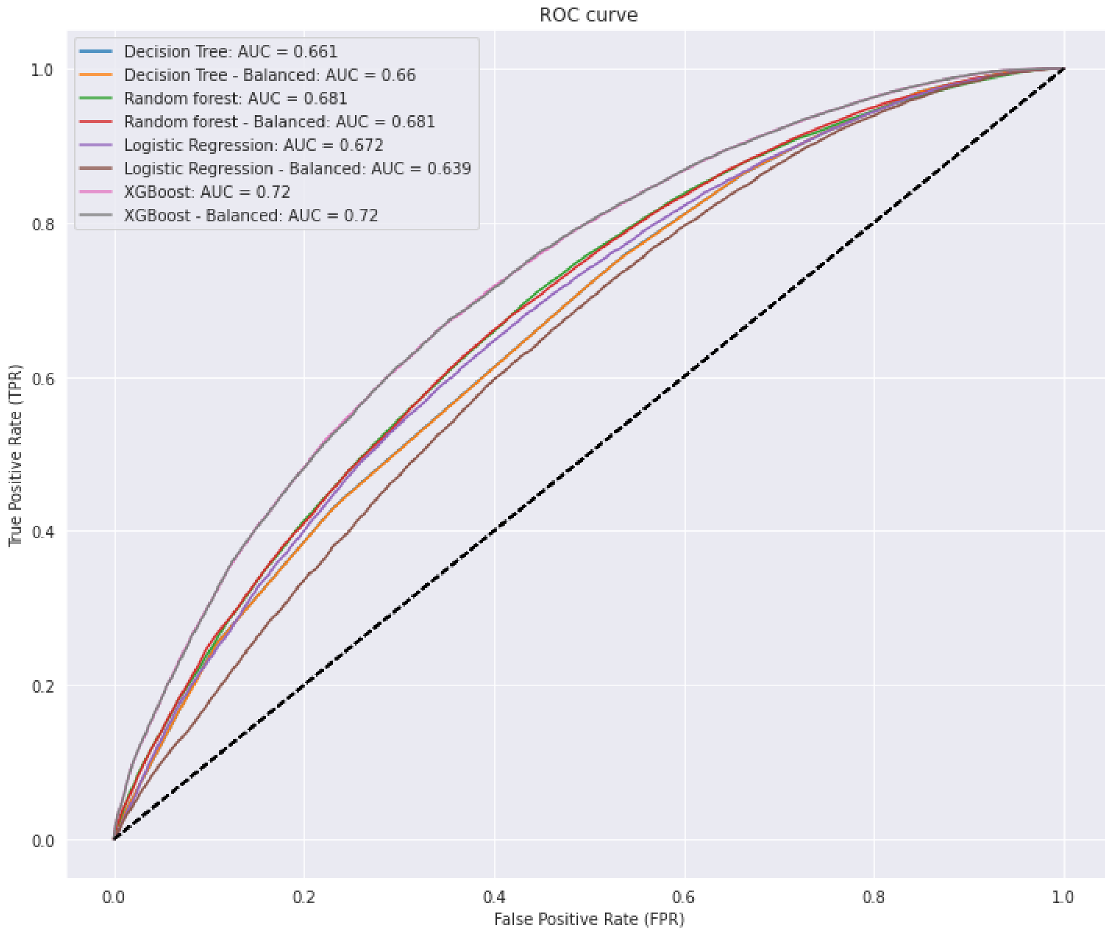
```
# Compute confusion matrix
class_names = ['failed', 'successful']
DecisionTree_matrix_imb = confusion_matrix(y_val, y_pred)
DecisionTree_matrix_bal = confusion_matrix(y_val, y_pred_bal)
cnf_matrix_xg = confusion_matrix(y_val, y_pred_xg)
cnf_matrix_xg_bal = confusion_matrix(y_val, y_pred_xg_bal)

np.set_printoptions(precision=2)

plt.figure(figsize=(40,30))

#Plot imbalanced Reg confusion matrix
plt.subplot(421)
plot_confusion_matrix(DecisionTree_matrix_imb, classes=class_names, title='Reg␣
↪imbalanced data:\n Confusion matrix without normalization')

#Plot imbalanced Reg normalized confusion matrix
plt.subplot(422)
plot_confusion_matrix(DecisionTree_matrix_imb, classes=class_names,␣
↪normalize=True, title='Reg imbalanced data:\n Normalized confusion matrix')

#Plot balanced Reg confusion matrix
plt.subplot(423)
plot_confusion_matrix(DecisionTree_matrix_bal, classes=class_names, title='Reg␣
↪balanced data:\n Confusion matrix without normalization')

#Plot balanced Reg normalized confusion matrix
plt.subplot(424)
plot_confusion_matrix(DecisionTree_matrix_bal, classes=class_names,␣
↪normalize=True, title='Reg balanced data:\n Normalized confusion matrix')

#Plot imbalanced XGB confusion matrix
plt.subplot(425)
plot_confusion_matrix(cnf_matrix_xg, classes=class_names, title='XGBoost␣
↪imbalanced data:\n Normalized confusion matrix')

#Plot imbalanced XGB normalized confusion matrix
plt.subplot(426)
plot_confusion_matrix(cnf_matrix_xg, classes=class_names, normalize=True,␣
↪title='XGBoost imbalanced data:\n Normalized confusion matrix')

#Plot balanced XGB confusion matrix
plt.subplot(427)
plot_confusion_matrix(cnf_matrix_xg_bal, classes=class_names, title='XGBoost␣
↪balanced data:\n Normalized confusion matrix')
```

```
#Plot balanced XGB normalized confusion matrix
plt.subplot(428)
plot_confusion_matrix(cnf_matrix_xg_bal, classes=class_names, normalize=True, title='XGBoost balanced data:\n Normalized confusion matrix')

plt.show()
```



6.10 ROC & AUC

```
[ ]: ## ROC chart & AUC
plt.figure(figsize=(12,10))

for test, pred, name in zip([y_val, y_val, y_val, y_val, y_val, y_val, y_val,␣
 ↪y_val], [y_probs[:,1], y_probs_bal[:,1], y_rf_probs[:,1], y_rf_probs_bal[:
 ↪,1], y_reg_probs[:,1], y_reg_probs_bal[:,1], xg_proba[:,1], xg_proba_bal[:
 ↪,1]], ['Decision Tree', 'Decision Tree - Balanced', 'Random forest', 'Random␣
 ↪forest - Balanced', 'Logistic Regression', 'Logistic Regression - Balanced',␣
 ↪'XGBoost', 'XGBoost - Balanced']):
    fpr, tpr, _ = roc_curve(test, pred)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='{}: AUC = {}'.format(name, round(roc_auc, 3)))
    plt.legend(loc='best')
    plt.plot([0,1],[0,1], color='black', linestyle='--')

plt.title('ROC curve')
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')

plt.show()
```



7 Final model

7.1 Hyperparameter optimization

[ ]:
```
max_depth = [2,3,4,5,6,7]

learning_rate = [0.01, 0.05, 0.1, 0.15, 0.2]

gamma = [0.0, 0.1, 0.2, 0.3]

min_child_weight = [1, 3, 5, 7, 9]
```

[ ]:
```
# Create the param grid
param_grid = {'max_depth': max_depth,
              'learning_rate': learning_rate,
              'gamma': gamma,
              'min_child_weight': min_child_weight,
```

```
            }
print(param_grid)
```

```
{'max_depth': [2, 3, 4, 5, 6, 7], 'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
'gamma': [0.0, 0.1, 0.2, 0.3], 'min_child_weight': [1, 3, 5, 7, 9]}
```

[ ]:
```
xg_RandomGrid = RandomizedSearchCV(estimator = xg, param_distributions =␣
↪param_grid, cv = 3, verbose=1, n_jobs = -1, n_iter = 5, scoring = 'f1',␣
↪random_state=42)
```

[ ]:
```
%%time
xg_RandomGrid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
CPU times: user 33.4 s, sys: 316 ms, total: 33.8 s
Wall time: 3min 47s
```

[ ]:
```
RandomizedSearchCV(cv=3, estimator=XGBClassifier(), n_iter=5, n_jobs=-1,
                   param_distributions={'gamma': [0.0, 0.1, 0.2, 0.3],
                                        'learning_rate': [0.01, 0.05, 0.1, 0.15,
                                                          0.2],
                                        'max_depth': [2, 3, 4, 5, 6, 7],
                                        'min_child_weight': [1, 3, 5, 7, 9]},
                   random_state=42, scoring='f1', verbose=1)
```

[ ]:
```
xg_RandomGrid.best_estimator_
```

[ ]:
```
XGBClassifier(gamma=0.2, learning_rate=0.2, max_depth=5)
```

7.2 Final model evaluation

[ ]:
```
final_model  = xg_RandomGrid.best_estimator_
```

[ ]:
```
final_model_enh_probs = xg_RandomGrid.predict_proba(X_val)
final_model_predicted = xg_RandomGrid.predict(X_val)
print ("Accuracy is: ", (accuracy_score(y_val,final_model_predicted)*100).
↪round(2))
```

```
Accuracy is:  67.53
```

-
-
-
-
-
-
- Final evalution using test data

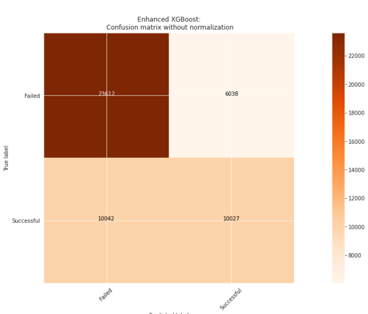
[ ]:
```
final_model_enh_probs_test = xg_RandomGrid.predict_proba(X_test)
final_model_predicted_test = xg_RandomGrid.predict(X_test)
print ("Accuracy is: ", (accuracy_score(y_test,final_model_predicted_test)*100).
↪round(2))
```

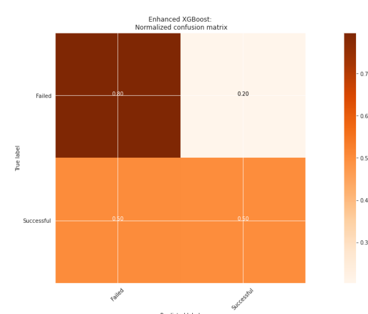
```
Accuracy is:  67.66
```

[ ]:
```
print(classification_report(y_test, final_model_predicted_test))
```

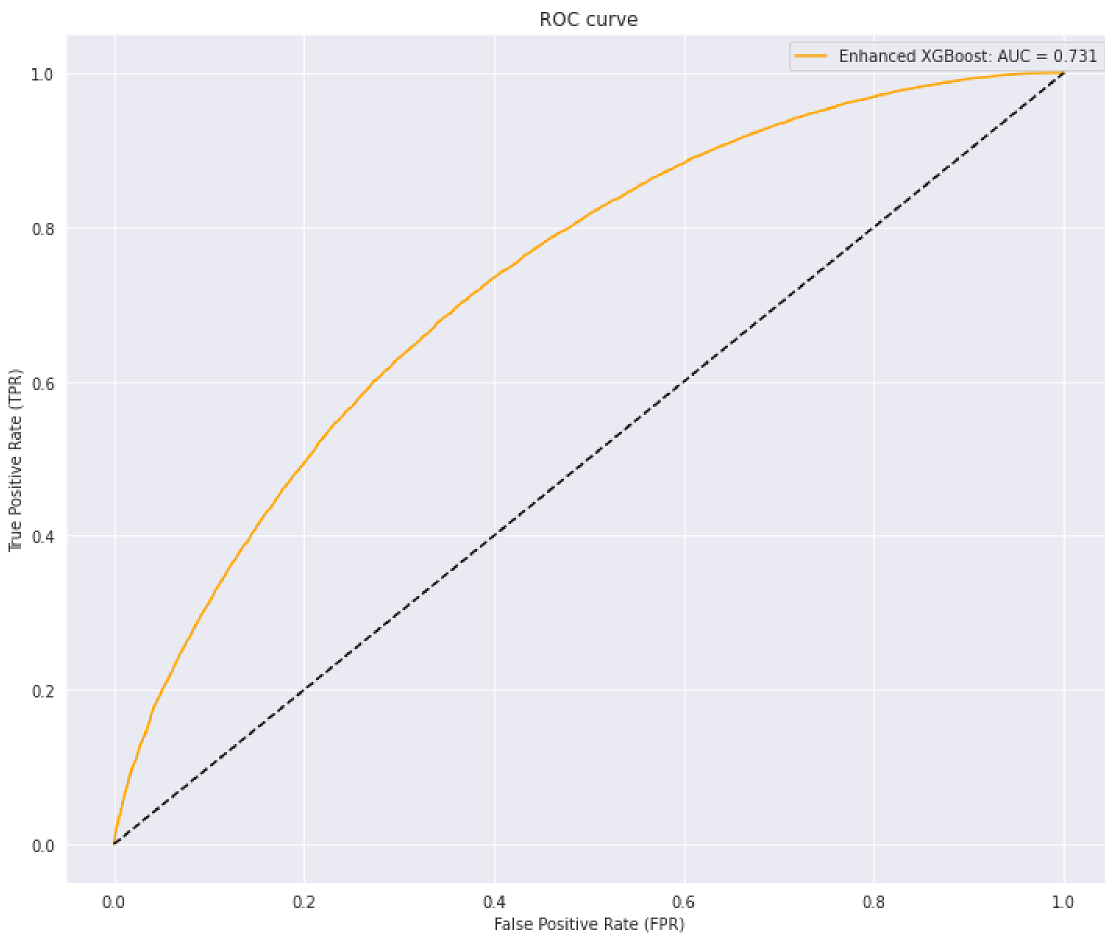
```
              precision    recall  f1-score   support

           0       0.70      0.80      0.75     29650
           1       0.62      0.50      0.55     20069

    accuracy                           0.68     49719
   macro avg       0.66      0.65      0.65     49719
weighted avg       0.67      0.68      0.67     49719
```

[ ]:
```
# Compute confusion matrix
class_names = ['Failed', 'Successful']
Enhanced_RF = confusion_matrix(y_test, final_model_predicted_test)

np.set_printoptions(precision=2)

plt.figure(figsize=(40,30))

#Plot final random forestconfusion matrix
plt.subplot(421)
plot_confusion_matrix(Enhanced_RF, classes=class_names, title='Enhanced XGBoost:
↪\n Confusion matrix without normalization')

#Plot imbalanced Reg normalized confusion matrix
plt.subplot(422)
plot_confusion_matrix(Enhanced_RF, classes=class_names, normalize=True,␣
↪title='Enhanced XGBoost:\n Normalized confusion matrix')

plt.show()
```





```
[ ]: ## ROC chart & AUC
plt.figure(figsize=(12,10))

for test, pred, name in zip([y_test], [final_model_enh_probs_test[:,1]],␣
 ↪['Enhanced XGBoost']):
    fpr, tpr, _ = roc_curve(test, pred)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='{}: AUC = {}'.format(name, round(roc_auc, 3)),␣
 ↪color='#FFA500')
    plt.legend(loc='best')
    plt.plot([0,1],[0,1], color='black', linestyle='--')

plt.title('ROC curve')
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')

plt.show()
```



SHAP

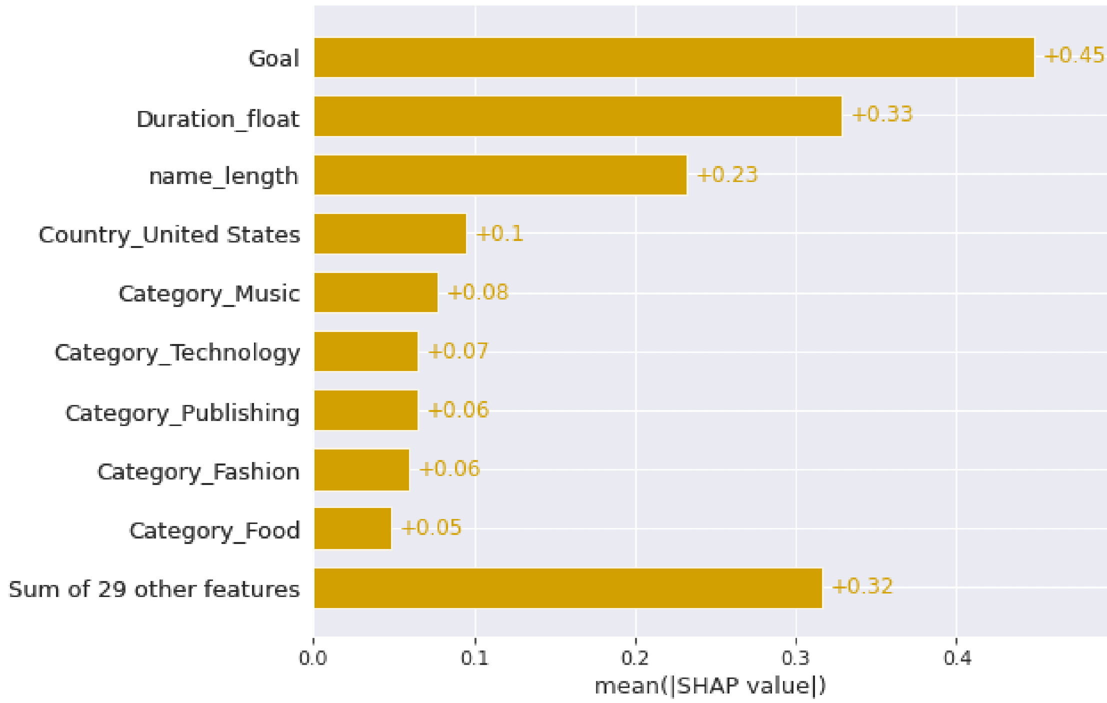
```
[ ]: def change_colors():
         default_pos_color = "#ff0051"
         default_neg_color = "#008bfb"

         # Custom colors
         positive_color = "#D2A000"
         negative_color = "#767171"
         for fc in plt.gcf().get_children():
             # Ignore last Rectangle
             for fcc in fc.get_children()[:-1]:
                 if (isinstance(fcc, matplotlib.patches.Rectangle)):
                     if (matplotlib.colors.to_hex(fcc.get_facecolor()) ==␣
      ↪default_pos_color):
                         fcc.set_facecolor(positive_color)
                     elif (matplotlib.colors.to_hex(fcc.get_facecolor()) ==␣
      ↪default_neg_color):
                         fcc.set_color(negative_color)
                 elif (isinstance(fcc, plt.Text)):
                     if (matplotlib.colors.to_hex(fcc.get_color()) ==␣
      ↪default_pos_color):
                         fcc.set_color(positive_color)
                     elif (matplotlib.colors.to_hex(fcc.get_color()) ==␣
      ↪default_neg_color):
                         fcc.set_color(negative_color)
         plt.show()
```

```
[ ]: def ShapValues(classifier, dataset):
         """
         classifier: model
         dataset: pandas dataframe
         slice_size: Row amount from dataset

         Returns shap_values
         """
         test_sample = dataset
         explainer = shap.Explainer(classifier)
         return explainer(test_sample)
```

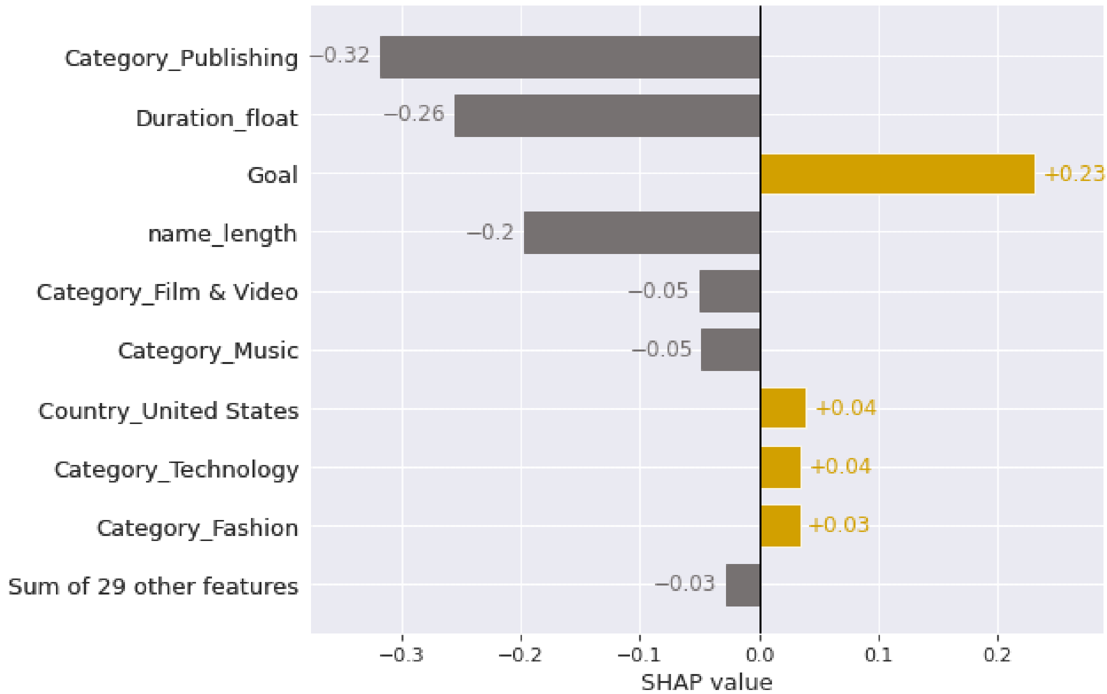
```
[ ]: ##############################################################
     MODEL = final_model #INSERT USED MODEL HERE!!!
     DATASET = X_test #INSERT USED DATASET HERE!!!
     ###############################################################
```

```
shap_values = ShapValues(MODEL, DATASET)
```

[ ]:
```
shap.plots.bar(shap_values, show=False)
change_colors()
```



[ ]:
```
shap.plots.bar(shap_values[0], show=False)
change_colors()
```



[ ]:
```
shap.plots.beeswarm(shap_values, show=False, color=plt.get_cmap("coolwarm"))
```