

Kickstarter Projects

Notebook 01 : Data Loading

Group's members:

- Crnigoj Gabriele 134176
- Ferraro Tommaso 132998
- Stinat Kevin 134905

The image shows the Kickstarter logo, which consists of the word "KICKSTARTER" in a bold, sans-serif font. The "KICK" part is in black, and the "STARTER" part is in a bright green color.

Tags:

business, finance, crowdfunding, accounting and auditing.

URL:

Website: [www.kickstarter.com \(https://www.kickstarter.com/?lang=it\)](https://www.kickstarter.com/?lang=it) \ Kaggle: [www.kaggle.com/kickstarter-projects \(https://www.kaggle.com/kemical/kickstarter-projects\)](https://www.kaggle.com/kemical/kickstarter-projects)

Context:

Kickstarter is an American public benefit corporation based in Brooklyn, New York, that maintains a global crowdfunding platform focused on creativity. The company's stated mission is to "help bring creative projects to life". As of December 2019, Kickstarter has received more than 4.6 billion dollar in pledges from 17.2 million backers to fund 445,000 projects, such as films, music, stage shows, comics, journalism, video games, technology, publishing, and food-related projects. People who back Kickstarter projects are offered tangible rewards or experiences in exchange for their pledges. This model traces its roots to subscription model of arts patronage, where artists would go directly to their audiences to fund their work.

Business Model:

Kickstarter is one of a number of crowdfunding platforms for gathering money from the public, which circumvents traditional avenues of investment. Project creators choose a deadline and a minimum funding goal. If the goal is not met by the deadline, no funds are collected (a kind of assurance contract). The Kickstarter platform is open to backers from anywhere in the world and to creators from many countries, including the US, UK, Canada, Australia, New Zealand, The Netherlands, Denmark, Ireland, Norway, Sweden, Spain, France, Germany, Austria, Italy, Belgium, Luxembourg, Switzerland and Mexico. Kickstarter applies a 5% fee on the total amount of the funds raised. Their payments processor applies an additional 3–5% fee. Unlike many forums for fundraising or investment, Kickstarter claims no ownership over the projects and the work they produce. The web pages of projects launched on the site are permanently archived and accessible to the public. After funding is completed, projects and uploaded media cannot be edited or removed from the site. There is no guarantee that people who post projects on Kickstarter will deliver on their projects, use the money to implement their projects, or that the completed projects will meet backers' expectations. Kickstarter advises backers to use their own judgment on supporting a project. They also warn project leaders that they could be liable for legal damages from backers for failure to deliver on promises. Projects might also fail even after a successful fundraising campaign when creators underestimate the total costs required or technical difficulties to be overcome. Asked what made Kickstarter different from other crowdfunding platforms, co-founder Perry Chen said: "I wonder if people really know what the definition of crowdfunding is. Or, if there's even an agreed upon definition of what it is. We haven't actively supported the use of the term because it can provoke more confusion. In our case, we focus on a middle ground between patronage and commerce. People are offering cool stuff and experiences in exchange for the support of their ideas. People are creating these mini-economies around their project ideas. So, you aren't coming to the site to get something for nothing; you are trying to create value for the people who support you. We focus on creative projects—music, film, technology, art, design, food and publishing—and within the category of crowdfunding of the arts, we are probably ten times the size of all of the others combined."

Categories:

Creators categorize their projects into one of 13 categories and 36 subcategories. They are: Art, Comics, Dance, Design, Fashion, Film and Video, Food, Games, Music, Photography, Publishing, Technology and Theater. Of these categories, Film & Video and Music are the largest categories and have raised the most money. These categories, along with Games, account for over half the money raised. Video games and tabletop games alone account for more than 2 dollars out of every 10 dollars spent on Kickstarter.

Projects:

On June 21, 2012, Kickstarter began publishing statistics on its projects. As of December 4, 2019, there were 469,286 launched projects (3,524 in progress), with a success rate of 37.45\% (success rate being how many were successfully funded by reaching their set goal). The total amount pledged was 4,690,286,673 dollars. On February 9, 2012, Kickstarter hit a number of milestones. A dock made for the iPhone designed by Casey Hopkins became the first Kickstarter project to exceed one million dollars in pledges. A few hours later, a new adventure game project started by computer game developers, Double Fine Productions, reached the same figure, having been launched less than 24 hours earlier, and finished with over 3 million dollars pledged. This was also the first time Kickstarter raised over a million dollars in pledges in a single day.

Our Goal:

From 2012 to 2013, Wharton professor Ethan Mollick and Jeanne Pi conducted research into what contributes to a project's success or failure on Kickstarter. Some key findings from the analysis were that increasing goal size is negatively associated with success, projects that are featured on the Kickstarter homepage have an 89\% chance of being successful, compared to 30\% without, and that for an average 10,000 dollars for project, a 30-day project has a 35\% chance of success, while a 60-day project has a 29\% chance of success, all other things being constant. The ten largest Kickstarter projects by funds raised are listed below. Among successful projects, most raise between 1,000 dollar and 9,999 dollars . These dollar amounts drop to less than half in the Design, Games, and Technology categories. However, the median amount raised for the latter two categories remains in the four-figure range. There is substantial variation in the success rate of projects falling under different categories. Over two thirds of completed dance projects have been successful. In contrast, fewer than 30\% of completed fashion projects have reached their goal. Most failing projects fail to achieve 20\% of their goals and this trend applies across all categories. Indeed, over 80\% of projects that pass the 20\% mark reach their goal.

What is the current situation in Kikcstarter? What are the most successful Projects? In which Categories? There are lots of questions that we can answer. Let's Analyze the DataFrame!

Data Import

First we load the DataFrame from the `.csv` file and we can inspect the data, in particular we have to check the nature (categorical, numerical, etc.) of each column and the range of values (using the `describe()` method) or the unique values (using the `unique()` or `nunique()` method on each categorical data column). Those analysis will be done in the Notebook 02 and 03 because the first place we need to managed the columns `deadline` and `launched` .

```
In [1]: import numpy as np
import pandas as pd
import joblib
import pickle
```

```
In [2]: df_ks = pd.read_csv('2018.csv')
df_ks.head()
```

Out[2]:

	ID	name	category	main_category	currency	deadline	goal	launc
0	1000002330	The Songs of Adelaide & Abullah	Poetry	Publishing	GBP	2015-10-09	1000.0	2015-12-11
1	1000003930	Greeting From Earth: ZGAC Arts Capsule For ET	Narrative Film	Film & Video	USD	2017-11-01	30000.0	2017-04-24
2	1000004038	Where is Hank?	Narrative Film	Film & Video	USD	2013-02-26	45000.0	2013-00-20
3	1000007540	ToshiCapital Rekordz Needs Help to Complete Album	Music	Music	USD	2012-04-16	5000.0	2012-03-24
4	1000011046	Community Film Project: The Art of Neighborhood...	Film & Video	Film & Video	USD	2015-08-29	19500.0	2015-08-31

```
In [3]: df_ks.tail()
```

```
Out[3]:
```

	ID	name	category	main_category	currency	deadline	goal
378656	999976400	ChknTruk Nationwide Charity Drive 2014 (Canceled)	Documentary	Film & Video	USD	2014-10-17	50000.0
378657	999977640	The Tribe	Narrative Film	Film & Video	USD	2011-07-19	1500.0
378658	999986353	Walls of Remedy- New lesbian Romantic Comedy f...	Narrative Film	Film & Video	USD	2010-08-16	15000.0
378659	999987933	BioDefense Education Kit	Technology	Technology	USD	2016-02-13	15000.0
378660	999988282	Nou Renmen Ayiti! We Love Haiti!	Performance Art	Art	USD	2011-08-16	2000.0

```
In [4]: df_ks.shape
```

```
Out[4]: (378661, 15)
```

Managing and Data Cleaning

We noticed that the columns `deadline` and `launched` contain aggregated data and for better reading we proceed to split them in order to manage in a better way the data. We use 'tqdm' function to analyze the progress status of the processing of the function. One of the most important problem of those columns were that their format was `object` and not a `datetime`.

We have also calculated the period in which the crowdfunding was opened, calculating it as the difference between the launch and the deadline dates. We have combined the dataframes created above with the main dataframe in order to have the complete data and be ready for an effective and efficient analysis

```
In [5]: import datetime
from tqdm import tqdm
```

```
In [6]: df_temp_deadline = pd.DataFrame(columns = ['Deadline_Year', 'MDeadline_Month', 'Deadline_Day'])
df_temp_launched = pd.DataFrame(columns = ['Launch_Year', 'Launch_Month', 'Launch_Day'])
df_temp_d = pd.DataFrame(columns = ['crowdfunding_period'])

for k in tqdm(range(len(df_ks))):
    #stripping the date
    date1 = datetime.datetime.strptime(str(df_ks['deadline'][k]), "%Y-%m-%d")
    df_temp_deadline.loc[k] = [date1.year, date1.month, date1.day]
    date2 = datetime.datetime.strptime(str(df_ks['launched'][k]), "%Y-%m-%d %H:%M:%S")
    df_temp_launched.loc[k] = [date2.year, date2.month, date2.day]

    #calculating the period in which the crowdfunding was open
    days_open = int((date1 - date2).days)
    df_temp_d.loc[k] = [days_open]
```

100%|██████████| 378661/378661 [6:56:27<00:00, 8.13it/s]

```
In [7]: df_temp_deadline.head()
```

Out[7]:

	Deadline_Year	MDeadline_Month	Deadline_Day
0	2015	10	9
1	2017	11	1
2	2013	2	26
3	2012	4	16
4	2015	8	29

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

Out[8]:

	Launch_Year	Launch_Month	Launch_Day
0	2015	8	11
1	2017	9	2
2	2013	1	12
3	2012	3	17
4	2015	7	4

```
In [9]: df_temp_d.head()
```

Out[9]:

crowdfunding_period	
0	58
1	59
2	44
3	29
4	55

```
In [10]: df_kickstarter = pd.concat([df_ks, df_temp_launched], axis = 1)
df_kickstarter = pd.concat([df_kickstarter, df_temp_deadline], axis
= 1)
df_kickstarter = pd.concat([df_kickstarter, df_temp_d], axis = 1)
df_kickstarter.head()
```

Out[10]:

	ID	name	category	main_category	currency	deadline	goal	launc
0	1000002330	The Songs of Adelaide & Abullah	Poetry	Publishing	GBP	2015- 10-09	1000.0	2015- 12:11
1	1000003930	Greeting From Earth: ZGAC Arts Capsule For ET	Narrative Film	Film & Video	USD	2017- 11-01	30000.0	2017- 04:41
2	1000004038	Where is Hank?	Narrative Film	Film & Video	USD	2013- 02-26	45000.0	2013- 00:21
3	1000007540	ToshiCapital Rekordz Needs Help to Complete Album	Music	Music	USD	2012- 04-16	5000.0	2012- 03:21
4	1000011046	Community Film Project: The Art of Neighborhood...	Film & Video	Film & Video	USD	2015- 08-29	19500.0	2015- 08:31

5 rows × 22 columns

Data Dump


```
In [11]: from tempfile import mkdtemp
import os

os.mkdir('Kickstarter_Dataframe')
savedir = 'Kickstarter_Dataframe'
```

```
In [12]: filename = os.path.join(savedir, 'Kikstarter_Backup_File')

with open(filename, 'wb') as r:
    joblib.dump(df_kickstarter , r)
```

Conclusion of Notebook 01

One of the main problems of our dataframe is its size and we have decided to use the function dumps in order to have the possibility to manage it better. We proceeded by creating the folder in which we want to save the compressed database file; we are forced to use this type of approach because the computation times of our database could not make us work efficiently. In order to save the file we have to use Joblib library to save it into a directory called 'Kickstarter_Dataframe' and we create a file called Kikstarter_Backup_File to save our dataframe. We use "wb" because it is a binary file. If we open the file, we can't read it because it has a different way of reading than a normal file in fact for this reason, we have to load it with joblib. Then we can open a new Notebook and test if the dump of the binary file went fine, and we can keep working on our project. We adopted this way of operating for all the next Notebooks because it's more efficient and save us lots of time and we can parallelize our work.

Kickstarter Projects

Notebook 02

Group's members:

- Crnigoj Gabriele 134176
- Ferraro Tommaso 132998
- Stinat Kevin 134905

Data Loading

We load and dump the file with `joblib` insted of "pickle" because we want to load and dump the data in the right way in order to avoid the possibilities of errors during the reading step.

Analyzing the dataframe , as we expected, the columns 'ID', 'goal', 'pledged', 'backers', 'usd_pledged_real' and 'usd pledged' have numerical nature. Using the `.describe()` function we get some more information about the data. However some of them are meaningful.

```
In [1]: import numpy as np
import pandas as pd
import joblib
import pickle
import os
from tqdm import tqdm
```

```
In [2]: os.path.exists('Kickstarter_Dataframe')
```

```
Out[2]: True
```

```
In [3]: from tempfile import mkdtemp
savedir = 'Kickstarter_Dataframe'
filename = os.path.join(savedir, 'Kikstarter_Backup_File')

with open(filename, 'rb') as r:
    df_ks = joblib.load(r)

df_ks.head(2)
```

Out[3]:

	ID	name	category	main_category	currency	deadline	goal	launched
0	1000002330	The Songs of Adelaide & Abullah	Poetry	Publishing	GBP	2015-10-09	1000.0	2015-08-11 12:12:28
1	1000003930	Greeting From Earth: ZGAC Arts Capsule For ET	Narrative Film	Film & Video	USD	2017-11-01	30000.0	2017-09-02 04:43:57

2 rows × 22 columns

```
In [4]: df_ks.shape
```

Out[4]: (378661, 22)

```
In [5]: df_ks.dtypes
```

```
Out[5]: ID                int64
        name              object
        category          object
        main_category      object
        currency           object
        deadline           object
        goal               float64
        launched           object
        pledged            float64
        state              object
        backers            int64
        country            object
        usd_pledged         float64
        usd_pledged_real   float64
        usd_goal_real       float64
        Launch_Year        object
        Launch_Month       object
        Launch_Day         object
        Deadline_Year      object
        MDeadline_Month    object
        Deadline_Day       object
        crowdfunding_period object
        dtype: object
```

Managing and Data Cleaning : NaN and Null Values

In order to be able to use and work with those data, we decided to use the `fillna` function that allow us to insert the **unknown** value instead of the **null** values.

To check the number of Null values we used the function `isna()` & `isnull()` and we put those values inside a dataframe called **df_ks_NaN** which is made by 3801 line which are exactly the sum of the null values found with the function above.

For each element of this dataframe that has some missing value we fill them with "Unknown". And to be sure of the right execution we check if the sum of the missing values running again the `.isnull().sum()` functions.

```
In [6]: df_ks.isnull().sum()
```

```
Out[6]: ID                                0
        name                              4
        category                          0
        main_category                      0
        currency                          0
        deadline                          0
        goal                              0
        launched                          0
        pledged                           0
        state                             0
        backers                           0
        country                           0
        usd pledged                       3797
        usd_pledged_real                  0
        usd_goal_real                     0
        Launch_Year                       0
        Launch_Month                      0
        Launch_Day                        0
        Deadline_Year                     0
        MDeadline_Month                   0
        Deadline_Day                      0
        crowdfounding_period              0
        dtype: int64
```

```
In [7]: df_ks.isna().sum()
```

```
Out[7]: ID                                0
        name                              4
        category                          0
        main_category                      0
        currency                          0
        deadline                          0
        goal                              0
        launched                          0
        pledged                           0
        state                             0
        backers                           0
        country                           0
        usd pledged                       3797
        usd_pledged_real                  0
        usd_goal_real                     0
        Launch_Year                       0
        Launch_Month                      0
        Launch_Day                        0
        Deadline_Year                     0
        MDeadline_Month                   0
        Deadline_Day                      0
        crowdfounding_period              0
        dtype: int64
```

```
In [8]: df_ks_NaN = df_ks[df_ks.isna().any(axis=1)]
df_ks_NaN.head(2)
```

Out[8]:

	ID	name	category	main_category	currency	deadline	goal	I
169	1000694855	STREETFIGHTERZ WHEELIE MURICA	Film & Video	Film & Video	USD	2014- 09-20	6500.0	
328	100149523	Duncan Woods - Chameleon EP	Music	Music	AUD	2015- 08-25	4500.0	

2 rows × 22 columns

```
In [9]: df_ks_NaN.shape
```

Out[9]: (3801, 22)

```
In [10]: df_kickstarter = df_ks.fillna('Unknow')
df_kickstarter.head(2)
```

Out[10]:

	ID	name	category	main_category	currency	deadline	goal	launched
0	1000002330	The Songs of Adelaide & Abullah	Poetry	Publishing	GBP	2015- 10-09	1000.0	2015-08- 11 12:12:28
1	1000003930	Greeting From Earth: ZGAC Arts Capsule For ET	Narrative Film	Film & Video	USD	2017- 11-01	30000.0	2017-09- 02 04:43:57

2 rows × 22 columns

```
In [11]: df_kickstarter.isnull().sum()
```

```
Out[11]: ID          0
         name        0
         category    0
         main_category 0
         currency     0
         deadline     0
         goal         0
         launched     0
         pledged      0
         state        0
         backers      0
         country      0
         usd_pledged  0
         usd_pledged_real 0
         usd_goal_real 0
         Launch_Year  0
         Launch_Month 0
         Launch_Day   0
         Deadline_Year 0
         MDeadline_Month 0
         Deadline_Day  0
         crowdfounding_period 0
         dtype: int64
```

Managing and Data Cleaning : Unique and Unusual Values

Using the `nunique()` function we have seen how there are 378661 different IDs but only 375765 Name and this means that for some projects the name has been repeated.

In fact to check this type of values we used the `counter` function and, through a for loop, we created a dictionary which contains each name that appeared more than once in our database, with the number of times that it is repeated.

We then created a dataframe called **repetition** containing all the lines that had the repetition of the name. Analyzing this dataframe we noticed that there are some projects that have been attempted several times on different dates or with different parameters.

```
In [12]: df_kickstarter.drop(columns = ['deadline', 'launched'], inplace = T
True)
df_kickstarter.nunique()
```

```
Out[12]: ID          378661
name          375765
category       159
main_category   15
currency        14
goal           8353
pledged        62130
state           6
backers         3963
country         23
usd_pledged     95456
usd_pledged_real 106065
usd_goal_real   50339
Launch_Year      11
Launch_Month     12
Launch_Day       31
Deadline_Year    10
MDeadline_Month  12
Deadline_Day     31
crowdfunding_period 99
dtype: int64
```


In [13]: `df_kickstarter.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 378661 entries, 0 to 378660
Data columns (total 20 columns):
ID                378661 non-null int64
name              378661 non-null object
category          378661 non-null object
main_category     378661 non-null object
currency          378661 non-null object
goal              378661 non-null float64
pledged           378661 non-null float64
state             378661 non-null object
backers           378661 non-null int64
country           378661 non-null object
usd pledged       378661 non-null object
usd_pledged_real  378661 non-null float64
usd_goal_real     378661 non-null float64
Launch_Year       378661 non-null int64
Launch_Month      378661 non-null int64
Launch_Day        378661 non-null int64
Deadline_Year     378661 non-null int64
MDeadline_Month   378661 non-null int64
Deadline_Day      378661 non-null int64
crowdfunding_period 378661 non-null int64
dtypes: float64(4), int64(9), object(7)
memory usage: 57.8+ MB
```

```
In [14]: from collections import defaultdict
from collections import Counter

counter = defaultdict(lambda: 0)

counter = Counter(list(df_kickstarter.name))
project_inspection = dict(filter((lambda x: x[1] > 1), counter.items()))
```

```
In [15]: temp = pd.DataFrame()
repetition = pd.DataFrame(columns = df_kickstarter.columns)

for l in tqdm(df_kickstarter.name):
    for k in project_inspection.keys():
        if l == k:
            temp = df_kickstarter.loc[df_kickstarter.name == l]
            repetition = repetition.append(temp)

repetition.head()
```

100% | ██████████ | 378661/378661 [03:57<00:00, 1596.24it/s]

Out[15]:

	ID	name	category	main_category	currency	goal	pledged	
219	1000940032	Space Trucker	Video Games	Games	USD	150.0	165.0	suc
136862	1694996467	Space Trucker	Video Games	Games	USD	300000.0	5753.0	
241	1001052357	Spilt Milk	Graphic Novels	Comics	AUD	18000.0	832.0	
334003	771376120	Spilt Milk	Television	Film & Video	AUD	30000.0	0.0	
250	1001110351	Kingdoms of Erden: King of the Mountain	Tabletop Games	Games	USD	1000.0	745.0	c

Data Analysis: Top Campaign

In the following statement we analysed the dataframe and we have calculated some interesting parameters in order to understand better the database.

We find out that the campaign with the highest amount pledged is the "Pebble Time - Awesome Smartwatch, No Compromises". In fact the goal of this campaign was 500000, but eventually 20338986 (Over 20 million!) USD was pledged.

78471 people backed the campaign and the average backer pledged 259.19 USD.

```

In [16]: #Analysis
df_ks['discrepancy'] = df_ks['goal'] - df_ks['usd_pledged_real']
df_ks['target_reached'] = df_ks['discrepancy'] <= 0

target_reached = df_ks.loc[lambda df_ks: df_ks['target_reached'] ==
True]
target_not_reached = df_ks.loc[lambda df_ks: df_ks['target_reached']
] == False]

target_reached_perc = round(len(target_reached) / len(df_ks) * 100,
2)
target_not_reached_perc = round(len(target_not_reached) / len(df_ks)
) * 100, 2)

df_ks['avg_pledged'] = df_ks['usd_pledged_real'] / df_ks['backers']

df_ks['avg_pledged'].fillna(0, inplace=True)
df_ks.replace(to_replace = [np.inf, -np.inf], value = 0, inplace =
True)

#Print
print('Out of {} Kickstarter campaigns:\n\n{} % reached their targe
t over time.'.format(len(df_ks), target_reached_perc))

print('For the {} campaigns that reached their target, there was on
average {} USD pledged more than the target.\n'.format(len(target_r
eached), round(target_reached['usd_pledged_real'].mean(), 2)))

print('{} % of the campaigns did not reach their target.\nFor the {
} campaigns that did not reach their target, there was on average {
} USD pledged less than the target.\n'.format(target_not_reached_pe
rc, len(target_not_reached), round(target_not_reached['usd_pledged_
real'].mean(), 2)))

print('Kickstarter backers pledge {} USD to a campaign on average'.
format(round(df_ks['avg_pledged'].mean(), 2)))

```

Out of 378661 Kickstarter campaigns:

34.93 % reached their target over time.

For the 132255 campaigns that reached their target, there was on a
verage 22997.36 USD pledged more than the target.

65.07 % of the campaigns did not reach their target.

For the 246406 campaigns that did not reach their target, there wa
s on average 1577.66 USD pledged less than the target.

Kickstarter backers pledge 64.58 USD to a campaign on average

```
In [17]: perc_successful = len(df_ks[df_ks['state'] == 'successful']) / len(df_ks) * 100
perc_failed = len(df_ks[df_ks['state'] == 'failed']) / len(df_ks) * 100
perc_canceled = len(df_ks[df_ks['state'] == 'canceled']) / len(df_ks) * 100
perc_other = 100 - (perc_successful + perc_failed + perc_canceled)

print('{} % of campaigns were successful\n\
{} % of campaigns failed\n\
{} % of campaigns were canceled\n\
{} % of campaigns belong to other categories'.format(round(perc_successful, 2), round(perc_failed, 2), round(perc_canceled, 2), round(perc_other, 2)))
```

35.38 % of campaigns were successful
 52.22 % of campaigns failed
 10.24 % of campaigns were canceled
 2.17 % of campaigns belong to other categories

```
In [18]: # Get a list of all categories in 'main_category'
cat_list = list(df_ks['main_category'].unique())

# Create a mapping of the average pledged per backer for every category
pledge_mapping = dict()
for category in cat_list:
    cat_df = df_ks[df_ks['main_category'] == category]
    pledge_mapping.update({category : cat_df['avg_pledged'].mean()})

# Visualize the results
pledge_df = pd.DataFrame(pledge_mapping, index = ['avg_pledged'])
pledge_df.sort_values('avg_pledged', axis = 1, inplace=True, ascending=False)
display(pledge_df.head())
```

	Technology	Design	Film & Video	Dance	Theater	Fashion	Food
avg_pledged	95.985817	94.35646	76.1804	70.725558	70.447174	63.856153	62.741457

```
In [19]: top_campaign = df_ks.loc[df_ks['usd_pledged_real'].idxmax()]
top_name = df_ks.loc[df_ks['usd_pledged_real'].idxmax()][ 'name' ]
top_goal = df_ks.loc[df_ks['usd_pledged_real'].idxmax()][ 'goal' ]
top_pledged = df_ks.loc[df_ks['usd_pledged_real'].idxmax()][ 'usd_pl
edged_real' ]
top_backers = df_ks.loc[df_ks['usd_pledged_real'].idxmax()][ 'backer
s' ]
top_avg_pledged = df_ks.loc[df_ks['usd_pledged_real'].idxmax()][ 'av
g_pledged' ]

print('The campaign with the highest amount pledged is the "{}".\n\
n\
The goal of this campaign was {}, but eventually {} (Over 20 millio
n!) USD was pledged.\n\n\
{} people backed the campaign and the average backer pledged {}
USD.'.format(top_name,int(top_goal),top_pledged,
top_backers, top_avg_pledged))
```

The campaign with the highest amount pledged is the "Pebble Time - Awesome Smartwatch, No Compromises".

The goal of this campaign was 500000, but eventually 20338986.27 (Over 20 million!) USD was pledged.


78471 people backed the campaign and the average backer pledged 25 9.1911186298123 USD.

```
In [20]: display(top_campaign)
```

ID	1799
979574	
name	Pebble Time - Awesome Smartwatch, No Compr
omises	
category	Product
Design	
main_category	
Design	
currency	
USD	
deadline	2015
-03-28	
goal	
500000	
launched	2015-02-24 15
:44:42	
pledged	2.03
39e+07	
state	succ
essful	
backers	
78471	
country	
US	
usd pledged	2.03
39e+07	
usd_pledged_real	2.03
39e+07	
usd_goal_real	
500000	
Launch_Year	
2015	
Launch_Month	
2	
Launch_Day	
24	
Deadline_Year	
2015	
MDeadline_Month	
3	
Deadline_Day	
28	
crowdfunding_period	
31	
discrepancy	-1.98
39e+07	
target_reached	
True	
avg_pledged	2
59.191	
Name: 157270, dtype: object	

KICKSTARTER Discover Start Search Projects Sign up Log in


Pebble Time - Awesome Smartwatch, No Compromises



Color e-paper smartwatch with up to 7 days of battery and a new timeline interface that highlights what's important in your day.

[Learn More](#)

Created by
Pebble
Technology



Summary: Thanks to this preliminary analysis we ended up to understand that:

- Only 34.93% of projects succeeded
- The category which gather the best mean of money pledged is Technology with 95.985817\$/backer
- 'Pebble Time - Awesome Smartwatch, No Compromises' is the best project with over 20 million USD pledged.

Gini Index

In the following statement we have calculated the Gini index of our categorical attributes. Our goal for this part is to find out if there are some particular pattern in those attributes that have a higher discriminative power. In fact the lower the Gini value, the higher the discriminative power.

But as we can see, the values of the gini index are almost all on 0.5 which means that the 'state' has no particular occurrences in every categorical attributes.

```
In [21]: categorical = list(filter(lambda x: x not in ['ID', 'name', 'goal',
'pledged', 'backers', 'usd pledged', 'launched', 'deadline',
'usd_pledged_real', 'usd_goal_real', 'Launch_Year',
'Launch_Month', 'Launch_Day', 'Deadline_Year',
'MDeadline_Month', 'MDeadline_Day', 'crowdfunding_period'], df_kickstarter.columns))
categorical
```

```
Out[21]: ['category', 'main_category', 'currency', 'state', 'country']
```

```
In [22]: def gini(df, attribute, c_attribute):
    _df = df[[attribute, c_attribute]].dropna()
    _df_agg = (_df.groupby([attribute, c_attribute]).size() / _df.groupby(attribute).size()) ** 2
    _df_agg = _df_agg.reset_index().groupby(attribute).apply(lambda s: 1.0 - sum(s[0]))
    _df_agg.loc['categories_list'] = (_df_agg * (_df.groupby('state').size() / _df.shape[0])).sum()
    return _df_agg
```

```
In [23]: for index, attribute in enumerate(set(categorical) - set(["state"])):
    print(gini(df_kickstarter, attribute, "state"))
    print('\n\n')
```

```
currency
AUD          0.581375
CAD          0.599655
CHF          0.561537
DKK          0.625147
EUR          0.567235
GBP          0.602576
HKD          0.675506
JPY          0.718750
MXN          0.593583
NOK          0.584491
NZD          0.583197
SEK          0.591352
SGD          0.632598
USD          0.588249
categories_list  0.000000
dtype: float64
```

```
category
3D Printing    0.628263
Academic       0.528303
Accessories    0.595749
Action         0.474828
```


Animals	0.509681
Animation	0.550386
Anthologies	0.480119
Apparel	0.506539
Apps	0.375185
Architecture	0.557015
Art	0.578207
Art Books	0.565413
Audio	0.537835
Bacon	0.533855
Blues	0.605369
Calendars	0.597371
Camera Equipment	0.627358
Candles	0.403617
Ceramics	0.584940
Children's Books	0.548663
Childrenswear	0.498120
Chiptune	0.372245
Civic Design	0.553178
Classical Music	0.506976
Comedy	0.589021
Comic Books	0.544855
Comics	0.569727
Community Gardens	0.470632
Conceptual Art	0.566832
Cookbooks	0.560748
...	
Small Batch	0.551069
Software	0.443156
Sound	0.639645
Space Exploration	0.598549
Spaces	0.569740
Stationery	0.597861
Tabletop Games	0.591251
Taxidermy	0.674556
Technology	0.537124
Television	0.468404
Textiles	0.521660
Theater	0.503587
Thrillers	0.565421
Translations	0.496988
Typography	0.554698
Vegan	0.554098
Video	0.370218
Video Art	0.521894
Video Games	0.575000
Wearables	0.647673
Weaving	0.557984
Web	0.408602
Webcomics	0.551807
Webseries	0.544267
Woodworking	0.500223
Workshops	0.545509

```
World Music      0.568382
Young Adult      0.512835
Zines            0.582845
categories_list  0.000000
Length: 160, dtype: float64
```

```
main_category
Art            0.574624
Comics         0.563151
Crafts         0.513920
Dance          0.505661
Design         0.614965
Fashion        0.539899
Film & Video    0.585785
Food           0.509330
Games          0.636333
Journalism     0.507205
Music          0.601310
Photography    0.546799
Publishing     0.559565
Technology     0.539030
Theater        0.522945
categories_list 0.000000
dtype: float64
```

```
country
AT      0.543011
AU      0.570346
BE      0.562822
CA      0.588850
CH      0.553518
DE      0.567991
DK      0.616509
ES      0.563357
FR      0.588952
GB      0.592368
HK      0.675506
IE      0.571710
IT      0.506336
JP      0.718750
LU      0.526015
MX      0.593583
N, 0"   0.118388
NL      0.543143
NO      0.570510
NZ      0.567285
SE      0.577745
SG      0.632598
US      0.581051
```

```
categories_list    0.000000  
dtype: float64
```

```
In [24]: df_kickstarter.dtypes
```

```
Out[24]: ID                int64  
name                object  
category            object  
main_category       object  
currency            object  
goal                float64  
pledged             float64  
state              object  
backers             int64  
country             object  
usd pledged         object  
usd_pledged_real    float64  
usd_goal_real       float64  
Launch_Year         int64  
Launch_Month        int64  
Launch_Day          int64  
Deadline_Year       int64  
MDeadline_Month     int64  
Deadline_Day        int64  
crowdfunding_period int64  
dtype: object
```

```
In [25]: df_kickstarter.astype({'ID': 'float64', 'backers': 'float64', 'Launch_Year': 'float64', 'Launch_Month': 'float64', 'Launch_Day': 'float64', 'Deadline_Year': 'float64', 'MDeadline_Month': 'float64', 'Deadline_Day': 'float64', 'crowdfunding_period': 'float64'}).dtypes
```

```
Out[25]: ID                float64
name                  object
category              object
main_category          object
currency              object
goal                  float64
pledged               float64
state                 object
backers               float64
country               object
usd pledged           object
usd_pledged_real      float64
usd_goal_real         float64
Launch_Year           float64
Launch_Month          float64
Launch_Day            float64
Deadline_Year         float64
MDeadline_Month       float64
Deadline_Day          float64
crowdfunding_period   float64
dtype: object
```

Data Visualisation

In this phase we plotted the most significant data so we can best represent our dataframe in the best way possible. Clearly the best way to analyze our data is to divide the projects according to their state: successful failed & others (canceled, live, undifined, suspended).

We also analyzed our data by dividing them by category to identify any recurrency in the outcome of the kickstarting project.

Futhermore we have plotted the distinctions for the pledged & backers sorted by categories.

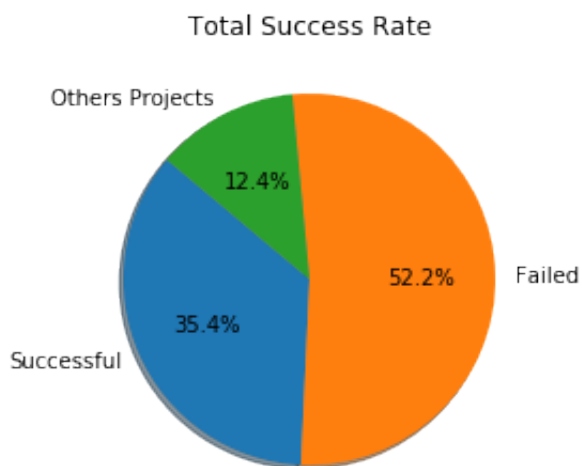
```
In [26]: import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

```
In [27]: success_projects = df_kickstarter[df_kickstarter['state'] == 'successful']['state'].count()
fail_projects = df_kickstarter[df_kickstarter['state'] == 'failed']['state'].count()
others_projects = (
    df_kickstarter[df_kickstarter['state'] == 'canceled']['state'].count() +
    df_kickstarter[df_kickstarter['state'] == 'live']['state'].count() +
    df_kickstarter[df_kickstarter['state'] == 'undefined']['state'].count() +
    df_kickstarter[df_kickstarter['state'] == 'suspended']['state'].count())
```

```
In [28]: total=success_projects+fail_projects+others_projects
suc=success_projects/total
fail=fail_projects/total
other=others_projects/total
```

```
In [29]: sizes = [suc, fail, other]
labels = 'Successful', 'Failed', 'Others Projects'
plt.pie(sizes, labels = labels, autopct = '%1.1f%%', shadow=True, startangle = 140)
plt.title("Total Success Rate")
```

Out[29]: Text(0.5, 1.0, 'Total Success Rate')



```
In [30]: failed = df_kickstarter.loc[df_kickstarter.state=='failed']
successful = df_kickstarter.loc[df_kickstarter.state=='successful']
canceled = df_kickstarter.loc[df_kickstarter.state=='canceled']
print('Mean duration of failed campaigns',failed['crowdfunding_per
iod'].mean())
print('Mean duration of successful campaigns',successful['crowdfoun
ding_period'].mean())
print('Mean duration of canceled campaigns',canceled['crowdfunding
_period'].mean())
```

Mean duration of failed campaigns 34.17335208047785
Mean duration of successful campaigns 31.156469288423065
Mean duration of canceled campaigns 37.26911472704299

```
In [31]: groupby_main_category = successful.groupby(['main_category']).mean(
)
groupby_main_category
```

Out[31]:

	ID	goal	pledged	backers	usd_pledged_real	u
main_category						
Art	1.079846e+09	4642.114081	7836.474882	90.358123	6971.894850	
Comics	1.059403e+09	5693.607838	11902.964118	231.404314	11385.660235	
Crafts	1.077424e+09	3539.475650	7022.015139	94.148463	5629.696889	
Dance	1.079357e+09	4941.846497	5550.954380	63.396065	5194.016121	
Design	1.079661e+09	17869.363175	69272.926059	614.783791	62858.524147	1
Fashion	1.064358e+09	10527.923923	23219.932871	217.001252	20286.555534	
Film & Video	1.072990e+09	11486.109404	14380.737041	152.560428	13951.872429	1
Food	1.077941e+09	12177.780807	18039.985200	180.557929	17349.390516	1
Games	1.075683e+09	15280.369228	56018.565317	814.384087	54228.755136	1
Journalism	1.085346e+09	9818.032045	12728.778745	152.306324	10344.950316	
Music	1.074948e+09	5928.511495	7543.683579	102.185354	7338.132395	
Photography	1.071967e+09	6644.391955	10296.176714	108.746445	10111.690306	
Publishing	1.079333e+09	6427.316163	10204.198186	158.823089	9436.910726	
Technology	1.059648e+09	28334.819035	98603.649686	717.678738	93085.762218	2
Theater	1.063860e+09	5329.732612	6113.351310	70.505663	5973.134904	

In [32]: `pip install plotly`

```
Requirement already satisfied: plotly in /Users/tommasoferraro/anaconda3/lib/python3.7/site-packages (4.5.2)
Requirement already satisfied: six in /Users/tommasoferraro/anaconda3/lib/python3.7/site-packages (from plotly) (1.12.0)
Requirement already satisfied: retrying>=1.3.3 in /Users/tommasoferraro/anaconda3/lib/python3.7/site-packages (from plotly) (1.3.3)
Note: you may need to restart the kernel to use updated packages.
```

In [33]: `%matplotlib inline
import plotly
from plotly import tools
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
init_notebook_mode(connected=True)
import warnings
warnings.filterwarnings('ignore')`

```
In [34]: trace1 = go.Bar(
            x=successful.main_category.value_counts().index,
            y=successful.main_category.value_counts().values,
            opacity=0.65
        )

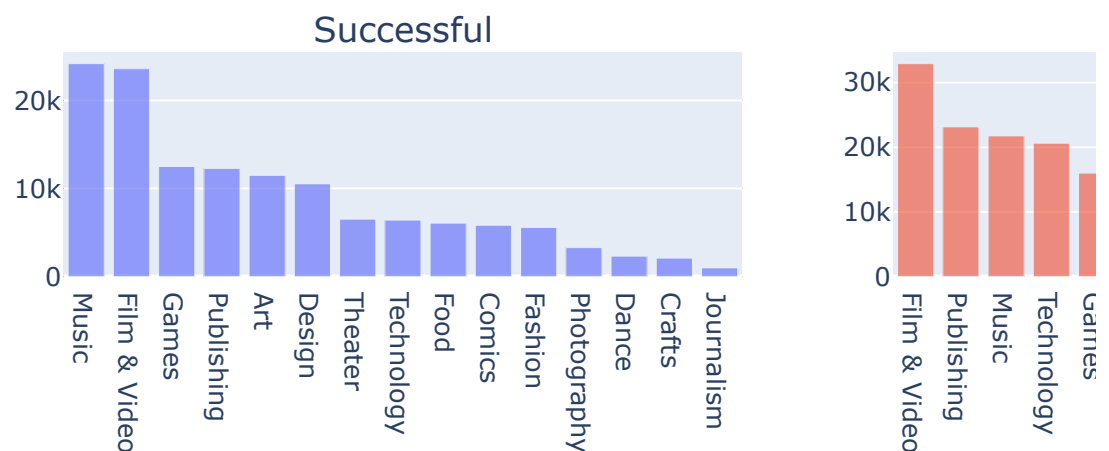
        trace2 = go.Bar(
            x=failed.main_category.value_counts().index,
            y=failed.main_category.value_counts().values,
            opacity=0.65
        )

        fig = tools.make_subplots(rows=1, cols=2, subplot_titles=('Successful', 'Failed'))
        fig.append_trace(trace1, 1, 1)
        fig.append_trace(trace2, 1, 2)

        fig['layout'].update(height=300, width=900, title='Distribution of
        main categories in Successful & Failed Campaigns')

        iplot(fig)
```

Distribution of main categories in Successful & Failed Campaigns




```
In [35]: trace1 = go.Bar(
            x=successful.category.value_counts()[ :20].index,
            y=successful.category.value_counts()[ :20].values,
            opacity=0.65
        )

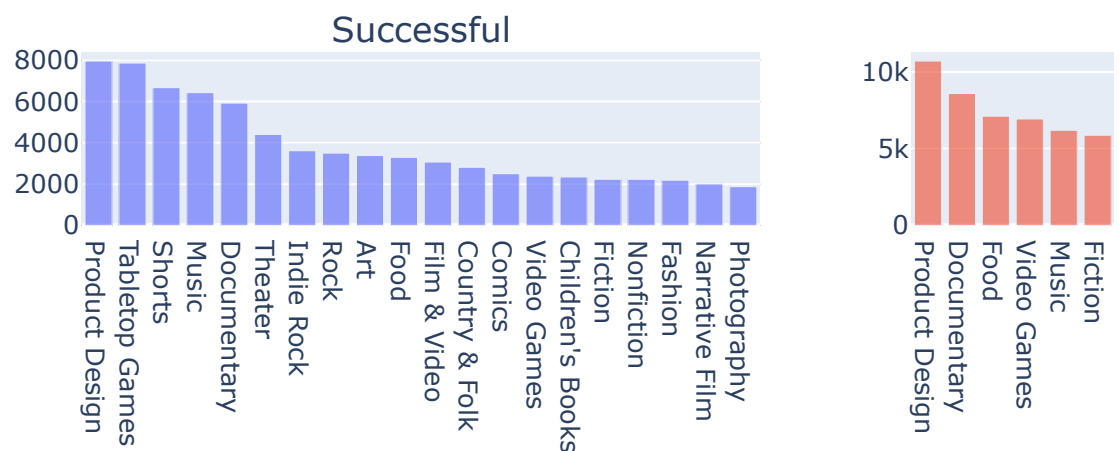
        trace2 = go.Bar(
            x=failed.category.value_counts()[ :20].index,
            y=failed.category.value_counts()[ :20].values,
            opacity=0.65
        )

        fig = tools.make_subplots(rows=1, cols=2, subplot_titles=('Successful', 'Failed'))
        fig.append_trace(trace1, 1, 1)
        fig.append_trace(trace2, 1, 2)

        fig['layout'].update(height=300, width=900, title='Distribution of
        top 20 categories in Successful & Failed Campaigns')

        iplot(fig)
```

Distribution of top 20 categories in Successful & Failed Campaigns



As we can see from the graphs above we notice how the "Film & video" category represents most of the projects for both successful and failed projects.

Instead we notice, how for example the "Gaming" categories has many more projects successfully than the failed ones. This is maybe due to the fact that as we can see from the graph below there are more Backers for the gaming projects.

However we can see how the duration of the projects is almost the same regardless of the category.

```
In [36]: tracel = go.Bar(
            x=groupby_main_category.backers.index,
            y=groupby_main_category.backers.values,
            opacity=0.65
        )

trace2 = go.Bar(
            x=groupby_main_category['usd_pledged_real'].index,
            y=groupby_main_category['usd_pledged_real'].values,
            opacity=0.65
        )

trace3 = go.Bar(
            x=groupby_main_category.usd_goal_real.index,
            y=groupby_main_category.usd_goal_real.values,
            opacity=0.65
        )

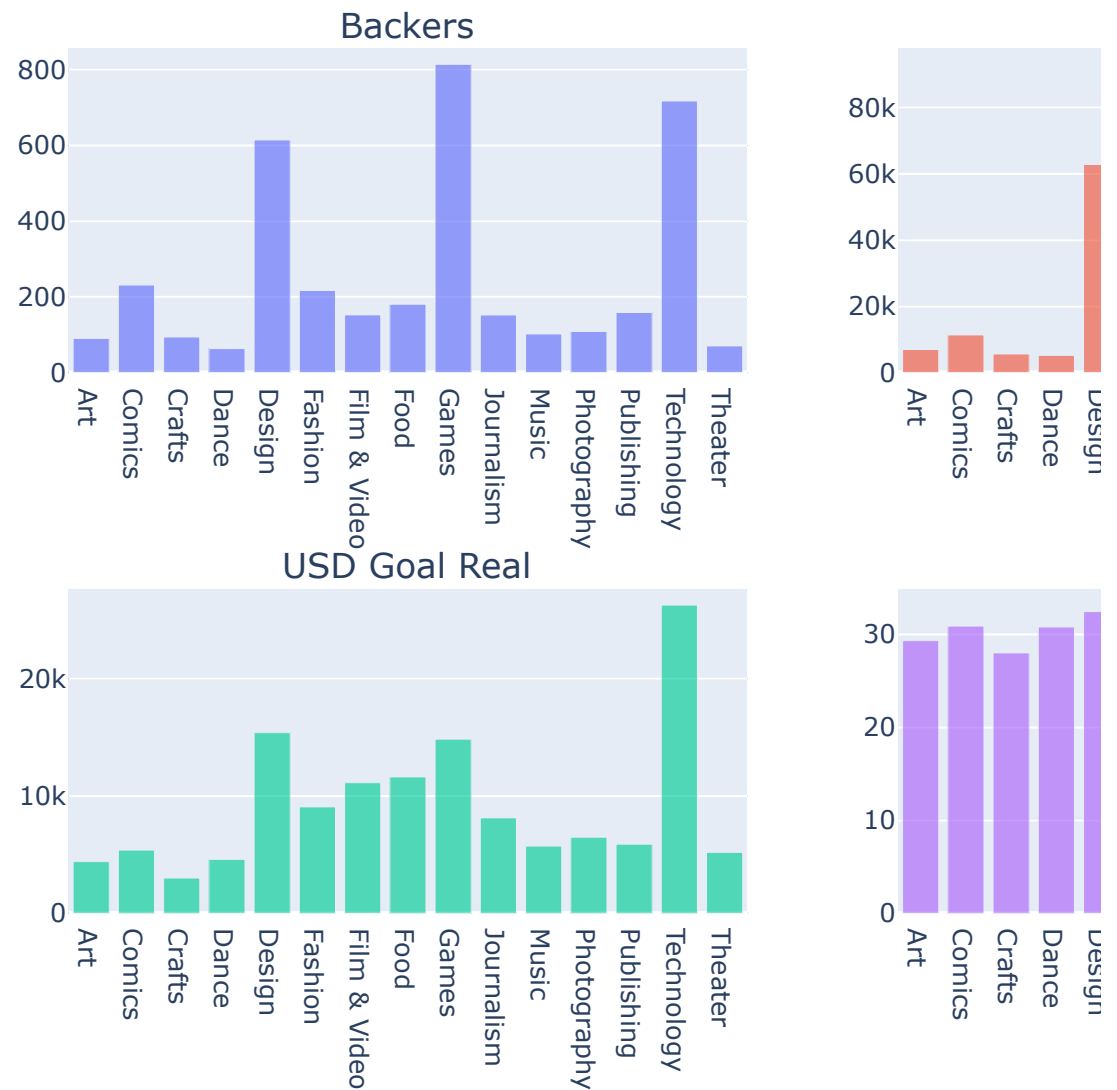
trace4 = go.Bar(
            x=groupby_main_category['crowdfunding_period'].index,
            y=groupby_main_category['crowdfunding_period'].values,
            opacity=0.65
        )

fig = tools.make_subplots(rows=2, cols=2, subplot_titles=('Backers'
, 'USD Pledged','USD Goal Real','Duration(days)'))
fig.append_trace(tracel, 1, 1)
fig.append_trace(trace2, 1, 2)
fig.append_trace(trace3, 2, 1)
fig.append_trace(trace4, 2, 2)

fig['layout'].update(height=600, width=900, title='Distribution acc
ording to Main Category of Successful Campaigns')

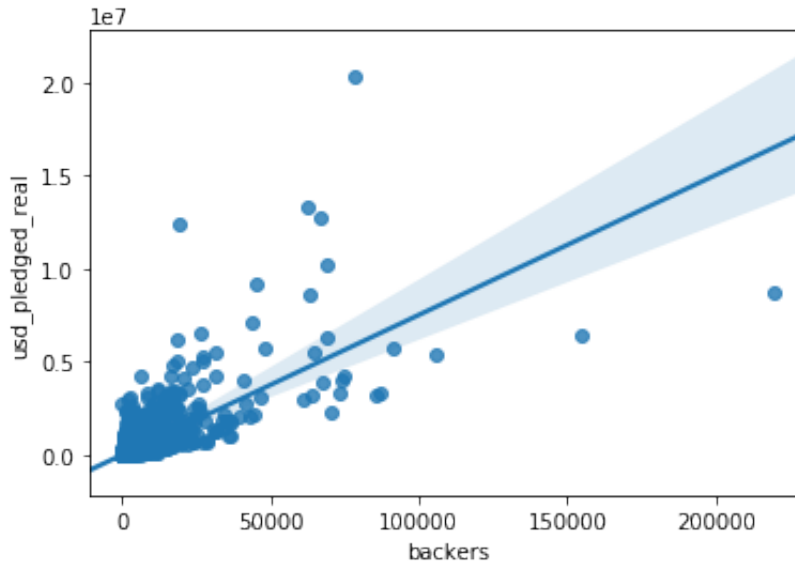
iplot(fig)
```

Distribution according to Main Category of Successful Campa



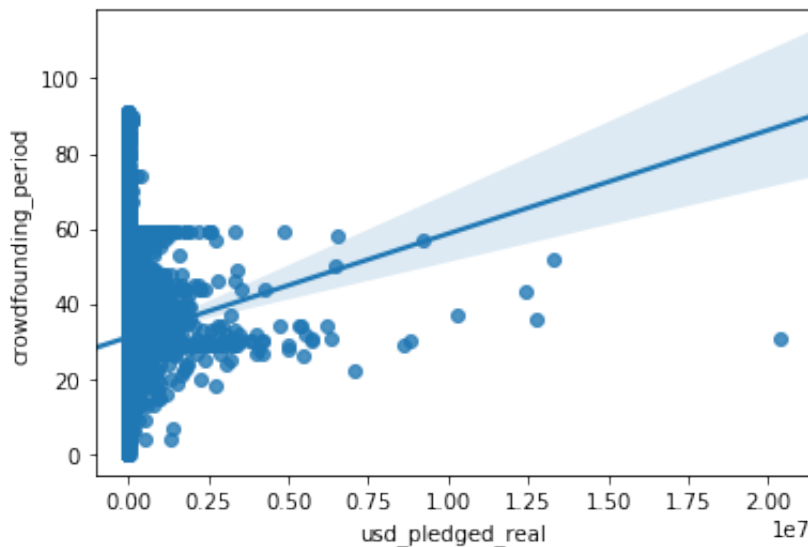
```
In [37]: sns.regplot(x='backers',y='usd_pledged_real', data=successful)
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x12e6a63c8>
```



```
In [38]: sns.regplot(x='usd_pledged_real',y='crowdfunding_period', data=successful)
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x134016320>
```



```
In [39]: ksdf_year = pd.DataFrame()
ksdf_year = {}

for year in range(2009, 2019):
    ksdf_year[year] = df_kickstarter[df_kickstarter['Launch_Year']
    == year]['Launch_Year'].count()
```

```
In [40]: ksdf_year = pd.Series(ksdf_year)
ksdf_year = pd.DataFrame(ksdf_year)
ksdf_year = ksdf_year.rename(columns = {0: "counts"})
```

```
In [41]: success_timely = []

for year in range(2009, 2019):
    success = len (df_kickstarter[(df_kickstarter['Launch_Year'] ==
year) & (df_kickstarter['state'] == 'successful')]['state'])
    overall = len (df_kickstarter[df_kickstarter['Launch_Year'] ==
year]['Launch_Year'])
    ratio = success/ overall
    success_timely.append(ratio)
    print ("Year = ",year, ratio * 100, '%')
```

```
Year = 2009 43.566591422121896 %
Year = 2010 43.66384637322939 %
Year = 2011 46.38868773106681 %
Year = 2012 43.46410785861776 %
Year = 2013 43.28777507747876 %
Year = 2014 31.156542918296555 %
Year = 2015 27.129366106080205 %
Year = 2016 32.81687185226637 %
Year = 2017 35.367816091954026 %
Year = 2018 0.0 %
```

```
In [42]: ksdf_year['success_ratio'] = success_timely
```

```
In [43]: backers_year = {}

for year in range(2009, 2019):
    backers_count = df_kickstarter[df_kickstarter['Launch_Year'] ==
year]['backers'].sum()
    (backers_count)
    backers_year[year] = backers_count
```

```
In [44]: ksdf_year['backers'] = pd.Series(backers_year)
```

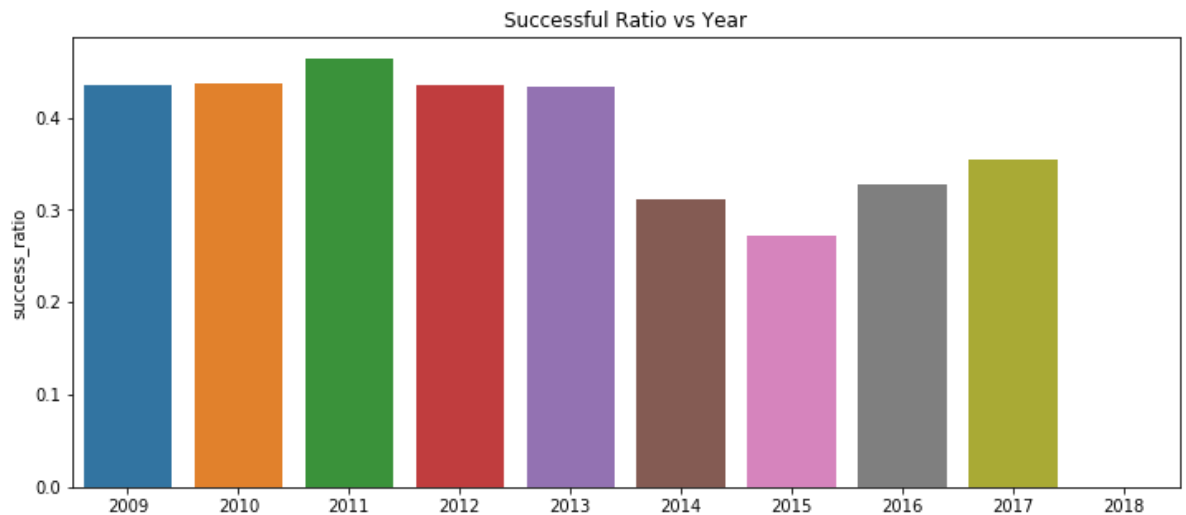
```
In [45]: ksdf_year
```

```
Out[45]:
```

	counts	success_ratio	backers
2009	1329	0.435666	43758
2010	10519	0.436638	406875
2011	26237	0.463887	1396473
2012	41165	0.434641	4343786
2013	44851	0.432878	6292568
2014	67745	0.311565	6194576
2015	77300	0.271294	7512058
2016	57184	0.328169	7148604
2017	52200	0.353678	6653360
2018	124	0.000000	1161

```
In [46]: plt.figure(figsize = (12, 5))  
plt.title("Successful Ratio vs Year")  
sns.barplot(x = ksdf_year.index, y = 'success_ratio', data = ksdf_year)
```

```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x12e4dbe80>
```



```
In [47]: pledged_sum = {}
cate_count = {}
back_cate = {}
success = {}

for category in list(set(df_kickstarter['main_category'])):
    amount = df_kickstarter[df_kickstarter['main_category'] == category]['usd_pledged_real'].sum()
    pledged_sum[category] = amount

# Create dataframe
cate_df = pd.Series(pledged_sum)
cate_df = pd.DataFrame(cate_df)
cate_df = cate_df.rename(columns = {0:"pledged_sum"})

for category in list(set(df_kickstarter['main_category'])):
    count = df_kickstarter[df_kickstarter['main_category'] == category]['main_category'].count()
    cate_count[category] = count

cate_df['count'] = pd.Series(cate_count)
cate_df['average_amount'] = cate_df['pledged_sum']/ cate_df['count']

for category in list(set(df_kickstarter['main_category'])):
    success_count = len(df_kickstarter[(df_kickstarter['main_category'] == category) &
    (df_kickstarter['state'] == "successful")])
    success[category] = success_count

cate_df["success_count"] = pd.Series(success)
cate_df["success_rate"] = cate_df['success_count']/ cate_df['count']

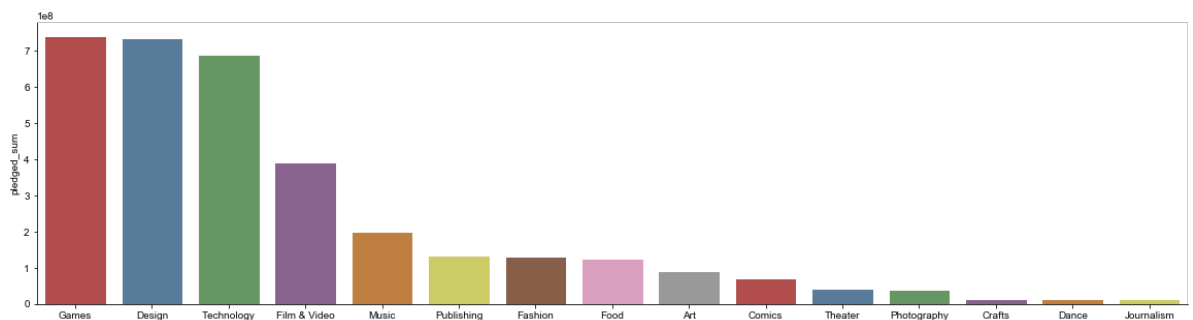
for category in set(df_kickstarter['main_category']):
    backers = df_kickstarter[df_kickstarter['main_category'] == category]['backers'].sum()
    back_cate[category] = backers

backers = pd.Series(back_cate)
cate_df['backers'] = backers
cate_df.head()
```

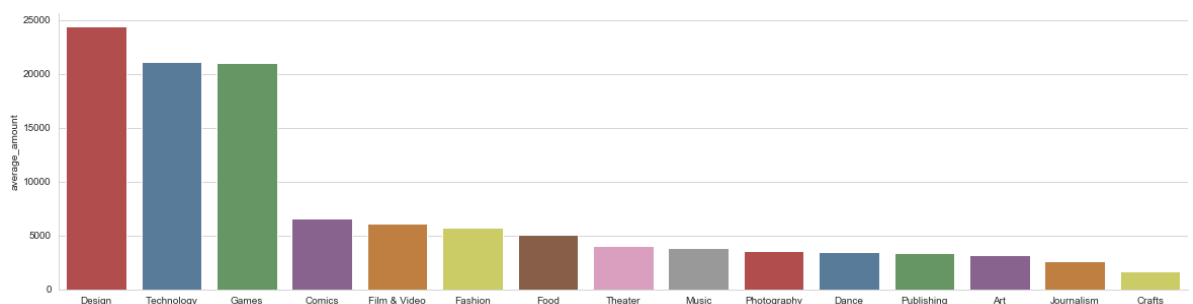
Out[47]:

	pledged_sum	count	average_amount	success_count	success_rate	backers
Technology	6.888725e+08	32569	21151.171165	6434	0.197550	5356513
Theater	4.371658e+07	10913	4005.918099	6534	0.598735	513536
Film & Video	3.915551e+08	63585	6157.978427	23623	0.371518	4197577
Publishing	1.335760e+08	39874	3349.951931	12300	0.308472	2231589
Games	7.413273e+08	35231	21041.903140	12518	0.355312	11336829

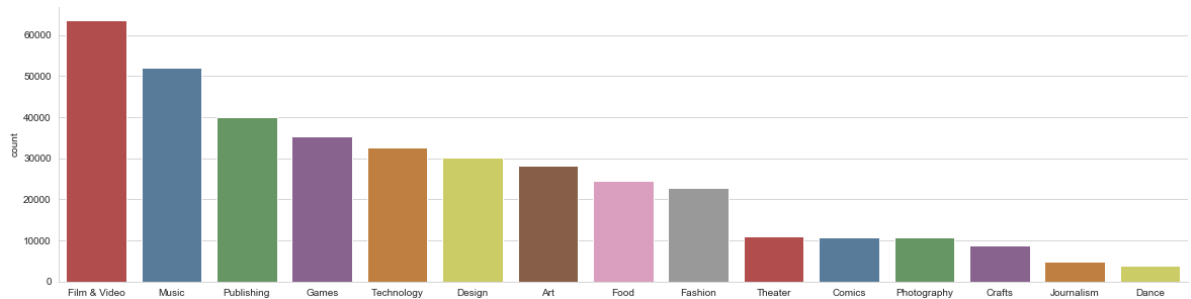
```
In [48]: #pledged_sum plot
cate_df = cate_df.sort_values('pledged_sum', ascending = False)
plt.subplots(figsize = (20,5))
sns.set_style("whitegrid")
sns.barplot(cate_df['pledged_sum'].index, y= cate_df['pledged_sum']
,
            palette="Set1",saturation = 0.5)
sns.despine(right = True, top = True)
```



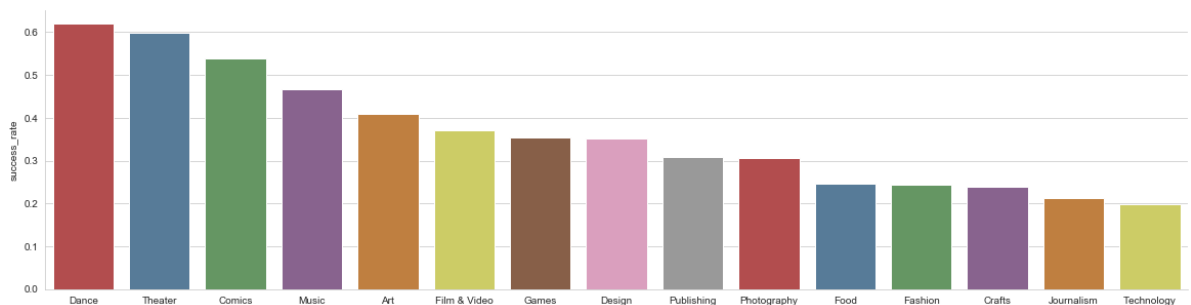
```
In [49]: #average amount plot
cate_df = cate_df.sort_values('average_amount', ascending = False)
plt.subplots(figsize = (20,5))
sns.set_style("whitegrid")
sns.barplot(cate_df['average_amount'].index, y= cate_df['average_
amount'] ,
            palette="Set1",saturation = 0.5)
sns.despine(right = True, top = True)
```



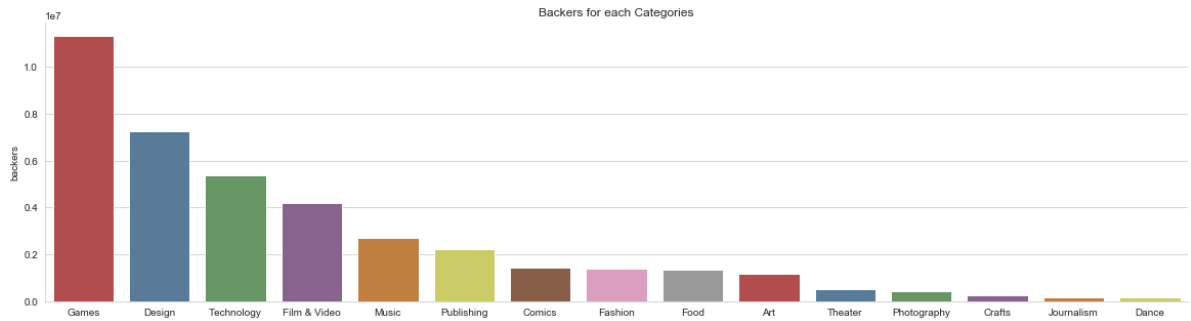

```
In [50]: #count plot
cate_df = cate_df.sort_values('count', ascending = False)
plt.subplots(figsize = (20,5))
sns.set_style("whitegrid")
sns.barplot(cate_df['count'].index, y= cate_df['count'] ,
            palette="Set1",saturation = 0.5)
sns.despine(right = True, top = True)
```



```
In [51]: # success rate plot
cate_df = cate_df.sort_values('success_rate', ascending = False)
plt.subplots(figsize = (20,5))
sns.set_style("whitegrid")
sns.barplot(cate_df['success_rate'].index, y= cate_df['success_rate'] ,
            palette="Set1",saturation = 0.5)
sns.despine(right = True, top = True)
```



```
In [52]: # backers plot
cate_df = cate_df.sort_values('backers', ascending = False)
plt.subplots(figsize = (20,5))
plt.title('Backers for each Categories')
sns.set_style("whitegrid")
sns.barplot(cate_df['backers'].index, y= cate_df['backers'] ,
            palette="Set1",saturation = 0.5)
sns.despine(right = True, top = True)
```



Data Inspection

```
In [53]: df_kickstarter.corr()
```

Out[53]:

	ID	goal	pledged	backers	usd_pledged_real	usd_goal_real
ID	1.000000	0.001679	0.000466	0.000654	-0.000025	0.001854
goal	0.001679	1.000000	0.007358	0.004012	0.005104	0.942692
pledged	0.000466	0.007358	1.000000	0.717079	0.952843	0.029424
backers	0.000654	0.004012	0.717079	1.000000	0.752539	0.016333
usd_pledged_real	-0.000025	0.005104	0.952843	0.752539	1.000000	0.005596
usd_goal_real	0.001854	0.942692	0.005024	0.004517	0.005596	1.000000
Launch_Year	-0.000574	0.015712	0.029424	0.016333	0.021875	0.000092
Launch_Month	0.000729	0.001183	0.002444	-0.002666	-0.000229	0.000092
Launch_Day	0.002207	-0.002581	0.000092	-0.001393	0.000161	-0.000092
Deadline_Year	-0.000717	0.016343	0.029135	0.015381	0.021364	0.000092
MDeadline_Month	0.001157	-0.000005	0.005790	0.003861	0.004561	0.000092
Deadline_Day	-0.000197	-0.001201	0.001620	-0.000536	0.000797	-0.000092
crowdfunding_period	-0.001131	0.004321	0.000847	-0.000792	0.000946	0.000092

```
In [54]: df_kickstarter.dtypes
```

```
Out[54]: ID                int64
name                object
category            object
main_category       object
currency            object
goal                float64
pledged             float64
state              object
backers             int64
country            object
usd pledged         object
usd_pledged_real    float64
usd_goal_real       float64
Launch_Year         int64
Launch_Month        int64
Launch_Day          int64
Deadline_Year       int64
MDeadline_Month     int64
Deadline_Day        int64
crowdfunding_period int64
dtype: object
```

```
In [55]: df_kickstarter = df_kickstarter.drop(columns = ['usd pledged'])
```

```
In [56]: categorical_print = list(filter(lambda x: x not in ['ID', 'name', 'goal', 'pledged', 'backers', 'usd_pledged_real', 'usd_goal_real', 'Launch_Year', 'Launch_Month', 'Launch_Day', 'Deadline_Year', 'MDeadline_Month', 'Deadline_Day', 'crowdfunding_period'], df_kickstarter.columns))
categorical_print
```

```
Out[56]: ['category', 'main_category', 'currency', 'state', 'country']
```

```
In [57]: for i in categorical_print:
          print(i, ': ', ', '.join(df_kickstarter[i].unique()), '\n')
```

category : Poetry, Narrative Film, Music, Film & Video, Restaurants, Food, Drinks, Product Design, Documentary, Nonfiction, Indie Rock, Crafts, Games, Tabletop Games, Design, Comic Books, Art Books, Fashion, Childrenswear, Theater, Comics, DIY, Webseries, Animation, Food Trucks, Public Art, Illustration, Photography, Pop, People, Art, Family, Fiction, Accessories, Rock, Hardware, Software, Weaving, Gadgets, Web, Jazz, Ready-to-wear, Festivals, Video Games, Anthologies, Publishing, Shorts, Electronic Music, Radio & Podcasts, Apps, Cookbooks, Apparel, Metal, Comedy, Hip-Hop, Periodicals, Dance, Technology, Painting, World Music, Photobooks, Drama, Architecture, Young Adult, Latin, Mobile Games, Flight, Fine Art, Action, Playing Cards, Makerspaces, Punk, Thrillers, Children's Books, Audio, Performance Art, Ceramics, Vegan, Graphic Novels, Fabrication Tools, Performances, Sculpture, Sound, Stationery, Print, Farmer's Markets, Events, Classical Music, Graphic Design, Spaces, Country & Folk, Wearables, Mixed Media, Journalism, Movie Theaters, Animals, Digital Art, Horror, Knitting, Small Batch, Installations, Community Gardens, DIY Electronics, Embroidery, Camera Equipment, Jewelry, Farms, Conceptual Art, Fantasy, Webcomics, Experimental, Science Fiction, Puzzles, R&B, Music Videos, Calendars, Video, Plays, Blues, Bacon, Faith, Live Games, Woodworking, Places, Footwear, 3D Printing, Academic, Zines, Musical, Workshops, Photo, Immersive, Letterpress, Gaming Hardware, Candles, Television, Space Exploration, Couture, Nature, Robots, Typography, Crochet, Translations, Textiles, Pottery, Interactive Design, Video Art, Quilts, Glass, Pet Fashion, Printing, Romance, Civic Design, Kids, Literary Journals, Taxidermy, Literary Spaces, Chiptune, Residencies

main_category : Publishing, Film & Video, Music, Food, Design, Crafts, Games, Comics, Fashion, Theater, Art, Photography, Technology, Dance, Journalism

currency : GBP, USD, CAD, AUD, NOK, EUR, MXN, SEK, NZD, CHF, DKK, HKD, SGD, JPY

state : failed, canceled, successful, live, undefined, suspended

country : GB, US, CA, AU, NO, IT, DE, IE, MX, ES, N,0", SE, FR, NL, NZ, CH, AT, DK, BE, HK, LU, SG, JP

```
In [58]: df_ks.groupby('state')['ID'].count()
```

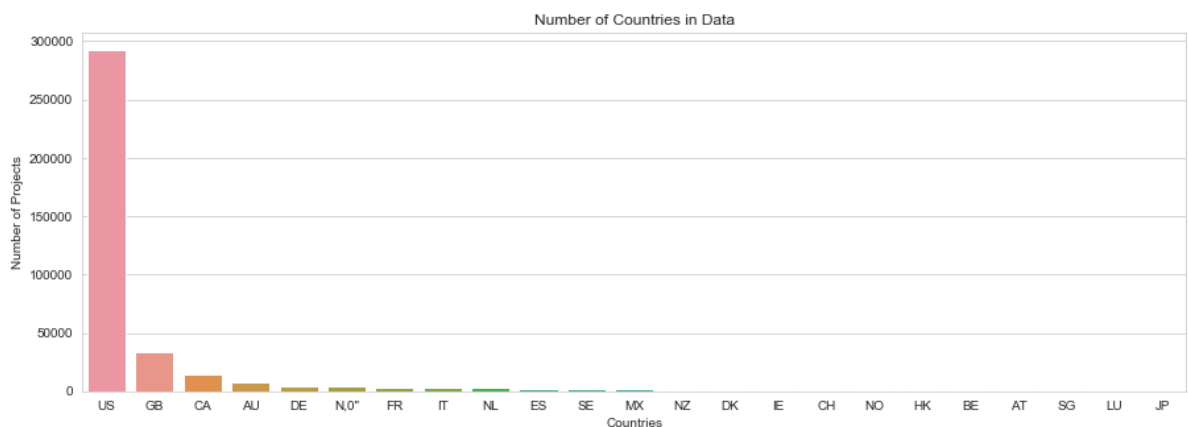
```
Out[58]: state
canceled      38779
failed        197719
live           2799
successful    133956
suspended      1846
undefined      3562
Name: ID, dtype: int64
```

```
In [59]: df_ks.groupby('country')['ID'].count()
```

```
Out[59]: country
AT          597
AU         7839
BE          617
CA        14756
CH          761
DE         4171
DK         1113
ES         2276
FR         2939
GB        33672
HK          618
IE          811
IT         2878
JP           40
LU           62
MX         1752
N,0"        3797
NL         2868
NO          708
NZ         1447
SE         1757
SG          555
US       292627
Name: ID, dtype: int64
```

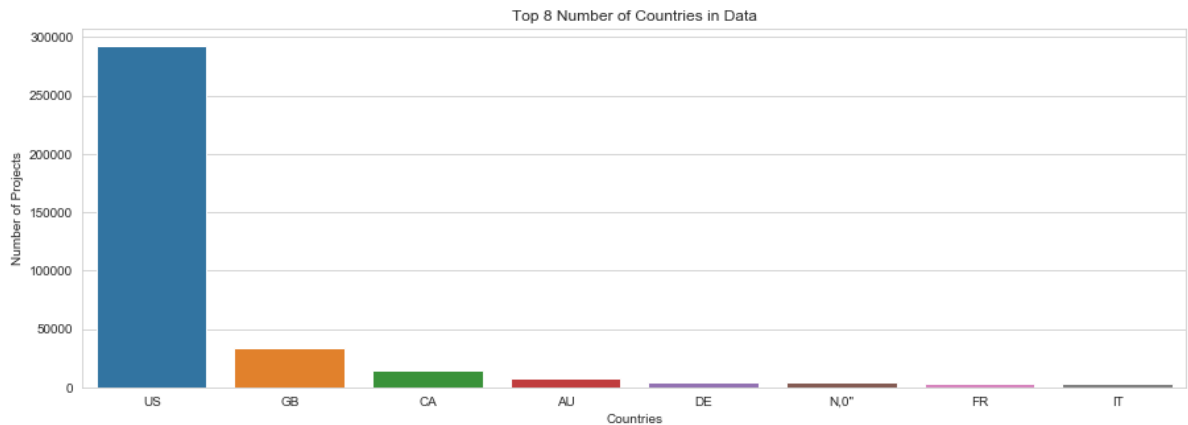
```
In [60]: categories_country = df_kickstarter.country.value_counts()
plt.figure(figsize = (15, 5))
sns.barplot(x = categories_country.index, y = categories_country.values)
plt.ylabel('Number of Projects')
plt.xlabel('Countries')
plt.title('Number of Countries in Data')
```

```
Out[60]: Text(0.5, 1.0, 'Number of Countries in Data')
```



```
In [61]: plt.figure(figsize = (15, 5))
sns.barplot(x = categories_country[:8].index, y = categories_countr
y[:8].values)
plt.ylabel('Number of Projects')
plt.xlabel('Countries')
plt.title('Top 8 Number of Countries in Data')
```

```
Out[61]: Text(0.5, 1.0, 'Top 8 Number of Countries in Data')
```



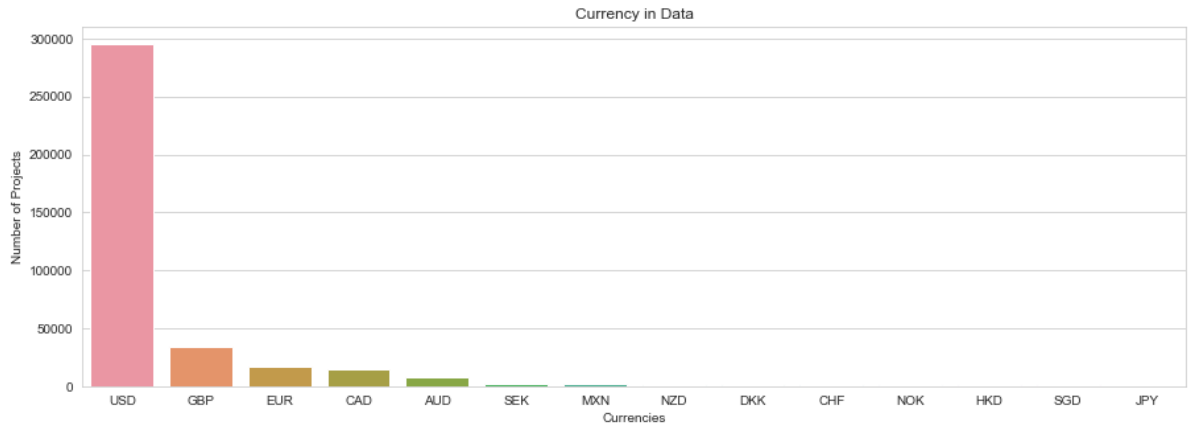
As we can see from the plot of the number of countries above, most of the projects are from the US and only a small percentage of them are from the other countries. And because of this, as it is logical to think the major currency used in the projects is the dollar, in fact this is what represents the following plot.

```
In [62]: df_ks.groupby('currency')['ID'].count()
```

```
Out[62]: currency
AUD      7950
CAD     14962
CHF       768
DKK     1129
EUR     17405
GBP     34132
HKD       618
JPY        40
MXN     1752
NOK       722
NZD     1475
SEK     1788
SGD       555
USD    295365
Name: ID, dtype: int64
```

```
In [63]: categories_currency = df_kickstarter.currency.value_counts()
plt.figure(figsize = (15, 5))
sns.barplot(x = categories_currency.index, y = categories_currency.
values)
plt.ylabel('Number of Projects')
plt.xlabel('Currencies')
plt.title('Currency in Data')
```

Out[63]: Text(0.5, 1.0, 'Currency in Data')

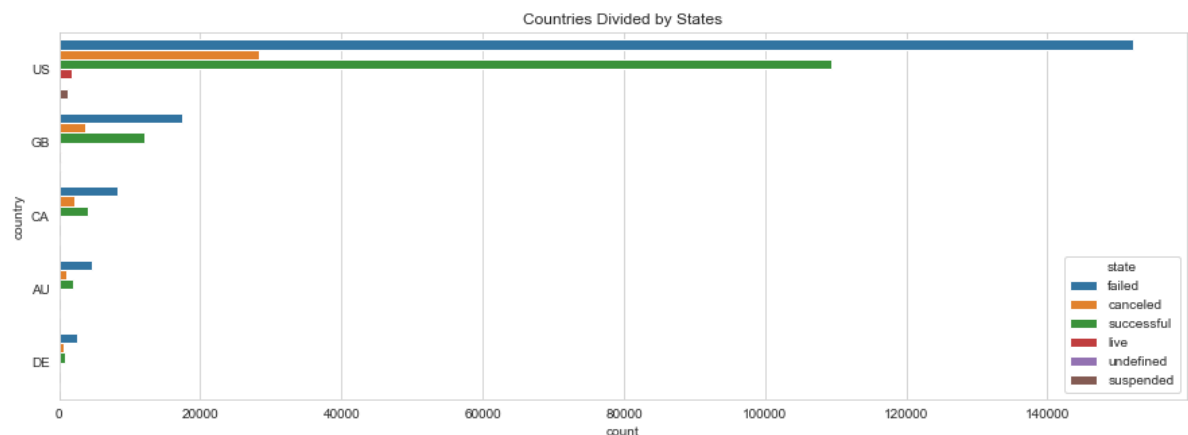


State Plotting in the Data

In the following paragraph we wanted to outline how the various projects are divided by each state.

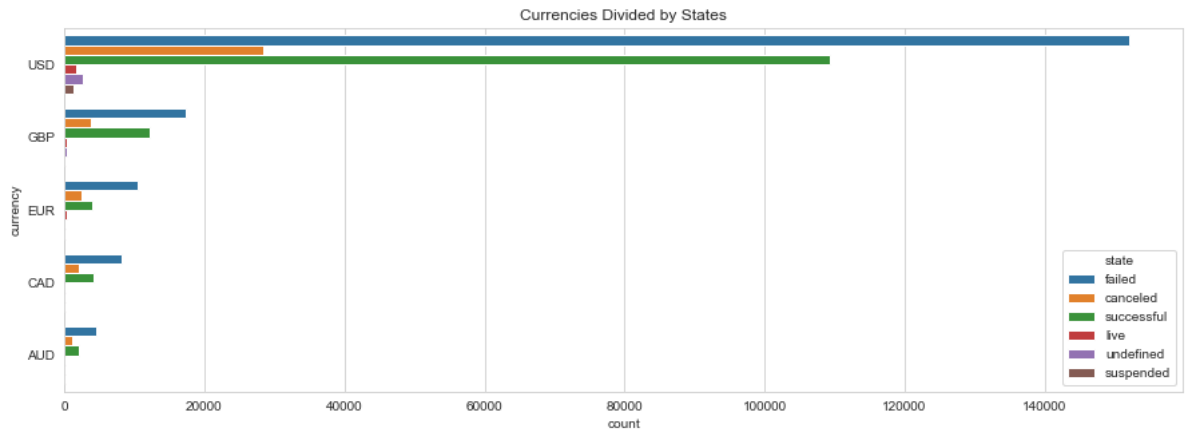
```
In [68]: plt.figure(figsize = (15, 5))
plt.title("Countries Divided by States")
sns.countplot(y = 'country', hue = 'state', data = df_kickstarter ,
order = ['US', 'GB', 'CA', 'AU', 'DE'])
```

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x12e959358>



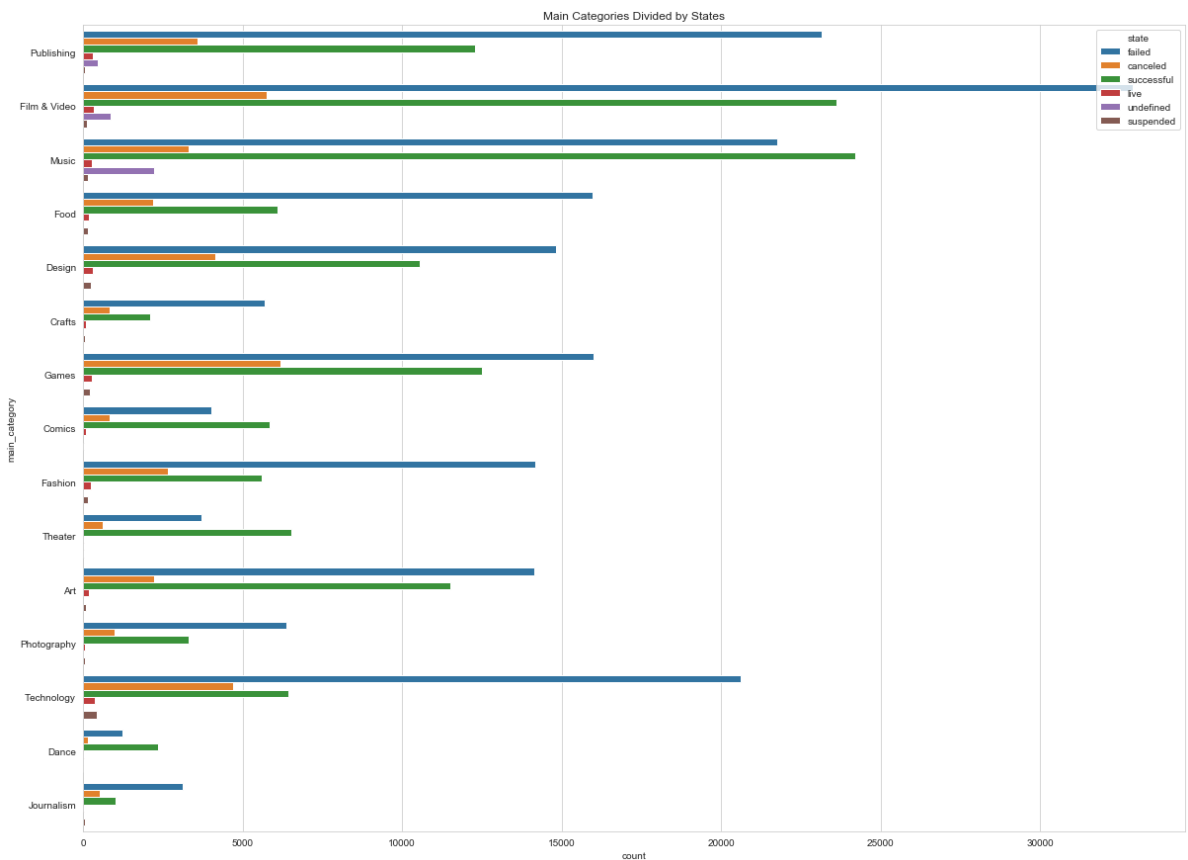
```
In [69]: plt.figure(figsize = (15, 5))
plt.title("Currencies Divided by States")
sns.countplot(y = 'currency' , hue = 'state', data = df_kickstarter
, order = ['USD', 'GBP', 'EUR', 'CAD', 'AUD'])
```

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x12eb97240>



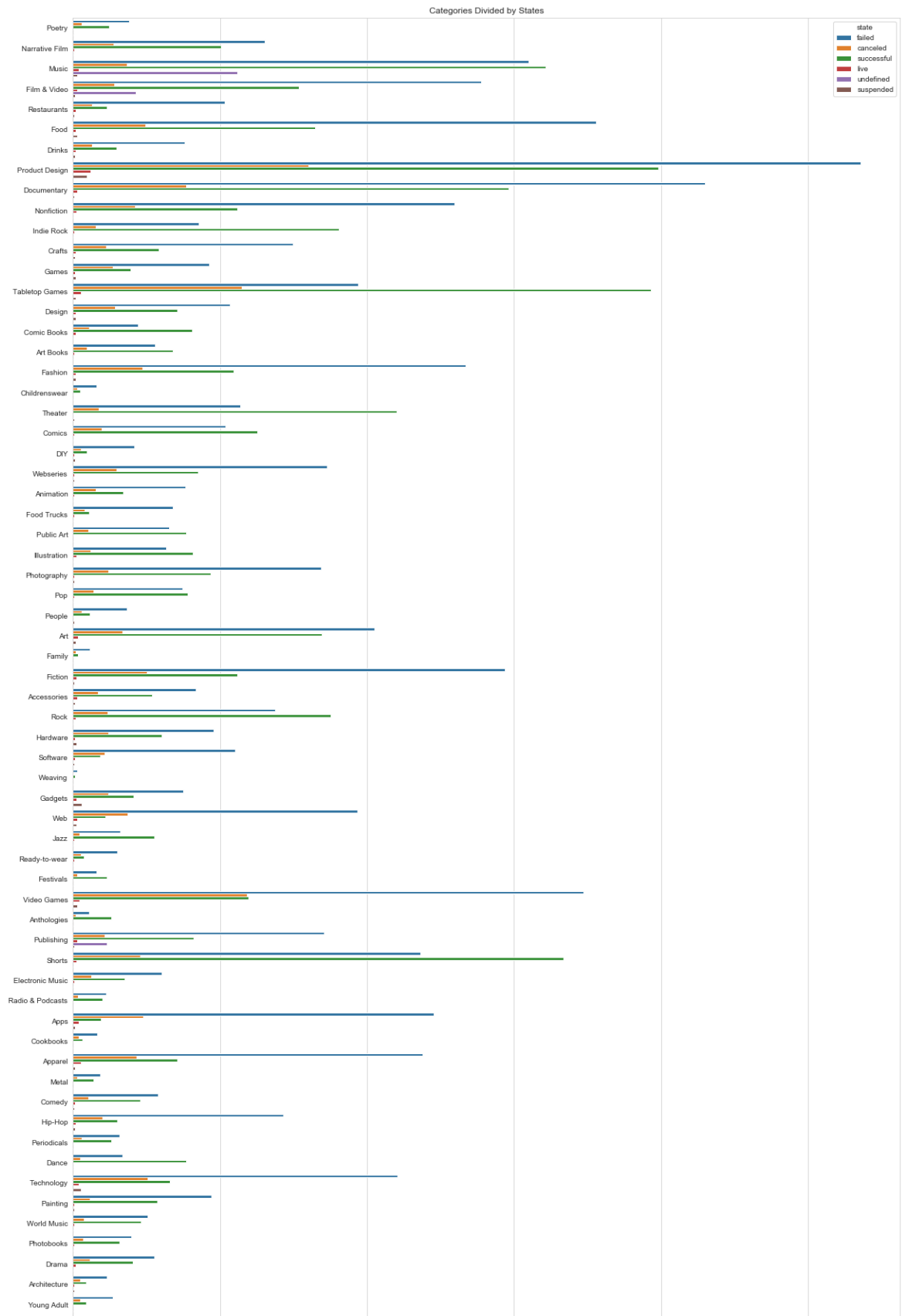
```
In [70]: plt.figure(figsize = (20, 15))
plt.title("Main Categories Divided by States")
sns.countplot(y = 'main_category', hue = 'state', data = df_kickstarter)
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x134050f98>

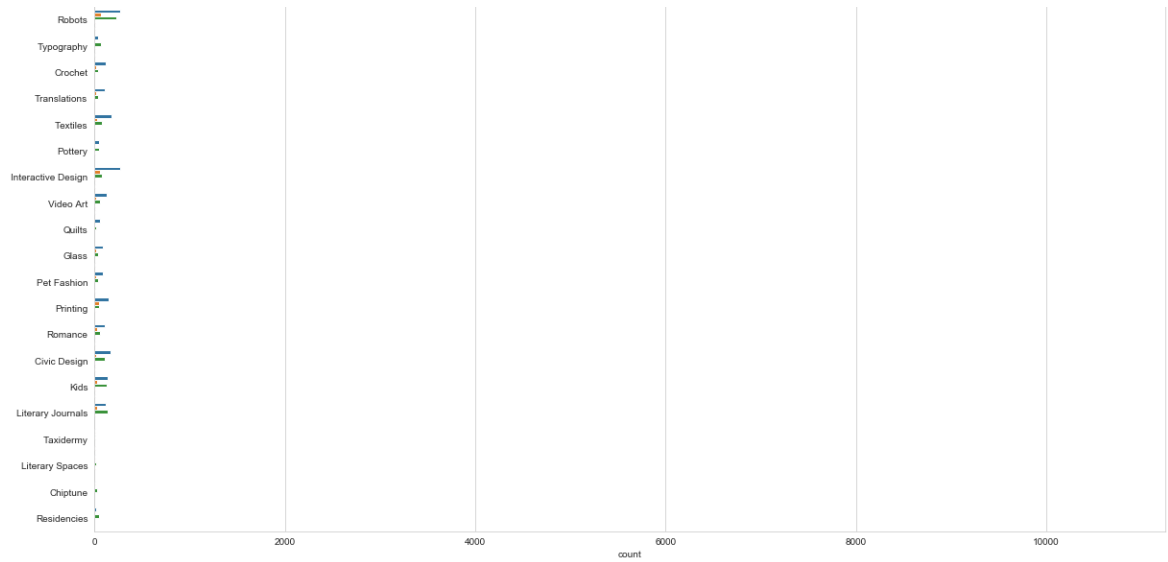



```
In [71]: plt.figure(figsize = (20, 80))  
plt.title("Categories Divided by States")  
sns.countplot(y = 'category', hue = 'state', data = df_kickstarter)
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x12ecd6630>







Derivate Plotting from Data

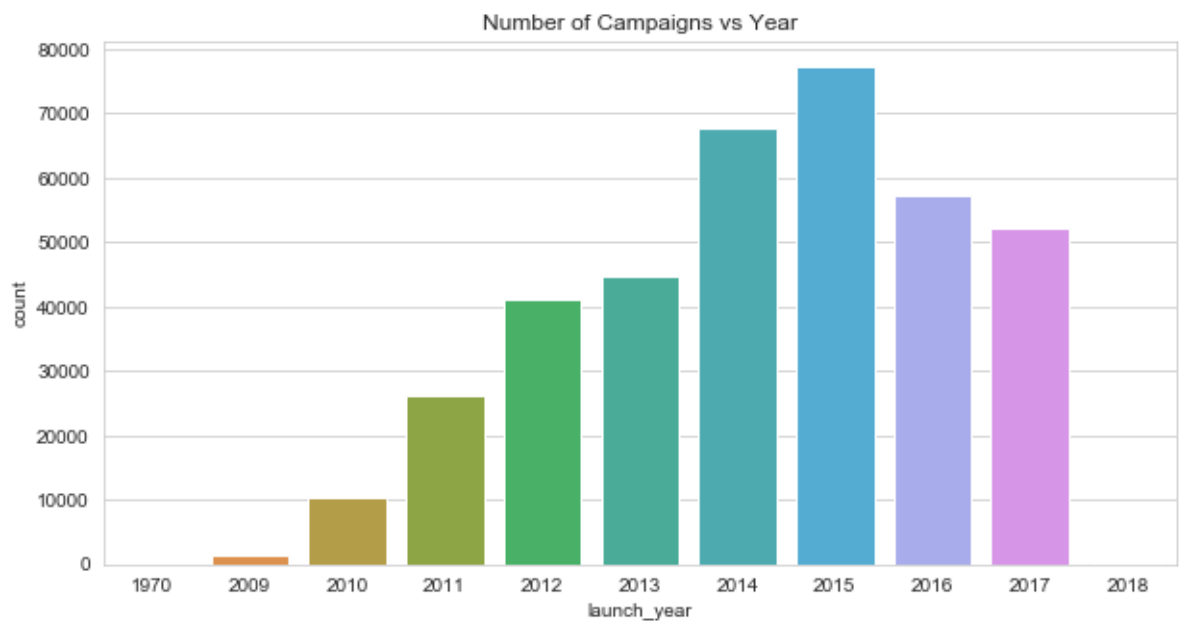
```
In [72]: df_years = pd.DataFrame()  
df_years["launch_year"] = df_kickstarter['Launch_Year']  
df_years.head()
```

Out[72]:

	launch_year
0	2015
1	2017
2	2013
3	2012
4	2015

```
In [73]: plt.figure(figsize = (10, 5))  
plt.title("Number of Campaigns vs Year")  
sns.countplot(df_years.launch_year)
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x1279b4160>



```
In [74]: backers = []

launchyear = df_kickstarter.Launch_Year.value_counts().index

for i in launchyear:
    new = df_kickstarter[df_kickstarter.Launch_Year == i]

    backers.append((i, new.backers.sum()))

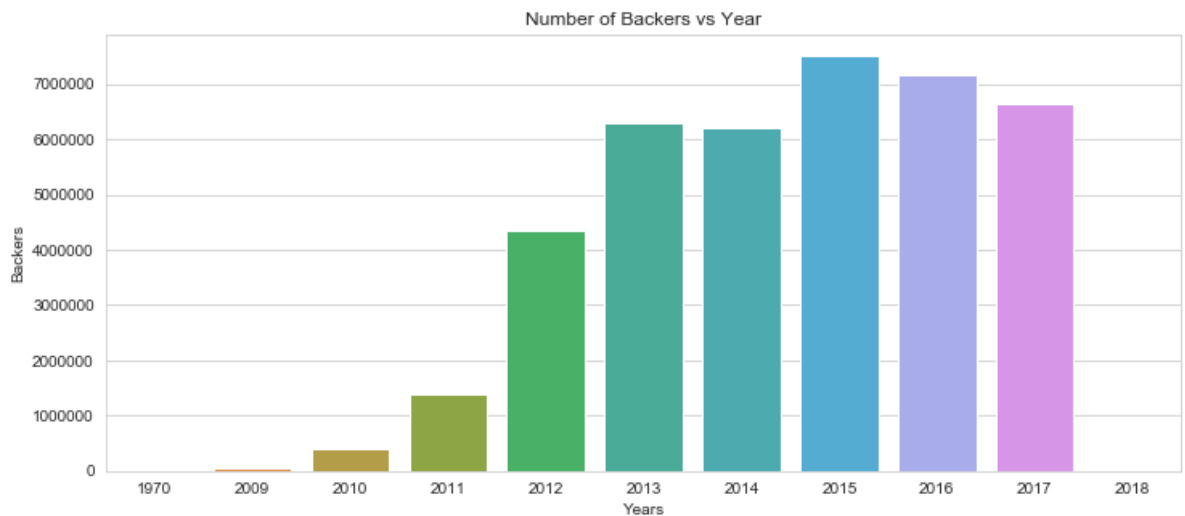
backers = pd.DataFrame(backers, columns = ["Years", "Backers"])
backers
```

Out[74]:

	Years	Backers
0	2015	7512058
1	2014	6194576
2	2016	7148604
3	2017	6653360
4	2013	6292568
5	2012	4343786
6	2011	1396473
7	2010	406875
8	2009	43758
9	2018	1161
10	1970	0

```
In [75]: plt.figure(figsize = (12, 5))
plt.title("Number of Backers vs Year")
sns.barplot(x = "Years", y = "Backers", data = backers)
```

```
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x126c8a940>
```



Above we have represented two graphs that indicate the distribution of the number of projects per year, with the difference that in one in the y axis there are the number of Campaigns while in the other there are the number of Backers

Comparison between Data and its Plot

```
In [76]: datasuccess = df_kickstarter[df_kickstarter.state == "successful"]
datafail = df_kickstarter[df_kickstarter.state == "failed"]

suc_categories = datasuccess.groupby("main_category")["usd_pledged_real"].sum()
suc_categories = suc_categories/1000000

failed_categories = datafail.groupby("main_category")["usd_pledged_real"].sum()
failed_categories = failed_categories/1000000
```

```
In [77]: failed_categories_goals = datafail.groupby("main_category")["usd_goal_real"].sum()
failed_categories_goals = failed_categories_goals/1000000

success_categories_goals = datasuccess.groupby("main_category")["usd_goal_real"].sum()
success_categories_goals = success_categories_goals/1000000
```

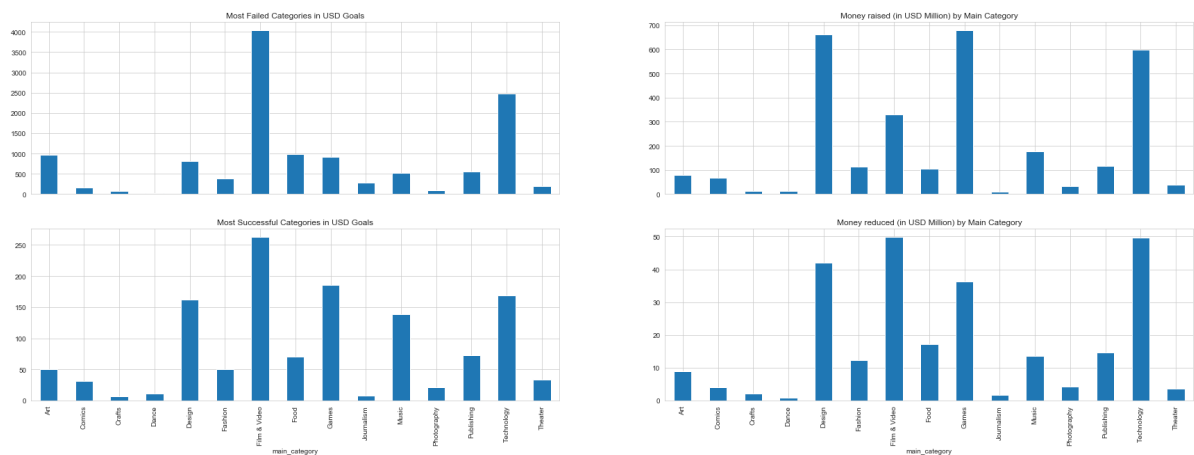
```
In [78]: f, axarr = plt.subplots(2,2, figsize=(30, 10), sharex=True)
ax1 = failed_categories_goals.plot.bar(ax=axarr[0][0])
ax1.set_title("Most Failed Categories in USD Goals")

ax2 = success_categories_goals.plot.bar(ax=axarr[1][0])
ax2.set_title("Most Successful Categories in USD Goals")

ax3 = suc_categories.plot.bar(ax=axarr[0][1])
ax3.set_title("Money raised (in USD Million) by Main Category")

ax4 = failed_categories.plot.bar(ax=axarr[1][1])
ax4.set_title("Money reduced (in USD Million) by Main Category")
```

```
Out[78]: Text(0.5, 1.0, 'Money reduced (in USD Million) by Main Category')
```



Concluding our graphic analysis through these summary diagrams we wanted to underline mainly the number of failed and successful campaigns for each category, in particular some categories such as Games and Theater has a higher percentage of successful projects, unlike others such as for example technology and fashion, which have numerous failed projects.

Kickstarter Projects

Notebook 03: Data Preparation

Group's members:

- Crnigoj Gabriele 134176
- Ferraro Tommaso 132998
- Stinat Kevin 134905

Data Loading for Data Prediction and Clustering

We load and dump the file with `joblib` instead of "pickle" because we want to load and dump the data in the right way in order to avoid the possibilities of errors during the reading step.

```
In [1]: import numpy as np
import pandas as pd
import joblib
import pickle
import os
from tqdm import tqdm
from tempfile import mkdtemp
```

```
In [2]: savedir = 'Kickstarter_Dataframe'
filename = os.path.join(savedir, 'Kikstarter_Backup_File')

with open(filename, 'rb') as r:
    df_prediction = joblib.load(r)

df_prediction.shape
```

```
Out[2]: (378661, 22)
```

Clustering: Columns Analysis

First we asked ourselves about the titles: in particular we wanted to understand if there was any correlation between the frequency of appearance of the words in the titles and the final state of the project. For this purpose we used the `nltk` library and the libraries related to it (i.e.

`TfidfVectorizer`, `stopwords`, `tokenize` and `stem`, ...) to analyze the columns of titles in the `DataFrame`. The results of the analysis is inconclusive: the frequency of appearance of the words is less than 0.0001% and for this reason, we can't consider that column significative for our analysis. We drop it from the `DataFrame` that we will use for our next analysis.

```
import nltk
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('stopwords')
nltk.download('punkt')

stopwords = nltk.corpus.stopwords.words('english')
stemmer = SnowballStemmer('english')

def tokenize_and_stem(synopsis):

    lists_tokenized = []

    synopsis = synopsis.lower() #Don't want capital letters

    sent_tokens = nltk.sent_tokenize(synopsis) #tokenizes the synopsis
into sentences in a list called sent_tokenized

    for sentence in sent_tokens:
        list_tokenized = nltk.word_tokenize(sentence) #tokenizes the wo
rds in a list called list_tokenized
        lists_tokenized.extend(list_tokenized)

    list_stemmed = list(stemmer.stem(i) for i in lists_tokenized) #list
with every token stemmed (the root of the word)

    only_words = list(filter(lambda x: x.isalpha(), list_stemmed)) #it
mantainces only words and nothing else

    without_stopwords = list(filter(lambda x: x not in stopwords, only_
words))
    #if lambda is true, keep the word in the liste, otherwise remove it

    return without_stopwords

tfidf_vectorizer_vectors = TfidfVectorizer(max_features = 350, max_df=0
.8, min_df=0.0001, analyzer='word', tokenizer = tokenize_and_stem)

tfidf_vectorizer_vectors = tfidf_vectorizer.fit_transform(df['name'])
tfidf_vectorizer_vectors
```

Data Preparation and Cleaning for Prediction

First we changed the name of some columns for greater convenience. Furthermore, we faced with a huge limitation: the computation times and the memory occupied during clustering and prediction operations. For this reason we were forced to find a sub DataFrame that it was possible to manage with our computers. But we don't select a random portion of the DataFrame with the function `.sample()`, we clean the inconsistent parts of our DataFrame comparing the columns and the rows with the previous analysis in the Notebook_02.

First we delete the columns 'deadline', 'launched', 'name', 'ID', 'Launch_Day', 'Deadline_Day', 'goal', 'pledged', 'usd pledged', because they are repeated in other columns or are meaningless for our analysis.

Furthermore, we deleted from the DataFrame the rows and consequently the columns of the projects that are not started in the country US and the projects that haven't the USD currency. The explanation of our behaviour can be found in the Notebook_02: the projects with the most percentage to be successful and numerous number of projects belong to the two categories expressed above.

The same reason is behind our choice to focus only 'Product Design', 'Documentary', 'Music', 'Tabletop Games', 'Shorts', 'Food', 'Film & Video': those categories have more chance to be successful than the others that are excluded by our analysis.

We also concentrate our analysis on the states 'failed', 'successful', 'canceled' because they are the most meaningful, indeed we do not consider the rows with NaN or/and Null values.

```
In [3]: #Let's drop and rename the columns which are useless for our analysis
df_prediction.rename(columns = {'MDeadline_Month':'Deadline_Month',
                                'crowdfunding_period':'period_days'}, inplace = True)
df_prediction = df_prediction.drop(columns = ['deadline', 'launched',
                                             'name', 'ID', 'Launch_Day', 'Deadline_Day'])
```

```
In [4]: df_prediction.drop(df_prediction[df_prediction.currency != 'USD'].index, inplace=True)
df_prediction.drop(df_prediction[df_prediction.country != 'US'].index, inplace=True)
df_prediction = df_prediction.drop(columns=['goal', 'pledged', 'country', 'usd pledged', 'currency'])
```

```
In [5]: main_category = ['Product Design', 'Documentary', 'Music', 'Tabletop Games', 'Shorts', 'Food', 'Film & Video']
other_states = ['failed', 'successful', 'canceled']

df = pd.DataFrame(columns = df_prediction.columns)
df_ks = pd.DataFrame(columns = df.columns)

for i in tqdm(main_category):
    temp = df_prediction
    temp = temp.drop(temp[temp.category != i ].index)
    df = df.append(temp)

for i in tqdm(other_states):
    temp = df
    temp = temp.drop(temp[temp.state != i ].index)
    df_ks = df_ks.append(temp)

100%|██████████| 7/7 [00:04<00:00, 1.46it/s]
100%|██████████| 3/3 [00:00<00:00, 3.21it/s]
```

```
In [6]: df_ks.shape
```

```
Out[6]: (79476, 11)
```

```
In [7]: df_ks.nunique()
```

```
Out[7]: category          7
main_category          5
state                  3
backers                2601
usd_pledged_real      26380
usd_goal_real         2910
Launch_Year           10
Launch_Month          12
Deadline_Year         10
Deadline_Month        12
period_days           95
dtype: int64
```

```
In [8]: filename = os.path.join(savedir, 'Kikstarter_Backup_File_Prova_Predizione')

with open(filename, 'wb') as r:
    joblib.dump(df_ks , r)
```

Data Preparation and Cleaning for Clustering

In first place we encode categorical features as a one-hot numeric array. `OneHotEncoder` is a function in which the input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. With this function This creates a binary column for each category and returns a sparse matrix or dense array (depending on the `sparse` parameter) By default, the encoder derives the categories based on the unique values in each feature. Alternatively, you can also specify the categories manually. This encoding is needed for feeding categorical data to the analysis of `Clustering` because it needs only Ordinal values.

We also normalized our data with the Function `.StandardScaler()` because our data aren't only binary and we need to normalized them to avoid that some values would have greater weight than others.

For the second part of our analysis, we use `Principal Component Analysis (PCA)`, that is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation. One important thing to note about PCA is that it is an Unsupervised dimensionality reduction technique, you can cluster the similar data points based on the feature correlation between them without any supervision (or labels)

Why does we apply `PCA` ? For Data Visualization: When working on any data related problem, the challenge in today's world is the sheer volume of data, and the variables/features that define that data. To solve a problem where data is the key, you need extensive data exploration like finding out how the variables are correlated or understanding the distribution of a few variables. Considering that there are a large number of variables or dimensions along which the data is distributed, visualization can be a challenge and almost impossible. Our main purpose was to find how much columns of our DataFrame can be deleted to streamline it.

```
In [9]: from sklearn.preprocessing import OneHotEncoder
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA
        import matplotlib.pyplot as plt
        import seaborn as sns
        from scipy import stats
```

```
In [10]: #change the type af the last date columns
df_ks = df_ks.fillna(0)

OneHotEncoder_obj = OneHotEncoder(sparse = False)
df_OneHotEncoder = pd.DataFrame(OneHotEncoder_obj.fit_transform(df_ks[
    ['category', 'main_category', 'state']]))
df_OneHotEncoder.shape
```

```
Out[10]: (79476, 15)
```

```
In [11]: categories_list = []

for i in OneHotEncoder_obj.categories_:
    for k in i:
        categories_list.append(k)

df_OneHotEncoder.columns = categories_list
```

```
In [12]: #create a new dataframe more suitable for our analysis
df_ks = df_ks.drop(columns=['category', 'main_category', 'state'])
df_ks = df_ks.reset_index()
df_ks = df_ks.drop(columns=['index'])
df_concat = pd.concat([df_ks, df_OneHotEncoder], axis = 1)

#and normalized it
x = StandardScaler().fit_transform(df_concat)
x_normalized = pd.DataFrame(x)
x_normalized.columns = df_concat.columns
x_normalized.head()
```

Out[12]:

	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launch_Month	Deadline_Year
0	-0.121874	-0.098294	-0.041742	1.191785	1.094765	1.176878
1	-0.076002	-0.079780	0.106267	-0.762258	0.792978	-0.809679
2	-0.124786	-0.099306	-0.031043	0.703275	-1.319529	0.680239
3	-0.117505	-0.094087	-0.030448	0.703275	0.491191	0.680239
4	-0.124786	-0.099306	-0.032826	0.703275	-1.017743	0.680239

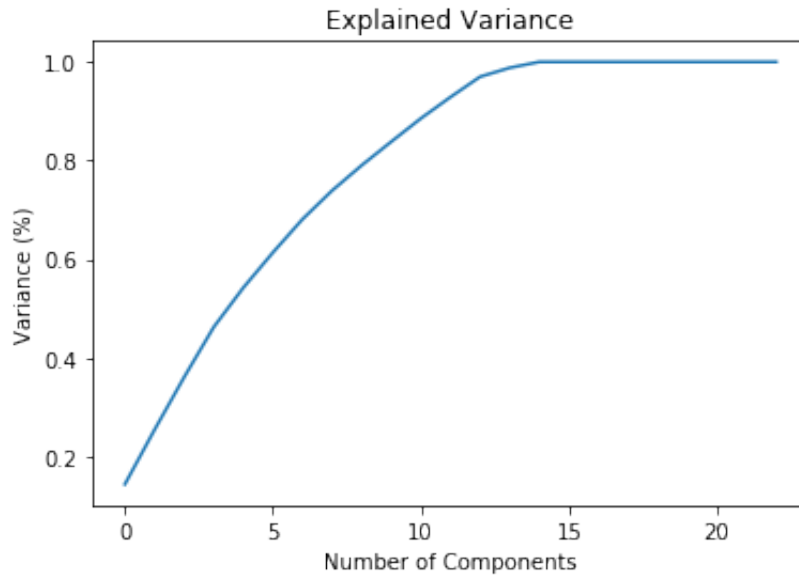
5 rows × 23 columns

```
In [13]: pca = PCA()

principalComponents = pca.fit(x_normalized)
principalComponents
```

Out[13]: PCA(copy=True, iterated_power='auto', n_components=None, random_state=None, svd_solver='auto', tol=0.0, whiten=False)

```
In [14]: plt.figure()
plt.plot(np.cumsum(principalComponents.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Explained Variance')
plt.show()
```



```
In [15]: filename = os.path.join(savedir, 'Kikstarter_Backup_File_Prova')

with open(filename, 'wb') as r:
    joblib.dump(x_normalized , r)
```

Conclusion of Notebook 03

As we can see from the Explained Variance graph above, the number of columns that PCA and the number of columns that we have decided to choose are more or less the same. For this reason, we decided to proceed to dump the new DataFrame that we have obtained for the next analysis.

Kickstarter Projects

Notebook 04: Clustering

Group's members:

- Crnigoj Gabriele 134176
- Ferraro Tommaso 132998
- Stinat Kevin 134905

Data Loading for Clustering

```
In [111]: import numpy as np
import pandas as pd
import joblib
import pickle
import os
from tqdm import tqdm
from tempfile import mkdtemp
```

```
In [112]: os.path.exists('Kickstarter_Dataframe')
```

```
Out[112]: True
```

```
In [166]: savedir = 'Kickstarter_Dataframe'
filename = os.path.join(savedir, 'Kikstarter_Backup_File_Prova')

with open(filename, 'rb') as r:
    df = joblib.load(r)

df.shape
```

```
Out[166]: (79476, 23)
```


In [167]: `df.head()`

Out[167]:

	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launch_Month	Deadline_Year
0	-0.121874	-0.098294	-0.041742	1.191785	1.094765	1.176878
1	-0.076002	-0.079780	0.106267	-0.762258	0.792978	-0.809679
2	-0.124786	-0.099306	-0.031043	0.703275	-1.319529	0.680239
3	-0.117505	-0.094087	-0.030448	0.703275	0.491191	0.680239
4	-0.124786	-0.099306	-0.032826	0.703275	-1.017743	0.680239

5 rows × 23 columns

Data Loading for Analysis

```
In [168]: savedir = 'Kickstarter_Dataframe'
filename = os.path.join(savedir, 'Kikstarter_Backup_File_Prova_Pred
           izione')

           with open(filename, 'rb') as r:
               df_analysis = joblib.load(r)

           df_analysis.shape
```

Out[168]: (79476, 11)

```
In [169]: df_1 = df_analysis[1500:4500]
df_2 = df_analysis[30000:33000]
df_3 = df_analysis[45000:48000]
df_4 = df_analysis[78000:79476]

df_clustering_2 = pd.DataFrame(columns = df_analysis.columns)

df_clustering_2 = df_clustering_2.append(df_1)
df_clustering_2 = df_clustering_2.append(df_2)
df_clustering_2 = df_clustering_2.append(df_3)
df_clustering_2 = df_clustering_2.append(df_4)

df_clustering_2.drop(df_clustering_2.index[10021], inplace=True)
df_clustering_2.drop(df_clustering_2.index[10116], inplace=True)

df_clustering_2 = df_clustering_2.reset_index()
df_clustering_2 = df_clustering_2.drop(columns = 'index')
```

In [170]: `df_analysis.head()`

Out[170]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Year
103	Product Design	Design	failed	4	156.0	500.0	201
121	Product Design	Design	failed	67	3012.0	125000.0	201
177	Product Design	Design	failed	0	0.0	9500.0	201
179	Product Design	Design	failed	10	805.0	10000.0	201
212	Product Design	Design	failed	0	0.0	8000.0	201

Clustering

After a long evaluation we decide to use a lower number of rows, instead of the original 378 thousands. There are many reasons:

- we need a data sample which have to be statistical meaningful and not all all the rows
- the computational time is very reduced and we are able to apply more algorithms than before

```
In [171]: import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.cluster import AgglomerativeClustering as AC
from sklearn import metrics
```

```
In [172]: df_clustering = pd.DataFrame(columns = df.columns)

df_1 = df[1500:4500]
df_2 = df[30000:33000]
df_3 = df[45000:48000]
df_4 = df[78000:79476]

df_clustering = df_clustering.append(df_1)
df_clustering = df_clustering.append(df_2)
df_clustering = df_clustering.append(df_3)
df_clustering = df_clustering.append(df_4)

df_clustering.drop(df_clustering.index[10021], inplace=True)
df_clustering.drop(df_clustering.index[10116], inplace=True)

df_clustering = df_clustering.reset_index()
df_clustering = df_clustering.drop(columns=['index'])
```

```
In [173]: df_clustering.shape
```

```
Out[173]: (10474, 23)
```

```
In [174]: df_clustering.head()
```

```
Out[174]:
```

	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launch_Month	Deadline_Year
0	-0.106583	-0.096233	-0.030448	0.703275	-0.414169	0.680239
1	-0.045420	-0.042401	-0.030448	1.191785	1.396552	1.176878
2	-0.117505	-0.098690	0.017105	-0.762258	-0.414169	-0.809679
3	-0.104399	-0.040690	-0.006672	1.680296	0.491191	1.673517
4	-0.114593	-0.083559	-0.036393	0.214764	1.396552	0.680239

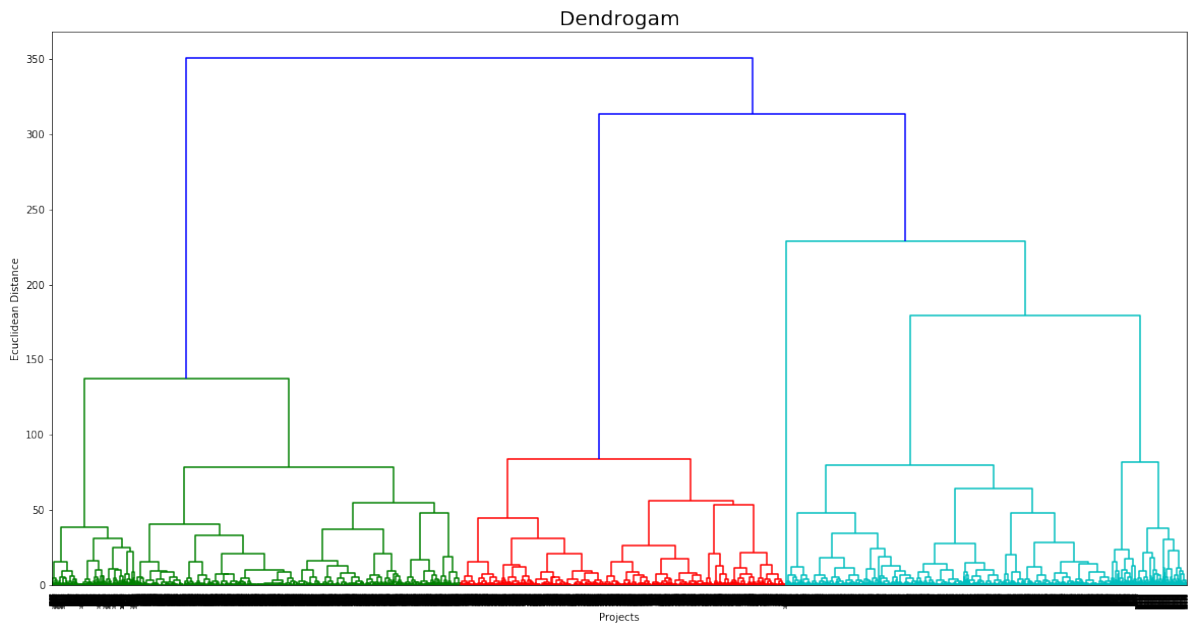
5 rows × 23 columns

Hierarchical Clustering

As first, we decide to do a hierarchical cluster analysis. The parameter linkage is set to 'ward' which is the best in our case. Our aim is to obtain a different cluster division which isn't base on, for example, the attributes 'category' or 'main_category'. This is only a first step because we are going to explore what is the best number of clusters by further analysis with other clustering algorithms.

```
In [25]: from scipy.cluster import hierarchy as sch
from scipy.cluster.hierarchy import ward, dendrogram
```

```
In [26]: plt.rcParams['figure.figsize'] = (20, 10)
dendrogram = sch.dendrogram(sch.linkage(df_clustering, method = 'ward'))
plt.title('Dendrogram', fontsize = 20)
plt.xlabel('Projects')
plt.ylabel('Euclidean Distance')
plt.show()
```



1) Agglomerative Clustering

We decide to apply the Agglomerative Clustering algorithm with the linkage set to ward because it lets us to a better silhouette score.

Agglomerative Clustering: Ward

```
In [122]: labels_ward = AC(n_clusters = 5, affinity = 'euclidean', linkage =
labels_ward
```

```
Out[122]: array([2, 2, 2, ..., 1, 1, 1], dtype=int64)
```

```
In [123]: metrics.silhouette_score(df_clustering, labels_ward, metric = 'euclidean')
```

```
Out[123]: 0.5569742662106647
```

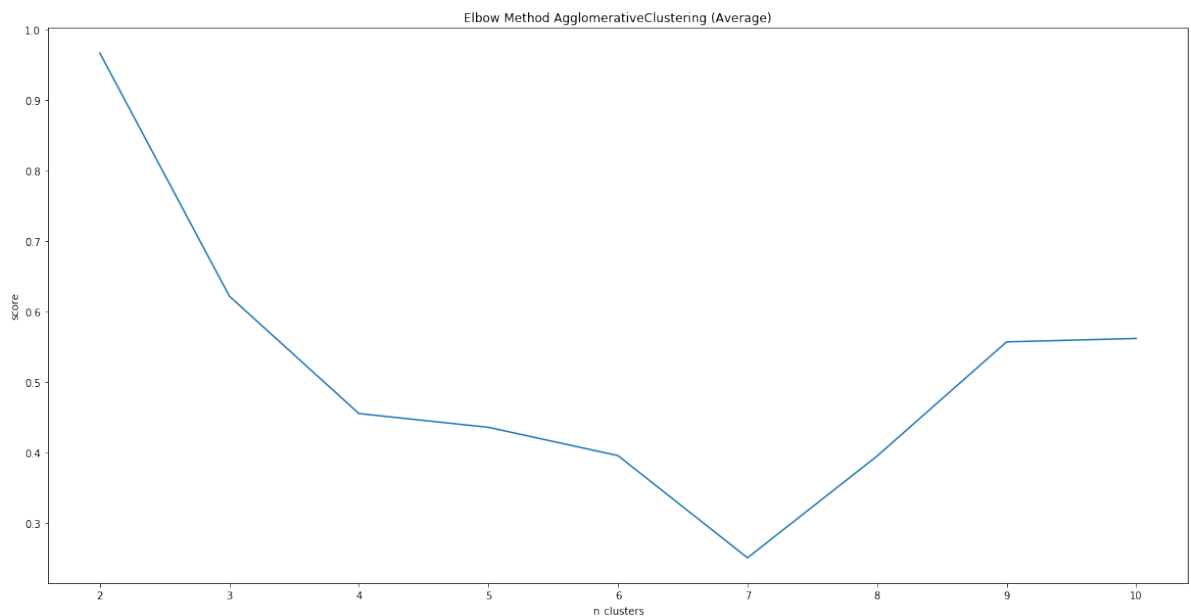
```
In [124]: silhouette_df = pd.DataFrame({'n_clusters':[x for x in range(2,11)]
    })

    for i in tqdm(range(0,9)):
        temp_clustering = AC(n_clusters = silhouette_df.n_clusters[i],
            affinity = 'euclidean',
                                linkage = 'average').fit
        _predict(df_clustering)
        silhouette_df.loc[[i], 'score'] = metrics.silhouette_score(df_clustering, temp_clustering)
```

100%|██████████| 9/9 [03:05<00:00, 20.71s/it]

```
In [125]: sns.lineplot(x = 'n_clusters', y = 'score', data = silhouette_df)
    plt.title('Elbow Method AgglomerativeClustering (Average)')
```

Out[125]: Text(0.5, 1.0, 'Elbow Method AgglomerativeClustering (Average)')



```
In [126]: labels_ward = AC(n_clusters = 7, affinity = 'euclidean', linkage =
    'ward').fit_predict(df_clustering)
    metrics.silhouette_score(df_clustering, labels_ward, metric = 'euclidean')
```

Out[126]: 0.46812296693408517

With the Elbow Method it seems that the best number of clusters is 9 for our dataframe.

Plotting Agglomerative Clustering with PCA()

Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation which converts a set of correlated variables to a set of uncorrelated variables. PCA is an unsupervised statistical technique used to examine the interrelations among a set of variables. PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. It is a simpler method for analysis

```
In [127]: from sklearn.decomposition import PCA
```

```
In [128]: ward = pd.DataFrame(labels_ward)
df = pd.concat([df_clustering, ward], axis = 1)
df.rename(columns = {0:'Clusters'}, inplace=True)
df.head(2)
```

Out[128]:

	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launch_Month	Deadline_Year
0	-0.106583	-0.096233	-0.030448	0.703275	-0.414169	0.680239
1	-0.045420	-0.042401	-0.030448	1.191785	1.396552	1.176878

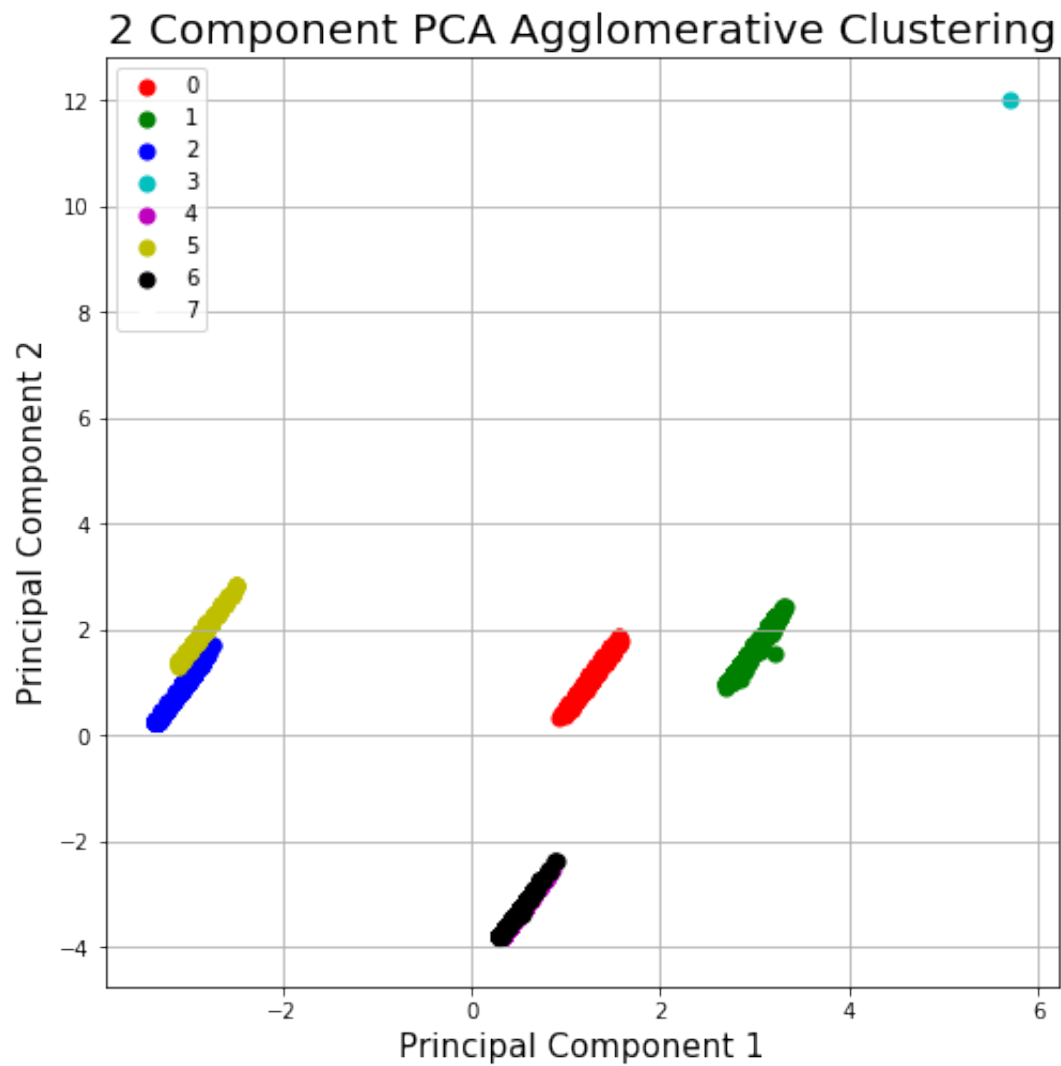
2 rows x 24 columns

```
In [129]: #PCA() application
pca_ward = PCA(n_components=2)
principalComponents = pca_ward.fit_transform(df_clustering)

principalDf = pd.DataFrame(data = principalComponents, columns = ['
principal component 1', 'principal component 2'])

df[['Clusters']].head()
finalDf = pd.concat([principalDf, df[['Clusters']]], axis = 1)

#Plot
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA Agglomerative Clustering', fontsize =
20)
targets = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k', 'w']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Clusters'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2'
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



```
In [130]: df_ACW = pd.concat([df_clustering_2, ward], axis = 1)
df_ACW.rename(columns = {0:'Clusters'}, inplace=True)
df_ACW.head(2)
```

Out[130]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Year
0	Product Design	Design	failed	25	474.0	10000.0	2015
1	Product Design	Design	failed	109	8778.0	10000.0	2016


```
In [131]: df_ACW.groupby('Clusters')['category'].count()
```

```
Out[131]: Clusters
0      699
1     3000
2     2999
3         1
4     1678
5      775
6     1322
Name: category, dtype: int64
```

```
In [132]: target_not_reached = df_ACW.loc[lambda df_ACW: df_ACW['Clusters'] =
= 3]
target_not_reached
```

```
Out[132]:
```

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launc
10115	Film & Video	Film & Video	canceled	0	0.0	700.0	

```
In [134]: target_not_reached = df_ACW.loc[lambda df_ACW: df_ACW['Clusters'] =
= 5]
target_not_reached.head()
```

```
Out[134]:
```

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launc
9242	Food	Food	canceled	0	0.0	25000.0	
9243	Food	Food	canceled	0	0.0	10000.0	
9244	Food	Food	canceled	0	0.0	200.0	
9245	Food	Food	canceled	9	1230.0	30000.0	
9246	Food	Food	canceled	6	875.0	22000.0	

2) KMeans

In order to improve our analysis, we decide to use also a K-Means method of clustering. We decide to use it because its silhouette score is a little better than the previous AgglomerativeClustering's one.

```
In [82]: from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 5 ,init='k-means++', n_init = 10 ,max_
iter=300,
                tol=0.0001, algorithm='elkan')

km_clusters = kmeans.fit_predict(df_clustering)
```

```
In [83]: metrics.silhouette_score(df_clustering, km_clusters)
```

```
Out[83]: 0.34144237195545457
```

```
In [84]: silhouette_kmeans = pd.DataFrame({'n_clusters':[x for x in range(2,
11)]})

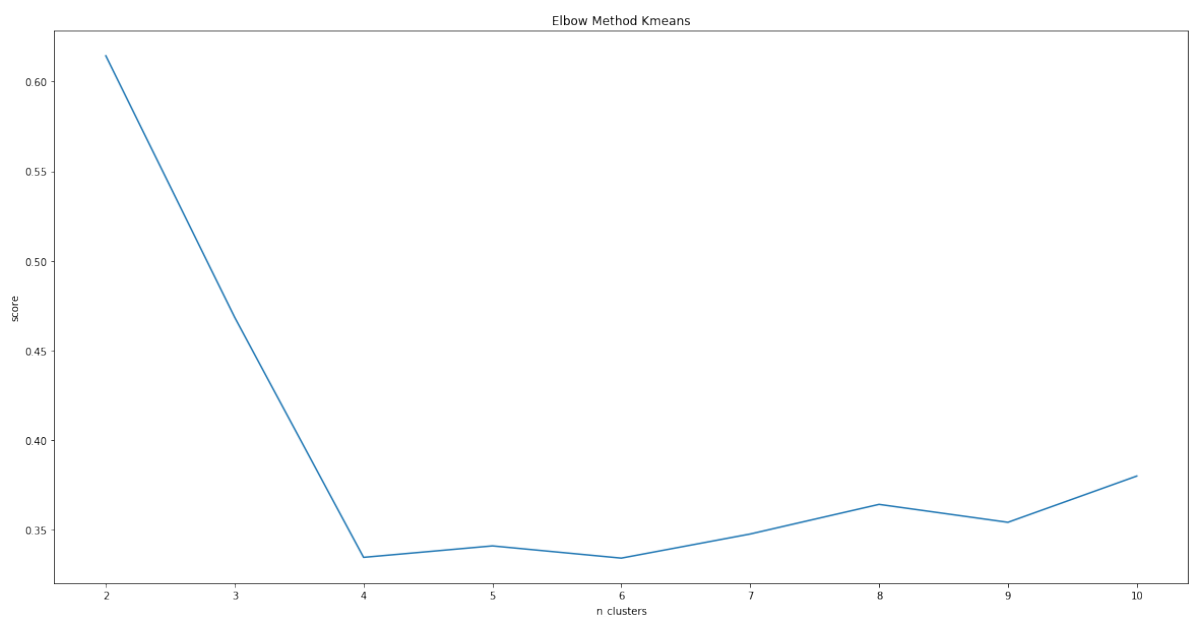
for i in tqdm(range(0,9)): #the dataframe 8 rows

    temp_clustering = KMeans(n_clusters = silhouette_kmeans.n_clusters[i],
random_state = 0, algorithm = 'elkan').fit_predict(df_clustering)
    silhouette_kmeans.loc[[i], 'score'] = metrics.silhouette_score(df_clustering,
temp_clustering)

100%|██████████| 9/9 [00:06<00:00, 1.49it/s]
```

```
In [85]: sns.lineplot(x='n_clusters', y='score', data=silhouette_kmeans)
plt.title('Elbow Method Kmeans')
```

```
Out[85]: Text(0.5, 1.0, 'Elbow Method Kmeans')
```



```
In [142]: kmeans_2 = KMeans(n_clusters = 4 ,init='k-means++', n_init = 10 ,max_iter=300,
                             tol=0.0001, random_state= 111 , algorithm
                             ='elkan')

km_clusters_2 = kmeans_2.fit_predict(df_clustering)
metrics.silhouette_score(df_clustering, km_clusters_2)
```

Out[142]: 0.5155902414341663

It seems that the best silhouette score is when we are using 9 clusters instead of 7. This is the same result that we found with AgglomerativeClustering method.

Plot KMeans with PCA()

```
In [87]: from sklearn.decomposition import PCA
import seaborn as sns
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
```

```
In [152]: ward = pd.DataFrame(km_clusters_2)
df = pd.concat([df_clustering, ward], axis = 1)
df.rename(columns = {0:'Clusters'}, inplace=True)
df.head(2)
```

Out[152]:

	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launch_Month	Deadline_Year
0	-0.106583	-0.096233	-0.030448	0.703275	-0.414169	0.680239
1	-0.045420	-0.042401	-0.030448	1.191785	1.396552	1.176878

2 rows × 24 columns

```
In [153]: my_dpi=96
plt.figure(figsize=(480/my_dpi, 480/my_dpi), dpi=my_dpi)

# Keep the 'specie' column appart + make it numeric for coloring
df['Clusters']=pd.Categorical(df['Clusters'])
my_color=df['Clusters'].cat.codes
df = df.drop('Clusters', 1)

pca = PCA(n_components=3)
pca.fit(df_clustering)

result=pd.DataFrame(pca.transform(df_clustering), columns=['PCA%i'
% i for i in range(3)], index=df.index)
result.head()
```

Out[153]:

	PCA0	PCA1	PCA2
0	0.465797	-3.428334	0.044780
1	0.368006	-3.616901	-0.111511
2	0.685832	-2.904276	0.492865
3	0.305405	-3.781869	-0.205442
4	0.505038	-3.348919	0.463524

<Figure size 480x480 with 0 Axes>

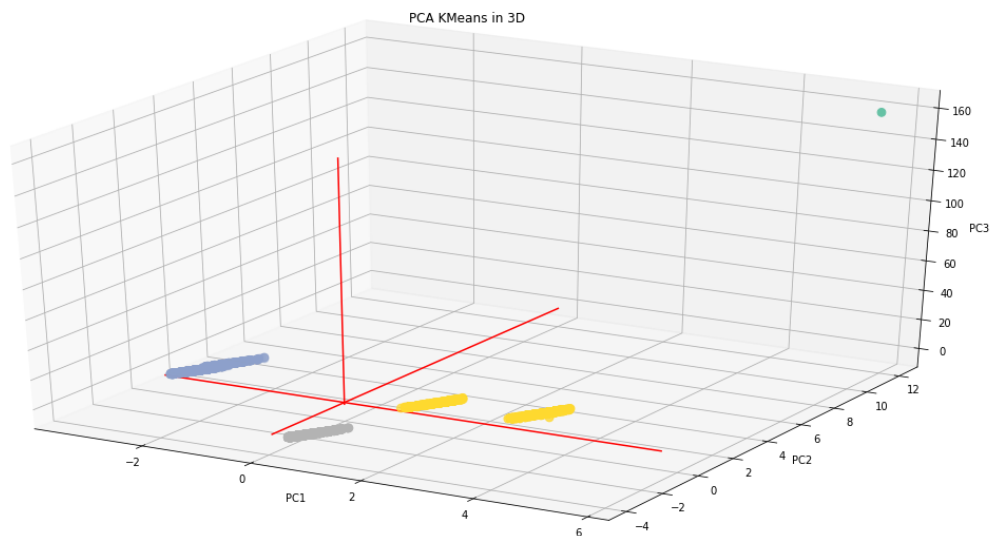
```
In [154]: # Plot initialisation
fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')
ax.scatter(result['PCA0'], result['PCA1'], result['PCA2'], c=my_color, cmap="Set2_r", s=60)

# make simple, bare axis lines through space:
xAxisLine = ((min(result['PCA0']), max(result['PCA0'])), (0, 0), (0, 0))
ax.plot(xAxisLine[0], xAxisLine[1], xAxisLine[2], 'r')
yAxisLine = ((0, 0), (min(result['PCA1']), max(result['PCA1'])), (0, 0))
ax.plot(yAxisLine[0], yAxisLine[1], yAxisLine[2], 'r')
zAxisLine = ((0, 0), (0, 0), (min(result['PCA2']), max(result['PCA2']))))
ax.plot(zAxisLine[0], zAxisLine[1], zAxisLine[2], 'r')

ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.set_title("PCA KMeans in 3D")

plt.show()
```



```
In [155]: df_ACW = pd.concat([df_clustering_2, ward], axis = 1)
df_ACW.rename(columns = {0: 'Clusters'}, inplace=True)
df_ACW.head(2)
```

Out[155]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Year
0	Product Design	Design	failed	25	474.0	10000.0	2015
1	Product Design	Design	failed	109	8778.0	10000.0	2016

```
In [159]: df_ACW.groupby('Clusters')['category'].count()
```

Out[159]: Clusters
0 3000
1 3698
2 3775
3 1
Name: category, dtype: int64

```
In [182]: target_not_reached = df_ACW.loc[lambda df_ACW: df_ACW['Clusters'] = 0]
target_not_reached.head()
```

Out[182]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Year
0	Product Design	Design	failed	25	474.0	10000.0	2015
1	Product Design	Design	failed	109	8778.0	10000.0	2016
2	Product Design	Design	failed	10	95.0	50000.0	2012
3	Product Design	Design	failed	28	9042.0	30000.0	2017
4	Product Design	Design	failed	14	2429.0	5000.0	2014

```
In [183]: target_not_reached = df_ACW.loc[lambda df_ACW: df_ACW['Clusters'] =
= 1]
target_not_reached.head()
```

Out[183]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	L
6000	Documentary	Film & Video	successful	302	31745.0	30000.0	
6001	Documentary	Film & Video	successful	41	2727.0	2500.0	
6002	Documentary	Film & Video	successful	179	25122.0	25000.0	
6003	Documentary	Film & Video	successful	159	46191.0	45000.0	
6004	Documentary	Film & Video	successful	12	585.0	500.0	

```
In [184]: target_not_reached = df_ACW.loc[lambda df_ACW: df_ACW['Clusters'] =
= 2]
target_not_reached.head()
```

Out[184]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Ye
3000	Food	Food	failed	2	20.0	16000.0	20
3001	Food	Food	failed	0	0.0	30.0	20
3002	Food	Food	failed	0	0.0	5000.0	20
3003	Food	Food	failed	5	96.0	15000.0	20
3004	Food	Food	failed	4	50.0	4000.0	20

```
In [160]: target_not_reached = df_ACW.loc[lambda df_ACW: df_ACW['Clusters'] =
= 3]
target_not_reached.head()
```

Out[160]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launc
10115	Film & Video	Film & Video	canceled	0	0.0	700.0	

3) DBSCAN

After the use of AgglomerativeClustering, KMeans it is now interesting to use an algorithm that doesn't require the number of clusters as input. In fact, DBSCAN algorithm views clusters as areas of high density separated by areas of low density.

So, the main DBSCAN advantages are:

- It does not require to specify the number of clusters in the data a priori (as opposed to k-means).
- It has a notion of noise, and is robust to outliers.
- It requires just two parameters and is mostly insensitive to the ordering of the points in the database.

The two main disadvantages are:

- It is not entirely deterministic: border points that are reachable from more than one cluster
- The quality of DBSCAN depends highly on the distance measure. The most common distance metric used is Euclidean distance.

```
In [161]: from sklearn.cluster import DBSCAN
          db = DBSCAN(eps = 4, min_samples = 5, metric = 'euclidean').fit_predict(df_clustering)
          db
```

```
Out[161]: array([0, 0, 0, ..., 4, 4, 4], dtype=int64)
```

```
In [162]: metrics.silhouette_score(df_clustering, db)
```

```
Out[162]: 0.5609199343329515
```

```
In [163]: df_db = pd.DataFrame(db)
```

DBSCAN's Silhouette Score is very similar to the other's.

Plot DBSCAN with PCA()


```
In [176]: ward = pd.DataFrame(df_db)
df = pd.concat([df_clustering, ward], axis = 1)
df.rename(columns = {0:'Clusters'}, inplace=True)
df.head(2)
```

Out[176]:

	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launch_Month	Deadline_Year
0	-0.106583	-0.096233	-0.030448	0.703275	-0.414169	0.680239
1	-0.045420	-0.042401	-0.030448	1.191785	1.396552	1.176878

2 rows x 24 columns

```
In [177]: my_dpi=96
plt.figure(figsize=(480/my_dpi, 480/my_dpi), dpi=my_dpi)

# Keep the 'specie' column appart + make it numeric for coloring
df['Clusters']=pd.Categorical(df['Clusters'])
my_color=df['Clusters'].cat.codes
df = df.drop('Clusters', 1)

pca = PCA(n_components=3)
pca.fit(df_clustering)

result=pd.DataFrame(pca.transform(df_clustering), columns=['PCA%i'
%i for i in range(3)], index=df.index)
result.head()
```

Out[177]:

	PCA0	PCA1	PCA2
0	0.465797	-3.428334	0.044780
1	0.368006	-3.616901	-0.111511
2	0.685832	-2.904276	0.492865
3	0.305405	-3.781869	-0.205442
4	0.505038	-3.348919	0.463524

<Figure size 480x480 with 0 Axes>

```

In [178]: # Plot initialisation
fig = plt.figure()

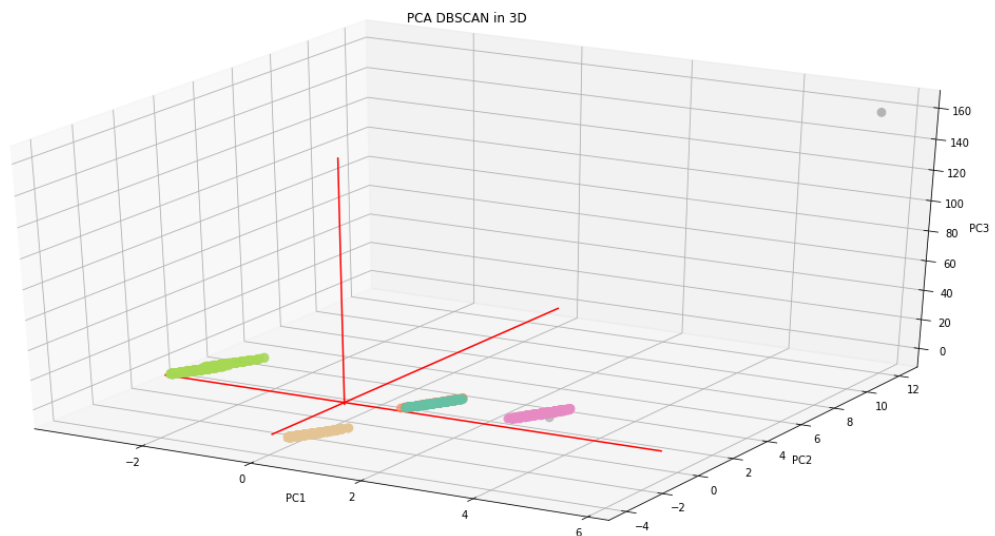
ax = fig.add_subplot(111, projection='3d')
ax.scatter(result['PCA0'], result['PCA1'], result['PCA2'], c=my_color, cmap="Set2_r", s=60)

# make simple, bare axis lines through space:
xAxisLine = ((min(result['PCA0']), max(result['PCA0'])), (0, 0), (0, 0))
ax.plot(xAxisLine[0], xAxisLine[1], xAxisLine[2], 'r')
yAxisLine = ((0, 0), (min(result['PCA1']), max(result['PCA1'])), (0, 0))
ax.plot(yAxisLine[0], yAxisLine[1], yAxisLine[2], 'r')
zAxisLine = ((0, 0), (0, 0), (min(result['PCA2']), max(result['PCA2']))))
ax.plot(zAxisLine[0], zAxisLine[1], zAxisLine[2], 'r')

ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.set_title("PCA DBSCAN in 3D")

plt.show()

```



```
In [179]: df_DBS = pd.concat([df_clustering_2, df_db], axis = 1)
df_DBS.rename(columns = {0: 'Clusters'}, inplace=True)
df_DBS.groupby('Clusters')['category'].count()
```

```
Out[179]: Clusters
-1         4
0        3000
1        3774
2        2999
3         242
4         455
Name: category, dtype: int64
```

```
In [181]: target_not_reached = df_DBS.loc[lambda df_DBS: df_DBS['Clusters'] =
= -1]
target_not_reached.head()
```

```
Out[181]:
```

	category	main_category	state	backers	usd_pledged_real	usd_goal_real
3706	Food	Food	failed	0	0.00	9592440.0
8335	Documentary	Film & Video	successful	16850	859425.69	650000.0
10115	Film & Video	Film & Video	canceled	0	0.00	700.0
10228	Film & Video	Film & Video	canceled	0	0.00	10474836.0

Conclusion

The dataframe has already a partition inside. For example, category or main_category. But we were interested to find out another one based on correlation between all the categories of the dataframe. In fact, we used all the columns to explore a possible number of alternative clustering. Observing Silhouette score and the PCA graphs, we can conclude that DBSCAN can provide to us a different view of the dataframe, although there are certain groups that are quite overlaying. From the 3D DBSCAN graph we can also claim that the green cluster is the greater and that it is quite far from the others. This means that DBSCAN was able to recognize sharply that set of data as an unicum. By contrast, Kmeans had to divided that set of data in two groups, yellow and light blue, but this doesn't mean that it is correct. The reason is that K-Means has to use all the number of groups that we gave to it as input, in that case 9 clusters. In conclusion, DBSCAN can be used successfully to obtain a new dataframe's categorization.

Kickstarter Projects

Notebook 05 : Prediction

Group's members:

- Crnigoj Gabriele 134176
- Ferraro Tommaso 132998
- Stinat Kevin 134905

Data Loading and Managing for Data Prediction

Classification is a classic data mining technique based on machine learning. Basically, classification is used to classify each item in a set of data into one of a predefined set of classes or groups. Classification method makes use of mathematical techniques such as decision trees, linear programming, neural network and statistics. In classification, we develop the software that can learn how to classify the data items into groups. In other words, the goal of classification is to accurately predict the target class for each case in the data. Note that:

- A classification task begins with a data set in which the class assignments are known.
- Classifications are discrete and do not imply order.

The simplest type of classification problem is binary classification. In binary classification, the target attribute has only two possible values: for example, yes or no. Multiclass targets have more than two values: for example, low, medium, high.

In the model build (training) process, a classification algorithm finds relationships between the values of the predictors and the values of the target. Different classification algorithms use different techniques for finding relationships. These relationships are summarized in a model, which can then be applied to a different data set in which the class assignments are unknown. Classification models are tested by comparing the predicted values to known target values in a set of test data. The historical data for a classification project is typically divided into two data sets: one for building the model; the other for testing the model.

First we proceed to load and manage the data: in particular we check the correct data loading, the `NaN` or `Null` values and we set new names to some columns.

In second place, we proceed to set the values in the categorical columns:

`Deadline_Year` , `Deadline_Month` , `Launch_Year` and `Launch_Month` . Now we are ready for the next steps of our analysis: the encoding.

```
In [1]: import numpy as np
import pandas as pd
import joblib
import pickle
import os
from tqdm import tqdm
from tempfile import mkdtemp
```

```
In [57]: savedir = 'Kickstarter_Dataframe'
filename = os.path.join(savedir, 'Kikstarter_Backup_File_Prova_Pred
izione')

with open(filename, 'rb') as r:
    df_prediction = joblib.load(r)

df_prediction.head()
```

Out[57]:

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Year
103	Product Design	Design	failed	4	156.0	500.0	201
121	Product Design	Design	failed	67	3012.0	125000.0	201
177	Product Design	Design	failed	0	0.0	9500.0	201
179	Product Design	Design	failed	10	805.0	10000.0	201
212	Product Design	Design	failed	0	0.0	8000.0	201

```
In [58]: df_prediction.shape
```

Out[58]: (79476, 11)

```
In [60]: df_prediction = df_prediction.reset_index()
df_prediction = df_prediction.drop(columns=['index'])
```

```
In [63]: df_prediction.head()
```

```
Out[63]:
```

	category	main_category	state	backers	usd_pledged_real	usd_goal_real	Launch_Year
0	Product Design	Design	failed	4	156.0	500.0	2016
1	Product Design	Design	failed	67	3012.0	125000.0	2012
2	Product Design	Design	failed	0	0.0	9500.0	2015
3	Product Design	Design	failed	10	805.0	10000.0	2015
4	Product Design	Design	failed	0	0.0	8000.0	2015

```
In [66]: df_prediction.rename(columns = {'MDeadline_Month':'Deadline_Month',
'crowdfunding_period':'period_days'}, inplace=True)
```

```
In [67]: #a little check
df_prediction.columns
```

```
Out[67]: Index(['category', 'main_category', 'state', 'backers', 'usd_pledged_real',
'usd_goal_real', 'Launch_Year', 'Launch_Month', 'Deadline_Year',
'Deadline_Month', 'period_days'],
dtype='object')
```

```
In [68]: categories_month = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

df_prediction['Deadline_Month'] = df_prediction['Deadline_Month'].astype(pd.Categorical(df_prediction['Deadline_Month'],
categories_month,ordered=True))
df_prediction['Launch_Month'] = df_prediction['Launch_Month'].astype(pd.Categorical(df_prediction['Launch_Month'],
categories_month,ordered=True))
```

```
In [69]: categories_years = [1970, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
2016, 2017, 2018]

df_prediction['Deadline_Year'] = df_prediction['Deadline_Year'].ast
ype(pd.Categorical(df_prediction['Deadline_Year'],

categories_years ,ordered=True))
df_prediction['Launch_Year'] = df_prediction['Launch_Year'].astype(
pd.Categorical(df_prediction['Launch_Year'],

categories_years ,ordered=True))
```

```
In [70]: df_prediction.dtypes
```

```
Out[70]: category          object
main_category          object
state                  object
backers                int64
usd_pledged_real      float64
usd_goal_real         float64
Launch_Year           category
Launch_Month          category
Deadline_Year         category
Deadline_Month        category
period_days           int64
dtype: object
```

Encoding

we encode all the columns of our DataFrame, paying attention to the correct category of each columns and its corresponding Encoder . Our **Goal** was to predict the final state of a Project so we encode all the columns except state in the encoding, otherwise state is put in the target_encoding .

we also restricted the predictions to only three possible states: successful , failed and canceled because those are more meaningful than the other (i.e. live). Then we proceed fitting our data and we can proceed with the core of this analysis: the prediction

```
In [71]: #OneHotEncoder is used for categorical attributes
from sklearn.preprocessing import OneHotEncoder

#OrdinalEncoder is used for ordinal attributes
from sklearn.preprocessing import OrdinalEncoder

#DataFrameMapper is used to combine the previous encoding methods
from sklearn_pandas import DataFrameMapper
```

```
In [72]: df_prediction.dtypes
```

```
Out[72]: category          object
main_category            object
state                    object
backers                   int64
usd_pledged_real         float64
usd_goal_real            float64
Launch_Year              category
Launch_Month             category
Deadline_Year            category
Deadline_Month           category
period_days              int64
dtype: object
```

```
In [73]: df_prediction.columns
```

```
Out[73]: Index(['category', 'main_category', 'state', 'backers', 'usd_pledged_real',
               'usd_goal_real', 'Launch_Year', 'Launch_Month', 'Deadline_Year',
               'Deadline_Month', 'period_days'],
              dtype='object')
```

```
In [82]: encoding = DataFrameMapper([
    ([ 'category', 'main_category'], OneHotEncoder(handle_unknown='ignore')),
    ([ 'Launch_Year'], OrdinalEncoder(categories=[list(df_prediction['Launch_Year'].cat.categories)])),
    ([ 'Launch_Month'], OrdinalEncoder(categories=[list(df_prediction['Launch_Month'].cat.categories)])),
    ([ 'Deadline_Year'], OrdinalEncoder(categories=[list(df_prediction['Deadline_Year'].cat.categories)])),
    ([ 'Deadline_Month'], OrdinalEncoder(categories=[list(df_prediction['Deadline_Month'].cat.categories)])),
    ([ 'backers', 'usd_pledged_real', 'usd_goal_real', 'period_days'], None) ])

encoding
```



```

Out[82]: DataFrameMapper(default=False, df_out=False,
                           features=[(['category', 'main_category'],
                                       OneHotEncoder(categories='auto', drop=N
one,
                                       dtype=<class 'numpy.float
64'>,
                                       handle_unknown='ignore',
                                       sparse=True)),
                                      (['Launch_Year'],
                                       OrdinalEncoder(categories=[[1970, 2009,
2010, 2011,
2012, 2013,
2014, 2015,
2016, 2017,
2018]],
                                       dtype=<class 'numpy.floa
t64'>)),
                                      (['Launch_Month'],
                                       Ordinal...
                                       dtype=<class 'numpy.floa
t64'>)),
                                      (['Deadline_Year'],
                                       OrdinalEncoder(categories=[[1970, 2009,
2010, 2011,
2012, 2013,
2014, 2015,
2016, 2017,
2018]],
                                       dtype=<class 'numpy.floa
t64'>)),
                                      (['Deadline_Month'],
                                       OrdinalEncoder(categories=[[1, 2, 3, 4,
5, 6, 7, 8,
9, 10, 11,
12]],
                                       dtype=<class 'numpy.floa
t64'>)),
                                      (['backers', 'usd_pledged_real', 'usd_go
al_real',
                                       'period_days'],
                                       None)],
                           input_df=False, sparse=False)

```

```

In [84]: df_prediction['state'] = df_prediction['state'].map(lambda s: 'succ
essful' if s.startswith("successful")
else s
if s.startswith("failed")
else s
if s.startswith("canceled")
else "u
ndefined")

```

```
In [85]: df_prediction.state.unique()
```

```
Out[85]: array(['failed', 'successful', 'canceled'], dtype=object)
```

```
In [86]: encoding_target = DataFrameMapper([(['state'], OrdinalEncoder()) ]
        )

        encoding_target
```

```
Out[86]: DataFrameMapper(default=False, df_out=False,
                          features=[(['state'],
                                     OrdinalEncoder(categories='auto',
                                                     dtype=<class 'numpy.float64'>))],
                          input_df=False, sparse=False)
```

```
In [87]: encoding.fit(df_prediction)
```

```

Out[87]: DataFrameMapper(default=False, df_out=False,
                           features=[(['category', 'main_category'],
                                       OneHotEncoder(categories='auto', drop=N
one,
                                       dtype=<class 'numpy.float
64'>,
                                       handle_unknown='ignore',
                                       sparse=True)),
                           (['Launch_Year'],
                               OrdinalEncoder(categories=[[1970, 2009,
2010, 2011,
2012, 2013,
2014, 2015,
2016, 2017,
2018]],
                                       dtype=<class 'numpy.floa
t64'>)),
                           (['Launch_Month'],
                               Ordinal...
                                       dtype=<class 'numpy.floa
t64'>)),
                           (['Deadline_Year'],
                               OrdinalEncoder(categories=[[1970, 2009,
2010, 2011,
2012, 2013,
2014, 2015,
2016, 2017,
2018]],
                                       dtype=<class 'numpy.floa
t64'>)),
                           (['Deadline_Month'],
                               OrdinalEncoder(categories=[[1, 2, 3, 4,
5, 6, 7, 8,
9, 10, 11,
12]],
                                       dtype=<class 'numpy.floa
t64'>)),
                           (['backers', 'usd_pledged_real', 'usd_go
al_real',
                               'period_days'],
                               None)],
                           input_df=False, sparse=False)

```

```

In [88]: encoding_target.fit(df_prediction)

```

```

Out[88]: DataFrameMapper(default=False, df_out=False,
                           features=[(['state'],
                                       OrdinalEncoder(categories='auto',
                                       dtype=<class 'numpy.floa
t64'>))],
                           input_df=False, sparse=False)

```

Classification

Classification makes use of mathematical techniques such as decision trees, linear programming, neural network and statistics. The goal is to develop the software that can learn how to classify the data items into groups. We decide to start our analysis with the use of a basic decision tree classifier. The iter is:

1) Encode the dataframe.\ 2) Separate the dataframe in groups with the `train_test_split` function.\ 3) Fit the model on the training groups.\ 4) Predict the test group.\ 5) Understand, with the accuracy score, how much the prediction differs from the reality.

First, we make an example of classification separating target columns from those which have to be predicted. In the next paragraph, we use `Pipeline` function in ordine to do a more efficient and complete analysis.

As first approach to classification we tried to use a `sklearn.tree.DecisionTreeClassifier` from sklearn library. This methodology first of all requires to sub-divide our data into two sets, one to train the algorithm and one to test it: for this first step the function

`sklearn.model_selection.train_test_split` comes really handy, also allowing the best division strategy. After several tries, the best division resulted to be the one with the set as below:

```
train_test_split(x, y, stratify = y)
```

```
In [89]: from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import accuracy_score
```

```
In [90]: X = encoding.transform(df_prediction.loc[:, df_prediction.columns != "state"])
```

```
In [91]: y = encoding_target.transform(df_prediction[['state']])
```

```
In [92]: X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
```

```
In [93]: #create a DecisionTreeClassifier object
        tree = DecisionTreeClassifier()
```

```
In [94]: tree.fit(X_train, y_train)
```

```
Out[94]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion
                                = 'gini',
                                max_depth=None, max_features=None, max_leaf
                                _nodes=None,
                                min_impurity_decrease=0.0, min_impurity_spl
                                it=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort='depr
                                ecated',
                                random_state=None, splitter='best')
```

```
In [95]: y_pred = tree.predict(X_test)
```

```
In [96]: accuracy_score(y_test, y_pred)
```

```
Out[96]: 0.8418642105792944
```

Pipeline and Cross_Validate Method

We managed to reach a value of accuracy of 0.84, meaning that 84 out of 100 sample are correctly predicted: as first try, this value is already quite satisfying for a classification on three levels. However, the application of better and more sophisticated techniques may lead to even better results. To continue our analysis the classification can be performed again using the

`sklearn.model_selection.cross_validate` function that iterates the classification for a pre-defined number of times, in order to limit problems like overfitting, underfitting and get an insight on how the model will generalize to an independent data set. In addition to that, different classifiers are tried, such as the `sklearn.svm.LinearSVC` and the ensemble classifier `sklearn.ensemble.RandomForestClassifier` and `sklearn.ensemble.AdaBoostClassifier`.

In order to make our analysis simpler, we decide to create some pipeline with different classification algorithms. They are set in steps. The first is the encoding step, where we recycle the encoding function. Secondly, we apply the classificatory. Finally we use the `Cross_Validate` function to understand which one is the most effective algorithm.

```
In [97]: from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ("preprocess", encoding),
    ("decision_tree", DecisionTreeClassifier())
])
```

```
In [98]: pipeline.fit(df_prediction, y) #we don't separate train and test, this is the reason of our accuracy result
y_pred = pipeline.predict(df_prediction)
accuracy_score(y_pred, y)
```

```
Out[98]: 0.997798077407016
```

```
In [99]: from sklearn.model_selection import cross_validate
results = cross_validate(pipeline, df_prediction, y, cv=10, return_train_score=False)
results
```

```
Out[99]: {'fit_time': array([2.28838086, 2.81809449, 2.15131044, 2.00835562,
                             2.26927352,
                             2.15631366, 2.10231876, 2.09333491, 2.00535059, 1.86540127
                             ]),
          'score_time': array([0.09297252, 0.14895272, 0.06897736, 0.07097769,
                                0.07597542,
                                0.0709784 , 0.06897831, 0.07097864, 0.07397628, 0.06897831
                                ]),
          'test_score': array([0.79214897, 0.79416205, 0.80045294, 0.84763463,
                                0.85606442,
                                0.74874182, 0.72492765, 0.82597206, 0.84711212, 0.86825217
                                ])}

```

```
In [100]: from sklearn.svm import LinearSVC #Linear Support Vector Classification
svc_pipeline = Pipeline([
    ("Preprocess", encoding),
    ("clf", LinearSVC())
])
```

```
In [101]: svc_results = cross_validate(svc_pipeline, df_prediction, y.T[0], cv=10, return_train_score=False)
svc_results
```

```

C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)
C:\Users\Kevin\Anaconda3\lib\site-packages\sklearn\svm\_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the
number of iterations.
    "the number of iterations.", ConvergenceWarning)

```

```

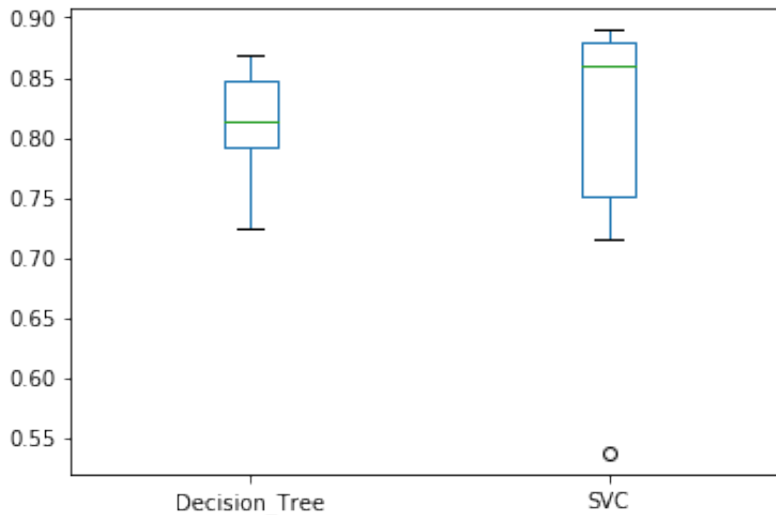
Out[101]: {'fit_time': array([107.81776857, 115.66807532, 114.98750234, 97.
01145935,
        107.62713265, 130.55823159, 131.8712759 , 111.71422672,
        113.16571689, 117.74833632]),
'score_time': array([0.09497809, 0.08497524, 0.11496496, 0.127955
91, 0.0779736 ,
        0.13595486, 0.10496736, 0.14195538, 0.1109736 , 0.09696937
]),
'test_score': array([0.88236034, 0.86524912, 0.85468042, 0.852793
16, 0.88210871,
        0.71489683, 0.71788096, 0.53705801, 0.87026551, 0.88951806
])}

```

```
In [102]: comparison = pd.DataFrame({ 'Decision_Tree': results['test_score'],
    'SVC': svc_results['test_score'] })
```

```
In [103]: comparison.plot.box()
```

```
Out[103]: <matplotlib.axes._subplots.AxesSubplot at 0x1abcfa2c940>
```



A comparison between SVC and DecisionTree methods : Now we do a cross validation. We separate in cv (groups) the dataframe, using K-fold method. Then cross_validate do a classification for every group using the algorithms. In this case, we decide to compare the decision tree with the SVC model. As we can see above, the decision tree algorithm works well and better than SVC. We can claim that because the “moustaches” are really separated than the decision tree’s ones.

Parameters Setting of the DecisionTree

Because the decision tree has a better prediction, we want to understand what is the best parameters setting in order to improve his effectiveness. In particular, we change **max_depth** (The maximum depth of the tree) and **min_sample_split** parameters (concerns the minimum number of samples required to split an internal node). We decide so because they are the main values that can avoid overfitting problems. In the end we save the best estimator, which may be used for further analysis.

```
In [104]: parameters = { 'decision_tree__max_depth': [2, 3, 4, 10, 15], 'decision_tree__min_samples_split': [0.02, 0.05, 0.1, 0.2] }
```

```
In [105]: from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import StratifiedKFold
```

```
In [106]: gs = GridSearchCV(pipeline, parameters, cv=StratifiedKFold(5))
```



```
In [107]: gs.fit(df_prediction, y)
```

```
Out[107]: GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
                        error_score=nan,
                        estimator=Pipeline(memory=None,
                                           steps=[('preprocess',
                                                  DataFrameMapper(default=Fa
lse,
                                                                df_out=False,
se,
                                                                features=[
(['category',
'main_category'],
OneHotEncoder(categories='auto',
drop=None,
dtype=<class 'numpy.float64'>,
handle_unknown='ignore',
sparse=True)),
(['Launch_Year...
                                                                min
_samples_leaf=1,
                                                                min
_samples_split=2,
                                                                min
_weight_fraction_leaf=0.0,
                                                                pre
sort='deprecated',
                                                                ran
dom_state=None,
                                                                spl
itter='best'))],
                        verbose=False),
                        iid='deprecated', n_jobs=None,
                        param_grid={'decision_tree__max_depth': [2, 3, 4, 10,
15],
                                   'decision_tree__min_samples_split': [0.02
, 0.05, 0.1,
                                                                0.2]
},
                        pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                        scoring=None, verbose=0)
```

```
In [108]: pd.DataFrame(gs.cv_results_).sort_values(by="rank_test_score").head(5)
```

Out[108]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_decision_tree__max
4	0.979884	0.138412	0.127361	0.010361	
5	0.900511	0.116596	0.131758	0.008232	
6	1.211011	0.128124	0.200336	0.062828	
0	1.095046	0.166224	0.197942	0.032218	
1	0.943702	0.287343	0.133751	0.028293	

```
In [109]: gs.best_params_
```

```
Out[109]: {'decision_tree__max_depth': 3, 'decision_tree__min_samples_split': 0.02}
```

```
In [110]: best_tree = gs.best_estimator_
```

Confusion Matrix

A Confusion Matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. The prediction isn't perfect, in particular there are too many errors in the 0 prediction. They are often predicted as 1.

A confusion matrix is a table that is often used to describe the performance of a classification model (or classifier) on a set of test data for which the true values are known. In other words, it is a performance measurement for machine learning classification problem where output can be two or more classes (3, in our case). Let's consider an example with binary classifier: the table will have 4 different combinations of predicted and actual values.

- True Positive: You predicted positive and it's true.
- True Negative: You predicted negative and it's true.
- False Positive: You predicted positive and it's false.
- False Negative: You predicted negative and it's false.

```
In [111]: import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [112]: from sklearn.metrics import confusion_matrix
X_train, X_test, y_train, y_test = train_test_split(df_prediction,
y, stratify=y)
best_tree.fit(X_train, y_train)
```

```
Out[112]: Pipeline(memory=None,
                    steps=[('preprocess',
                            DataFrameMapper(default=False, df_out=False,
                                              features=[('category', 'main_cat
egory'],
                                              OneHotEncoder(categori
es='auto',
                                                         drop=Non
e,
                                                         dtype=<c
lass 'numpy.float64'>,
                                                         handle_u
nknown='ignore',
                                                         sparse=T
rue)),
                            ('Launch_Year',
                             OrdinalEncoder(categor
ies=[1970,
2009,
2010,
2011,
2012,
2013,
2014,
2015,
2016,
2017,
2018]],
                                                         dtype=<
class 'n...
                                input_df=False, sparse=False)),
                            ('decision_tree',
                             DecisionTreeClassifier(ccp_alpha=0.0, class_weigh
t=None,
                                                         criterion='gini', max_dept
h=3,
                                                         max_features=None, max_lea
f_nodes=None,
```

```

        min_impurity_decrease=0.0,
        min_impurity_split=None,
        min_samples_leaf=1,
        min_samples_split=0.02,
        min_weight_fraction_leaf=0

        .0,

        om_state=None,

        verbose=False)

        min_impurity_decrease=0.0,
        min_impurity_split=None,
        min_samples_leaf=1,
        min_samples_split=0.02,
        min_weight_fraction_leaf=0

        presort='deprecated', rand

        splitter='best'))],

```

```

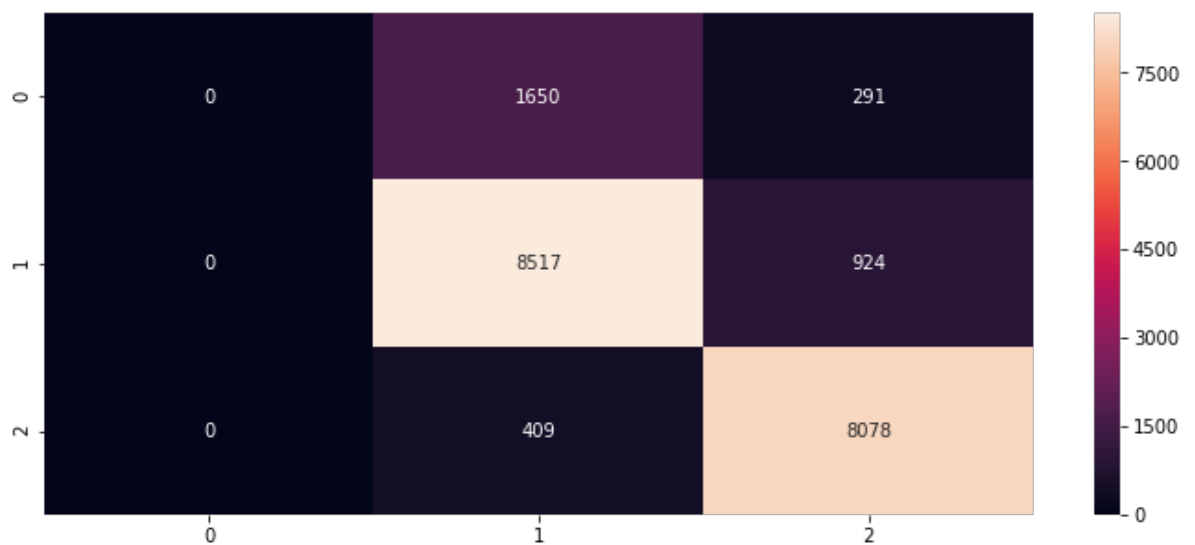
In [113]: plt.figure(figsize = (12, 5))
          cm = confusion_matrix(y_test, best_tree.predict(X_test))
          sns.heatmap(cm, annot=True, fmt="d")

```

```

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0xlabca826a20>

```



Other Algorithms : AdaBoostClassifier

AdaBoost is a machine learning meta-algorithm. It uses 'weak learners' which are combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favour of those instances misclassified by previous classifiers.

The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

Every learning algorithm tends to suit some problem types better than others and typically has many different parameters and configurations to adjust before it achieves optimal performance on a dataset. AdaBoost is often referred to as the best out-of-the-box classifier. When used with decision tree learning (decision trees as weak learners), information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

We combine AdaBoost and GridSearch to adapt and set the parameters for our analysis.

```
In [114]: from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
          from numpy import ravel
          from tqdm import tqdm
```

```
In [115]: abc_pipeline = Pipeline([
          ( "preprocess", encoding),
          ( "ada_classifier", AdaBoostClassifier(base_estimator = DecisionTreeClassifier(max_depth = 10)))
          ])
```

```
In [116]: param_grid_ADA = {"ada_classifier__n_estimators": [50, 75, 100, 150]}
```

```
In [117]: gs_ADA = GridSearchCV(abc_pipeline, param_grid_ADA , cv=StratifiedKFold(5))
```

```
In [118]: gs_ADA.fit(df_prediction, ravel(y))
```

```

Out[118]: GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
                        error_score=nan,
                        estimator=Pipeline(memory=None,
                                           steps=[('preprocess',
                                                  DataFrameMapper(default=False,
                                                                     df_out=False,
                                                                     features=[
('category',
'main_category'],
OneHotEncoder(categories='auto',
drop=None,
dtype=<class 'numpy.float64'>,
handle_unknown='ignore',
sparse=True)),
(['Launch_Year...
min_samples_split=2,
min_weight_fraction_leaf=0.0,
presort='deprecated',
random_state=None,
splitter='best'),
                                learnin
g_rate=1.0,
                                n_estim
ators=50,
                                random_
state=None)]),
                        verbose=False),
                        iid='deprecated', n_jobs=None,
                        param_grid={'ada_classifier__n_estimators': [50, 7
5, 100, 150]},
                        pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                        scoring=None, verbose=0)

```

```

In [119]: ada_df = pd.DataFrame(gs_ADA.cv_results_).sort_values(by="rank_test_score").head(5)

```

In [120]: ada_df

Out[120]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_ada_classifier__r
1	86.037386	6.362836	1.347968	0.200816	
2	113.196303	13.392369	1.380953	0.416204	
0	56.631328	2.059299	0.819346	0.036034	
3	171.407693	7.965838	2.287267	0.343134	

In [121]: best_ADA = gs_ADA.best_estimator_
best_ADA

Out[121]: Pipeline(memory=None,
 steps=[('preprocess',
 DataFrameMapper(default=False, df_out=False,
 features=[('category', 'main_cat
egory'],
 OneHotEncoder(categori
es='auto',
 drop=Non
e,
 dtype=<c
lass 'numpy.float64'>,
 handle_u
nknown='ignore',
 sparse=T
rue)),
 ('Launch_Year',
 OrdinalEncoder(categor
ies=[1970,
2009,
2010,
2011,
2012,
2013,
2014,
2015,
2016,

```

2017,

2018]],

dtype=<
class 'n...
base_estimator=DecisionTreeCla
ssifier(ccp_alpha=0.0,

class_weight=None,

criterion='gini',

max_depth=10,

max_features=None,

max_leaf_nodes=None,

min_impurity_decrease=0.0,

min_impurity_split=None,

min_samples_leaf=1,

min_samples_split=2,

min_weight_fraction_leaf=0.0,

presort='deprecated',

random_state=None,

splitter='best'),

learning_rate=1.0, n_estimator
s=75,

random_state=None)),

verbose=False)

```

```
In [122]: gs_ADA.best_params_
```

```
Out[122]: {'ada_classifier__n_estimators': 75}
```

```
In [123]: x_tr , x_tst, y_tr , y_tst = train_test_split(df_prediction, y)
```

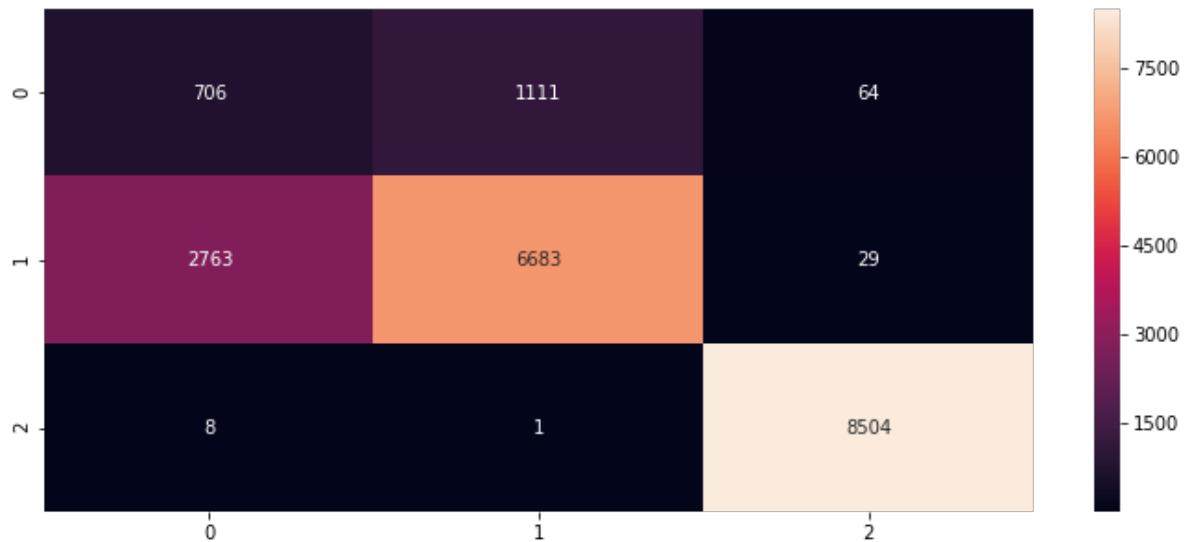
```
In [124]: best_ADA.fit(df_prediction, ravel(y))
y_pred_ADA = best_ADA.predict(df_prediction)
accuracy_score(y_pred_ADA, y)
```

```
Out[124]: 0.8155543811968393
```



```
In [125]: plt.figure(figsize = (12, 5))  
best_ADA.fit(x_tr,ravel(y_tr))  
cm = confusion_matrix(y_tst, best_ADA.predict(x_tst))  
sns.heatmap(cm, annot=True, fmt="d")
```

Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x1abca8f48d0>



The confusion matrix delineates that AdaBoost can recognize more 0 status than the decision tree, but decision tree is a little more precise, thanks to its accuracy score.

Other Algorithms: Random Forest

Random Forest is an ensemble learning method for classification. It is similar to AdaBoost but it uses elementary decision trees as 'weak learners'. It prevents the decision trees' habit to overfit to their training set.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is: "A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models".

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

- There needs to be some actual signal in our features so that models built using those features do better than random guessing.
- The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

We combine Random Forest and GridSarch to adapt and set the parameters for our analysis.

```
In [126]: RF_pipeline = Pipeline([
            ("preprocess", encoding),
            ("RF_classifier", RandomForestClassifier())
        ])
```

```
In [127]: param_grid_RF = {"RF_classifier__n_estimators": [50, 100, 150],
                           "RF_classifier__criterion": ["gini", "entropy"]
                           }
```

```
In [128]: gs_RF = GridSearchCV(RF_pipeline, param_grid_RF)
```

```
In [129]: gs_RF.fit(df_prediction, ravel(y))
```

```
Out[129]: GridSearchCV(cv=None, error_score=nan,
                      estimator=Pipeline(memory=None,
                      steps=[('preprocess',
                              DataFrameMapper(default=Fa
lse,
                      df_out=False,
                      features=[
```

```

([ 'category',
'main_category'],
OneHotEncoder(categories='auto',
drop=None,
dtype=<class 'numpy.float64'>,
handle_unknown='ignore',
sparse=True)),
([ 'Launch_Year'],
OrdinalEncoder(categories=[[1970,
2009,
2010,
2011,
2012...
_weight_fraction_leaf=0.0,
stimators=100,
obs=None,
_score=False,
dom_state=None,
bose=0,
m_start=False)]),
(verbose=False),
iid='deprecated', n_jobs=None,
param_grid={'RF_classifier__criterion': ['gini', '
entropy'],
'RF_classifier__n_estimators': [50, 10
0, 150]},
pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
scoring=None, verbose=0)

```

```

In [130]: RF_df = pd.DataFrame(gs_RF.cv_results_).sort_values(by="rank_test_s
core").head(5)

```

In [131]: RF_df

Out[131]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_RF_classifier_ci
5	50.933297	0.902323	1.305180	0.071004	(
0	15.072041	0.567623	0.459853	0.101626	
4	33.204750	2.076615	0.847330	0.160642	(
3	17.489527	0.238007	0.543825	0.028239	(
1	25.598292	3.606537	0.866124	0.133433	

In [132]: best_RF = gs_RF.best_estimator_
best_RF

Out[132]: Pipeline(memory=None,
 steps=[('preprocess',
 DataFrameMapper(default=False, df_out=False,
 features=[('category', 'main_cat
egory'],
 OneHotEncoder(categori
es='auto',
 drop=Non
e,
 dtype=<c
lass 'numpy.float64'>,
 handle_u
nknown='ignore',
 sparse=T
rue)),
 ('Launch_Year',
 OrdinalEncoder(categor
ies=[1970,
2009,
2010,
2011,
2012,
2013,
2014,
2015,

```

2016,
2017,
2018]],
dtype=<
class 'n...
RandomForestClassifier(bootstrap=True, ccp_alpha=
0.0,
class_weight=None, criteri
on='entropy',
max_depth=None, max_featur
es='auto',
max_leaf_nodes=None, max_s
amples=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1, min_sa
mples_split=2,
min_weight_fraction_leaf=0
.0,
n_estimators=150, n_jobs=N
one,
oob_score=False, random_st
ate=None,
verbose=0, warm_start=Fals
e)],
verbose=False)

```

```
In [133]: gs_RF.best_params_
```

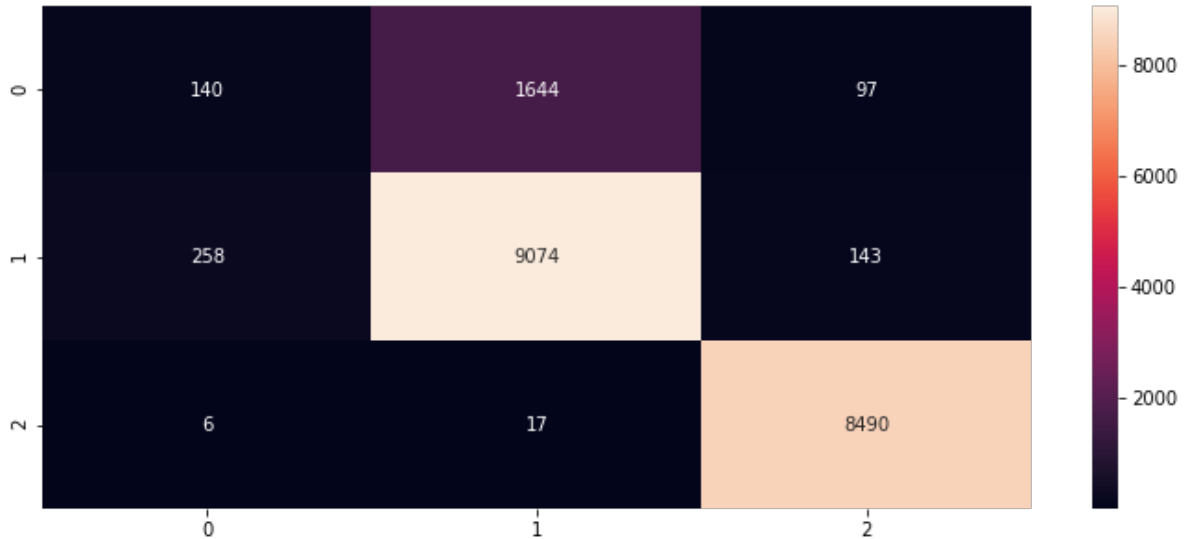
```
Out[133]: {'RF_classifier__criterion': 'entropy',
'RF_classifier__n_estimators': 150}
```

```
In [134]: best_RF.fit(df_prediction, ravel(y))
y_pred_RF = best_RF.predict(df_prediction)
accuracy_score(y_pred_RF, y)
```

```
Out[134]: 0.997798077407016
```

```
In [135]: plt.figure(figsize = (12, 5))
best_RF.fit(x_tr, ravel(y_tr))
cm = confusion_matrix(y_tst, best_RF.predict(x_tst))
sns.heatmap(cm, annot=True, fmt="d")
```

Out[135]: <matplotlib.axes._subplots.AxesSubplot at 0x1abca8d82e8>



Cross Validation: AdaBoosts and RandomForest

Our aim in this notebook is recognize the best classificatory for our dataframe. In order to do that we use a cross validation comparison. We are going to do the same we did for the decision tree algorithm at the beginning.

```
In [136]: results_ADA = cross_validate(best_ADA, df_prediction, ravel(y), cv=
10, return_train_score=False)
results_ADA
```

Out[136]: {'fit_time': array([103.6788919 , 96.37518406, 100.19326901, 88.50462341, 101.59822869, 102.27596617, 101.07885432, 103.32860613, 98.92904186, 105.20483994]), 'score_time': array([0.60579944, 0.25591612, 0.42286134, 0.67779112, 0.72876763, 0.64679527, 0.60280013, 0.62080574, 0.61680269, 0.44386315]), 'test_score': array([0.78271263, 0.79227479, 0.71930045, 0.76673377, 0.84197282, 0.71451938, 0.74329936, 0.74506103, 0.80948786, 0.81754121])}

```
In [137]: results_RF = cross_validate(best_RF, df_prediction, ravel(y), cv=10
, return_train_score=False)
results_RF
```

```
Out[137]: {'fit_time': array([51.57146406, 51.81435847, 58.00717163, 39.6855
4807, 43.03419638,
56.68082833, 58.69518209, 58.47254539, 59.12526846, 55.875
08488]),
'score_time': array([0.32089901, 0.83673477, 0.37787747, 0.657791
85, 0.57881546,
0.61481857, 1.00868082, 1.08565164, 1.082654 , 0.34989119
]),
'test_score': array([0.87506291, 0.87795672, 0.87518873, 0.889028
69, 0.89028686,
0.71942627, 0.72077514, 0.88800805, 0.89102806, 0.80999119
])}
```

Final Comparison between all the Algorithms

```
In [138]: final_comparison = pd.DataFrame({ 'Decision_Tree': results['test_sc
ore'], 'SVC': svc_results['test_score'],
'AdaBoostQualifier': results_ADA[ '
test_score'], 'Random_Forest': results_RF['test_score'] })
```

```
In [139]: final_comparison
```

```
Out[139]:
```

	Decision_Tree	SVC	AdaBoostQualifier	Random_Forest
0	0.792149	0.882360	0.782713	0.875063
1	0.794162	0.865249	0.792275	0.877957
2	0.800453	0.854680	0.719300	0.875189
3	0.847635	0.852793	0.766734	0.889029
4	0.856064	0.882109	0.841973	0.890287
5	0.748742	0.714897	0.714519	0.719426
6	0.724928	0.717881	0.743299	0.720775
7	0.825972	0.537058	0.745061	0.888008
8	0.847112	0.870266	0.809488	0.891028
9	0.868252	0.889518	0.817541	0.809991

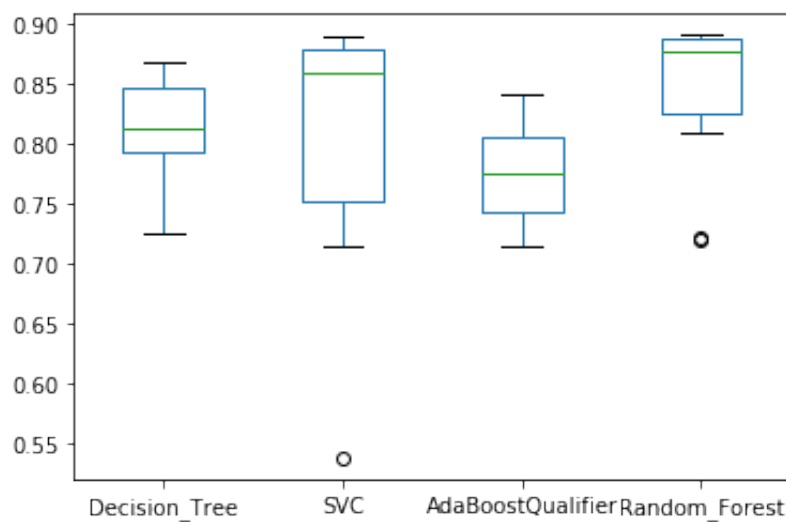
```
In [140]: final_comparison.describe()
```

```
Out[140]:
```

	Decision_Tree	SVC	AdaBoostQualifier	Random_Forest
count	10.000000	10.000000	10.000000	10.000000
mean	0.810547	0.806681	0.773290	0.843675
std	0.047389	0.115079	0.042873	0.069336
min	0.724928	0.537058	0.714519	0.719426
25%	0.792652	0.751609	0.743740	0.826259
50%	0.813213	0.859965	0.774723	0.876573
75%	0.847504	0.879148	0.805185	0.888774
max	0.868252	0.889518	0.841973	0.891028

```
In [141]: final_comparison.plot.box()
```

```
Out[141]: <matplotlib.axes._subplots.AxesSubplot at 0xlabce04f668>
```



Conclusion of Notebook 05.1:

This is an important graphic. In the first part of notebook 4, we should decide to use an SVC classifier. But thanks to the cross validation, we can argue several points:

- The usage of SVC, for our dataframe, led us to a poor accuracy, in fact its variance is worse than Decision_Tree and AdaBoost. Moreover, an ensemble method like AdaBoost will get worse our prediction.
- **The best algorithm to predict if a future project could be successful may be RandomForest.** Its mean is much better than the others' one. Furthermore, its variance is really good, although the presence of an outlier.

In conclusion, we claim that the RandomForest fitted model should be use to predict if a project will be successful or not.

Our Kickstarter Project

Now we have a model through which we can predict if a project may be successful or not.

```
In [165]: df_ks = pd.read_csv('nostroprogetto.csv')
df_ks.head()
```

Out[165]:

	Unnamed: 0	category	main_category	backers	usd_pledged_real	usd_goal_real	Launch_
0	0	Product Design	Design	21	2000	5000	;

```
In [166]: df_ks = df_ks.drop(columns = 'Unnamed: 0')
```

```
In [167]: df_ks
```

Out[167]:

	category	main_category	backers	usd_pledged_real	usd_goal_real	Launch_Year	Launc
0	Product Design	Design	21	2000	5000	2018	

```
In [185]: y_pred_RF = best_RF.predict(df_ks)
```

```
In [186]: y_pred_RF
```

```
Out[186]: array([1.])
```

Conclusion of Notebook 05.2:

This example argues that the project we want to create may be successful. The model we use to predict can be applied to other projects in order to evaluate them.

```
In [ ]:
```