BAHÇEŞEHİR UNIVERSITY

Faculty of Engineering and Natural Sciences

SEN4018 Data Science with Python Project

“Kickstarter Projects Dataset”

Students’ Name & Surname, ID: Yousef Elbayoumi, 1722326

Bora Doruk Bilgin, 1729464

Khaled Al Hariri, 1801284

Elnur Seyidov, 1805676

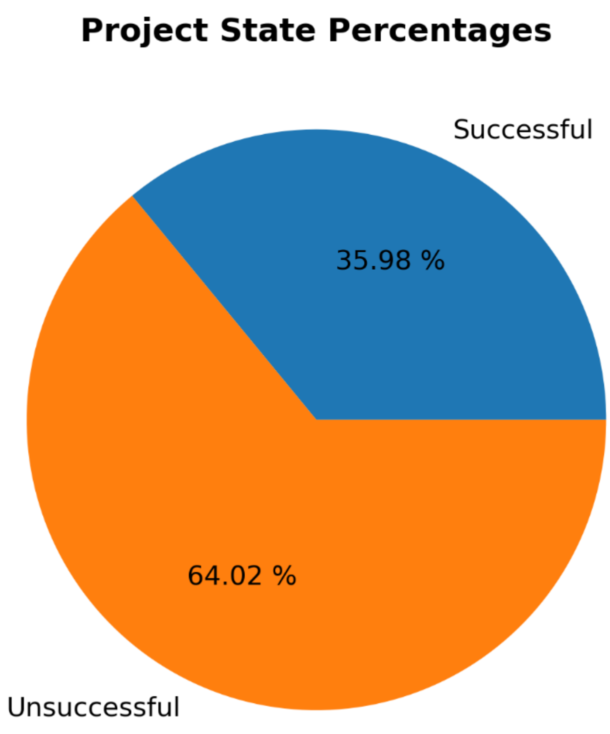

Table of Content

LIST OF TABLES

LIST OF FIGURES

1. Aim of the project

Our purpose of this project is to analyze the effect of attributes in this dataset on the projects, and their success. Another goal is to make comparisons between those attributes and show how one affects another. For instance, how successful are projects that have music as their main category and if those projects that have a big goal fails most of the time or not. We plan to display the correlations for the dataset such as the correlations between successful projects and people's pledged money and the correlations between the state of the project and its country of origin. We also want to see if these correlations contribute to the project being failed, cancelled, or suspended.

2. Feature names

Column	***Description***
ID	*Project identifier*
Name	*Name of the project*
Category	*Type of the project*
Main_category	*Main focus of the project*
Currency	*Currency in which the project is managed*
Deadline	*Date when the project needs to be completed*
Goal	*Amount of money the project wants to achieve*
Launched	*Date when the project was launched*
Pledged	*Amount of money that the people pledged*
State	*Final state of the project*
Backers	*Number of people that support the project*
Country	*Country where the project started*
USD pledged	*Conversion in US dollars of the pledged column (conversion done by kickstarter)*
USD_pledged_real	*Conversion in US dollars of the pledged column (converted more accurate from Fixer.io API)*
USD_goal_real	*The amount of money the project wants to achieve (converted more accurate from Fixer.io API)*

Table 2.1. Explains the feature names.

3. Dataset used

https://www.kaggle.com/kemical/kickstarter-projects/version/3

This dataset was taken from Kaggle.com. Kickstarter allows businesses to share their products descriptions and ask people to pledge money to start full-scaled productions. This dataset shows many attributes of Kickstarter projects, such as their categories, currencies, dates (launched time, deadline), state (successful, failed, etc.), goal, etc. This dataset shows the result of 321,616 unique projects in year 2018.

4. Data visualization (After Preprocessing – No Encoding)

Heatmap:

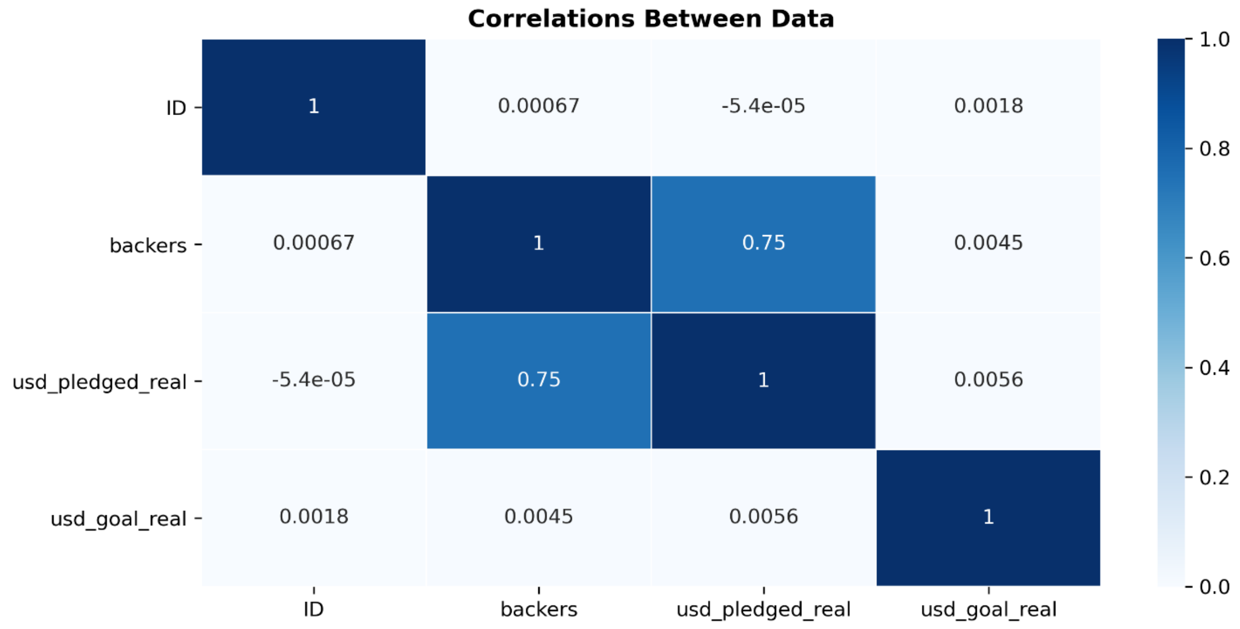

Figure 4.1. Data visualization - Heatmap.

Bar Chart:

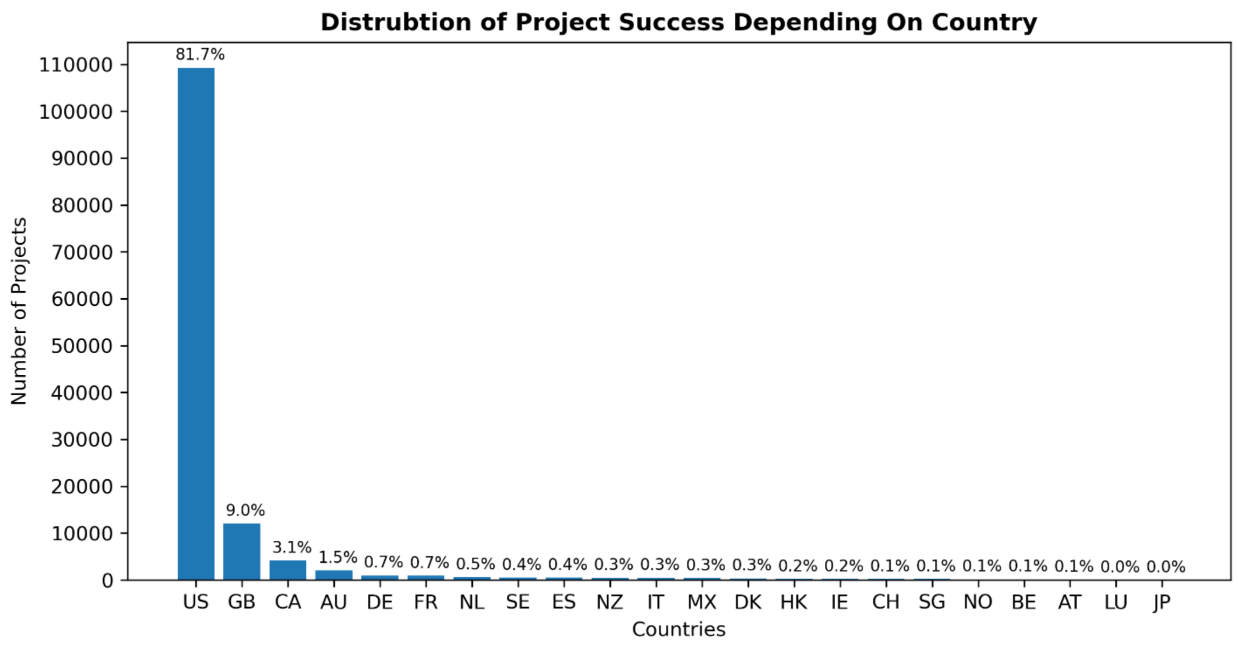

Figure 4.2. Data visualization – Bar Chart.

Stacked Bar Charts:

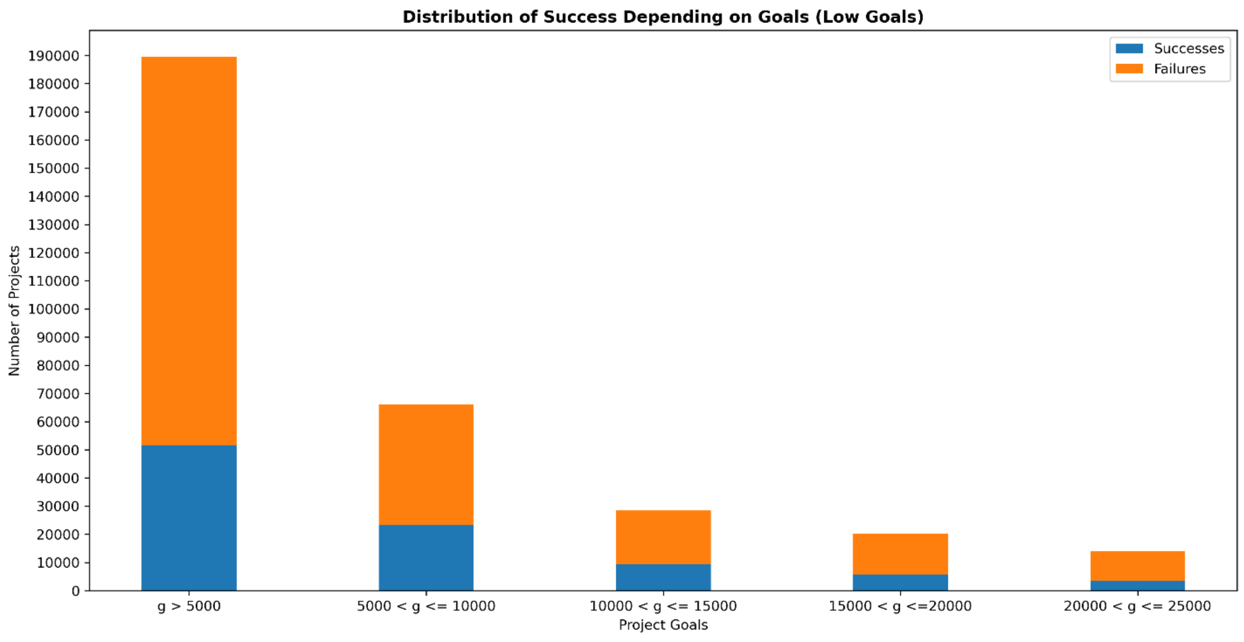

Figure 4.3. Data visualization – Stacked Bar Charts.

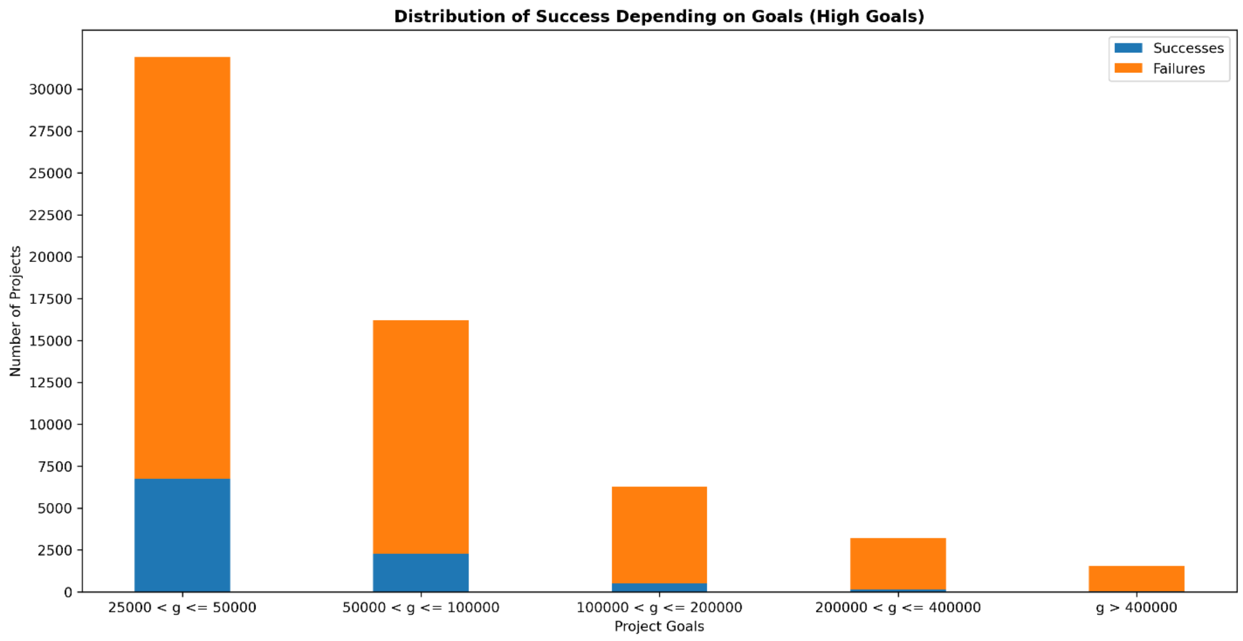

Figure 4.4. Data visualization – Stacked Bar Charts 2.

Pie Charts:

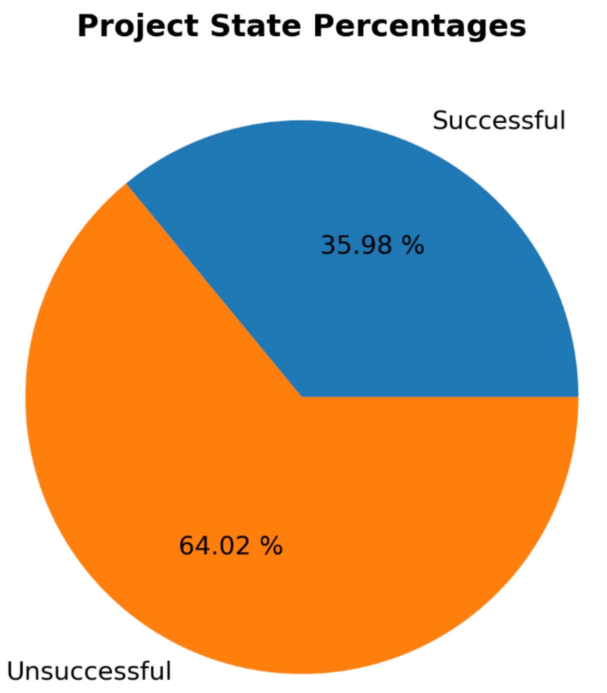

Figure 4.5. Data visualization – Pie Charts.

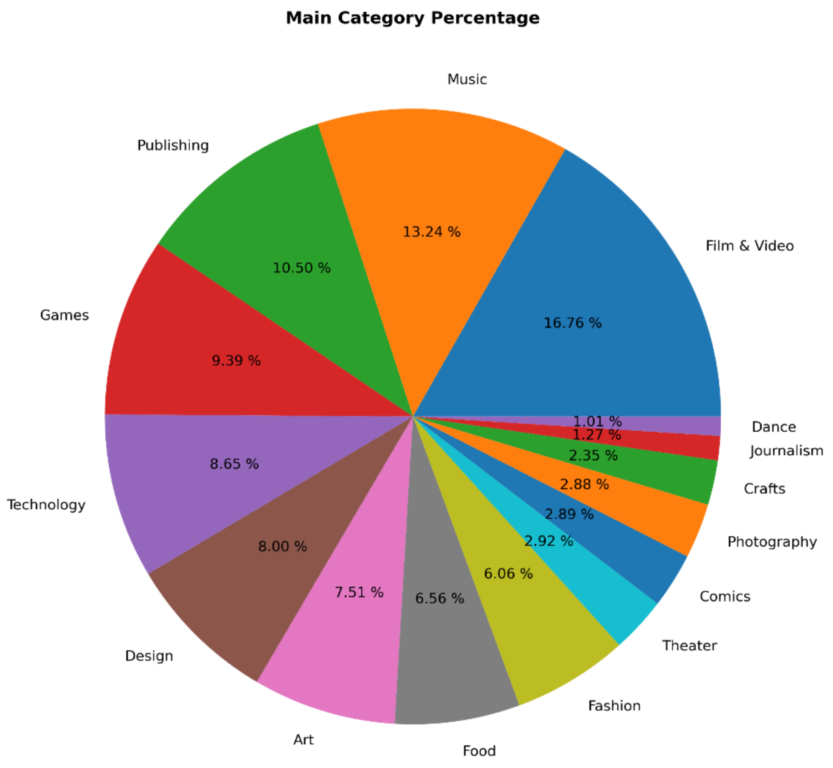

Figure 4.6. Data visualization – Pie Charts 2.

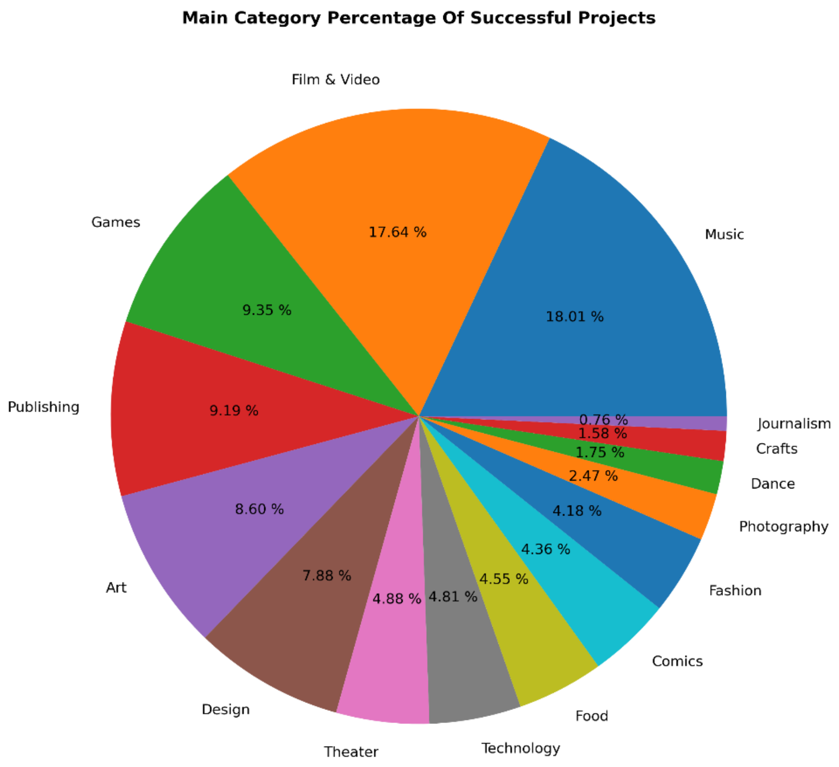

Figure 4.7. Data visualization – Pie Charts 3.

Categorical Plot:

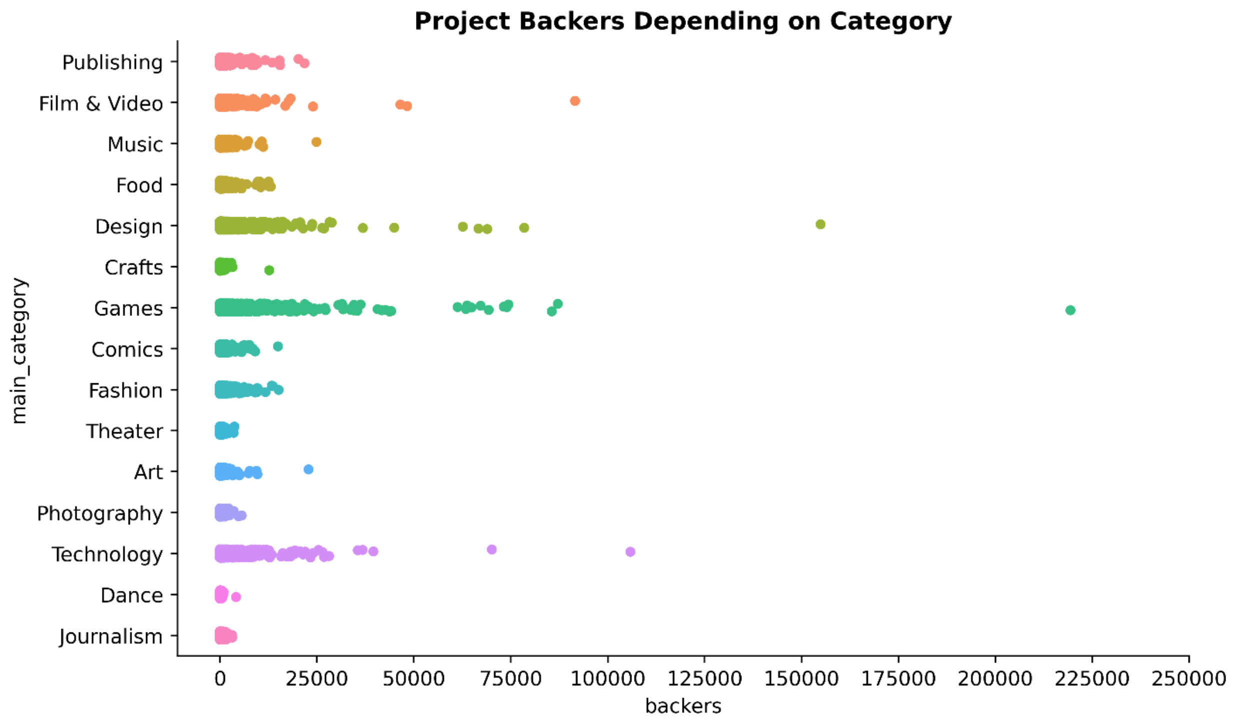

Figure 4.8. Data visualization – Categorical Plot.

5. Data preprocessing

5.1. Data formatting

The data was initially formatted in order to avoid any errors during the analysis. Some attributes were of not use for the analysis such as 'launched', 'deadline' and 'currency'. Dates will not be used which is why the first 2 will be dropped. For 'currency', only the 'usd_goal_real' and the 'usd_pledge_real' will be used to make all of the projects have the same currency. This is also why the goal, pledged and usd pledged columns will removed as well, since we want to only use one column for the pledge and one for the goal which are the real usd ones. The other attributes were all important for future analysis; therefore, it was decided to keep them.

However, the attribute 'state' interfered with the process of analysis due to the excess of values. Hence, it was decided to format them in a way that will give us results that are more precise in future operations. States were divided into 'successful' and 'unsuccessful'. The rows that had the ‘live’ state were all dropped since we can’t tell if they were successful or not.

5.2. Data cleaning

After checking all the null values in the dataset, it was found that only two attributes contained null values, which are 'USD pledged' and 'name'. It was later identified that all the null values in attribute 'USD pledged' were all associated with an unknown country 'N,0"'. Due to a lack of knowledge and practicality in these values in the dataset, it was concluded to remove them. The attribute 'name' contained four null values; therefore, it was decided to just rename them to 'unknown'.

5.3. Data sampling

Firstly, for repetition sampling, we picked samples randomly. So, while sampling we have enabled repetitions. Therefore, the data might be affected negatively since we could take the same rows again. Secondly, for non-repetition sampling, we picked different samples randomly and each sample is discarded after getting picked. This means that we have no repetition data in the samples.

5.4. Data encoding

The dataset had a lot of categorical columns that could be used for regression. We used one hot encoding to encode these columns. Firstly, ‘state’ column was encoded to binary values, 0 and 1 (1 meaning successful and 0 meaning unsuccessful). Secondly, we turned the ‘country’ column into a ‘continent’ column, decreasing the unique values from 22 to only 4. Then the ‘continent’ column was encoded using one hot encoding and turned into 4 columns. The extra column was removed leaving only 3 columns.

6. Data transformation

6.1. Aggregation

2 new columns were created by doing calculations between the ‘usd_pledged_real’ column and the ‘usd_goal_real’ column. The first column is ‘success_amount’, which specifies by how much the project succeeded, if it did. Calculated by dividing the ‘usd_pledged_real’ column by the ‘usd_goal_real’ column. If the result is more than 1 then the project did better than expected. The second column is ‘pledge_goal_difference’, which is calculated by getting the difference between ‘usd_pleded_real’ and ‘usd_goal_real’.

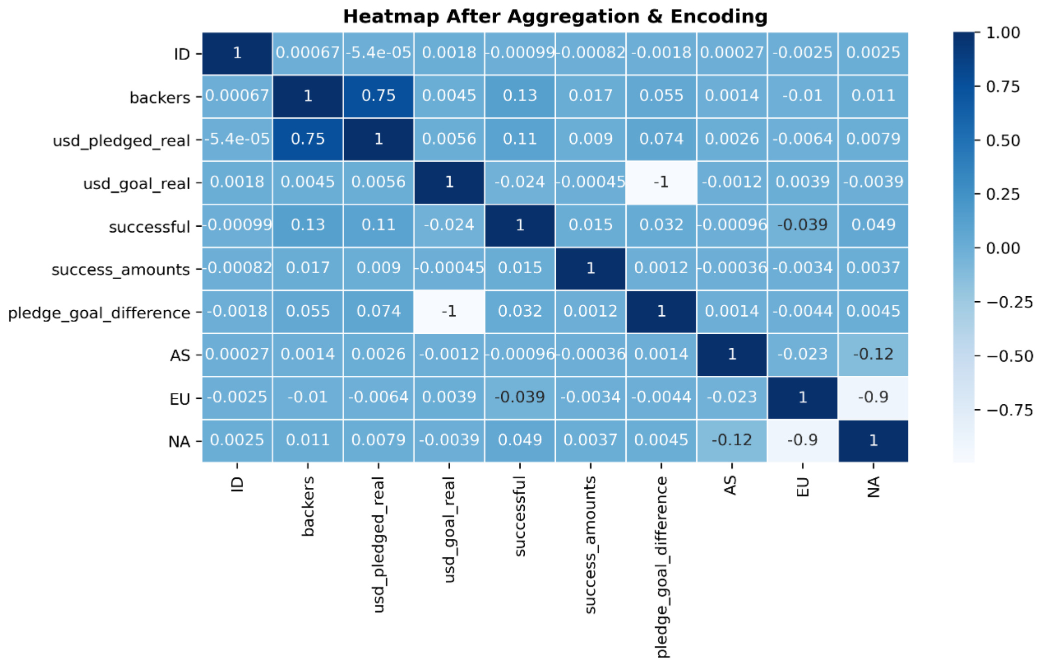

Figure 6.1.1. Heatmap after Aggregation and Encoding.

6.2. Scaling

The features were so varied and have different types of categories in our dataset, this is not an issue normally but some of the features had no correlation between each other, this was the problem and to solve it we did standardization to the feature’s values (numerical ones) with sklearn

StandardScaler, the mean of the dataset became 0 and the standard deviation became 1, the standardization formula is:
Z = (x – μ)/ σ
μ = population main, and σ = standard deviation.

6.3. Data summaries

After aggregation and encoding, we used 5 columns which contain numerical values to create data summaries to compare the actual data population and a sample we randomly selection with no repetition. You can see that they are analyzed and shown in the tables below. Since our dataset contains more than 300.000 project results, we decided to take sample data as 30.000 project randomly and check their data summaries and point estimations and compare them with the population.

Quantitative Attributes:

1. backers
2. usd_pledged_real
3. usd_goal_real
4. success_amounts
5. pledge_goal_difference

Population's data summaries:

	1.	***2.***	***3.***	***4.***	***5.***
MIN	*0.00*	*0.00*	*0.01*	*0.00*	*-166361390.7*
1Q	*2.00*	*31.24*	*2000.00*	*0.00*	*-10000.00*
MEDIAN	*12.00*	*627.59*	*5500.00*	*0.14*	*-1973.00*
3Q	*57.00*	*4066.00*	*16000.00*	*1.07*	*200.00*
MAX	*219382.00*	*20338986.27*	*166361390.7*	*104277.89*	*19838986.27*

Table 6.3.1. Data summaries - Population.

Sample's data summaries:

	1.	***2.***	***3.***	***4.***	***5.***
MIN	*0.00*	*0.00*	*0.79*	*0.00*	*-100000000*
1Q	*2.00*	*33.23*	*2000.00*	*0.01*	*-10000.00*
MEDIAN	*12.00*	*625.63*	*5500.00*	*0.14*	*-1965.14*
3Q	*56.00*	*4010.00*	*15838.32*	*1.06*	*190.57*
MAX	*73986.00*	*13285226.36*	*100000000.00*	*55266.57*	*13235226.36*

Table 6.3.2. Data summaries - Sample.

Population's point estimations:

Variables	***Mean***	***Standard Deviation***
Backers	*106.98*	*914.52*
usd_pledged_real	*9145.24*	*91620.97*
usd_goal_real	*45737.70*	*1151682.00*
success_amounts	*3.27*	*269.16*
pledge_goal_difference	*-36592.46*	*1154806.00*

Table 6.3.3. Point estimations - Population.

Sample's point estimations:

Variables	***Mean***	***Standard Deviation***
Backers	*107.41*	*967.80*
usd_pledged_real	*9760.43*	*137135.70*
usd_goal_real	*42483.33*	*975658.40*
success_amounts	*4.25*	*335.11*
pledge_goal_difference	*-32722.91*	*984623.30*

Table 6.3.4. Point estimations - Sample.

7. Feature selection – extraction

The method used for feature selection was removing features which has low variance by using Variance Threshold from sklearn. The threshold was set to 0.3 and it only got encoded categorical features. Due to that, no features were removed during the feature selection – extraction process.

8. Dataset splitting

8.1. Cross Validation Technique

The data was first split into train and test sets as a cross validation process in order to start the modelling. We fit the data in the linear model first to get the error rates, then we standardized it and fitted it again and got both of the training and testing models. After that we fitted the data on a decision tree model and a random forest model. We got the msqe and rmse rates for both. This was on all of the population. After that we got used the sample we got from random with no repetition and repeated the previous models with trained and test sets. The models were close to the original ones with the whole population.

9. Modelling

For modeling part, we made several models for our population data, we could do also for sample data to check the similarity between them as we did in "Data summaries" section, but the result can be predicted that they will be so close to each other as their data summaries were almost the same. Therefore, we didn't include the sample modeling.

9.1. Model Training

First of all, we split our data to "training" and "test" sets, then linear regression model was applied, we used "backers" and "usd_pledged_real" columns because of their correlation, which was 0.75 and a high correlation would bring the best results in linear regression.

Linear regression with training set:

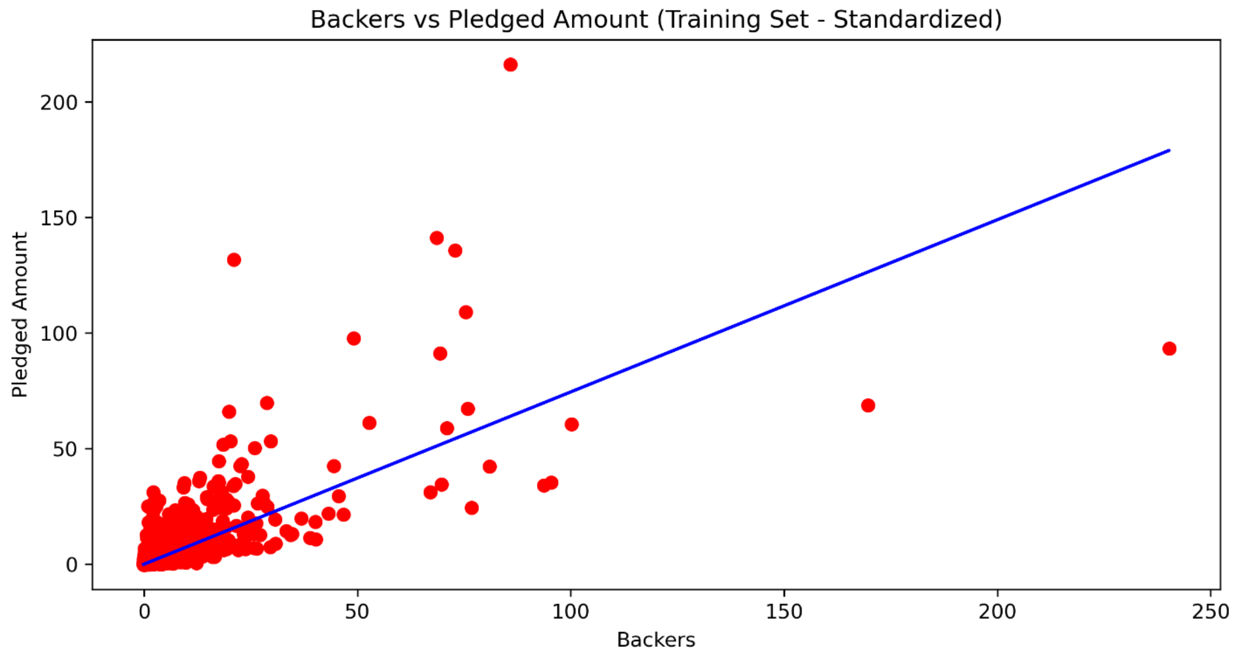

Figure 9.1.1. Linear regression – Training set.

Linear regression with testing set:

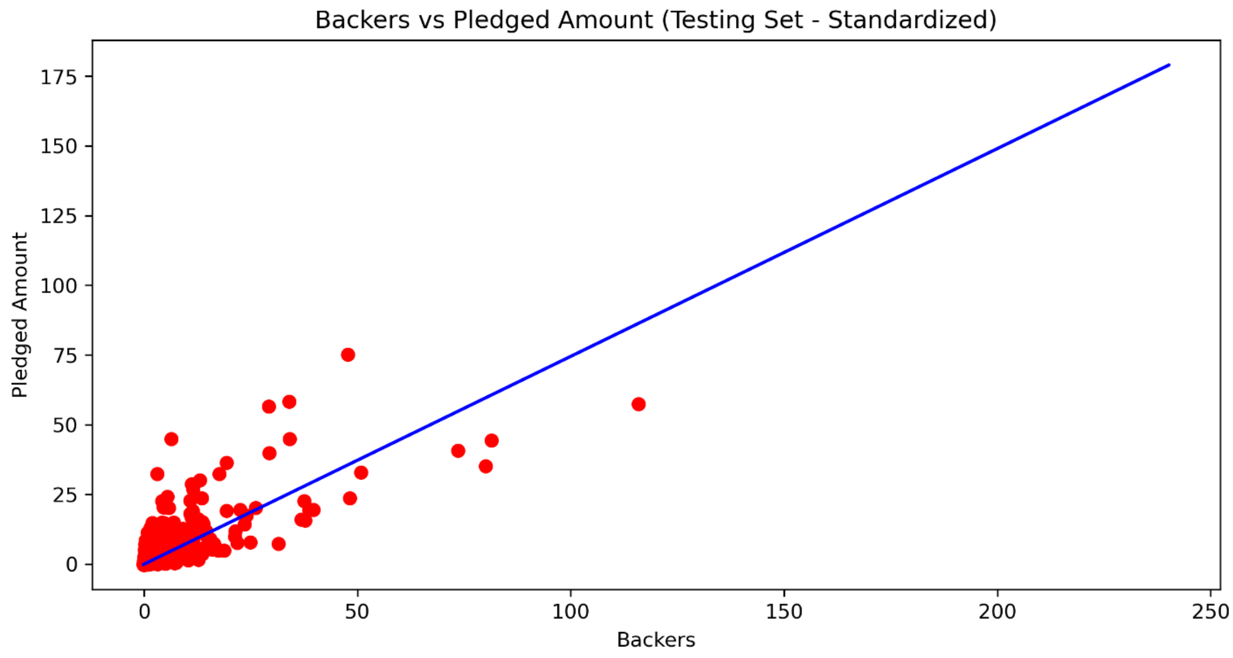

Figure 9.1.2. Linear regression – Testing set.

Also, we trained polynomial regression as well for more analyzing and understanding for the data. Moreover, we decided to use the second-degree model as it was fitting most accurately.

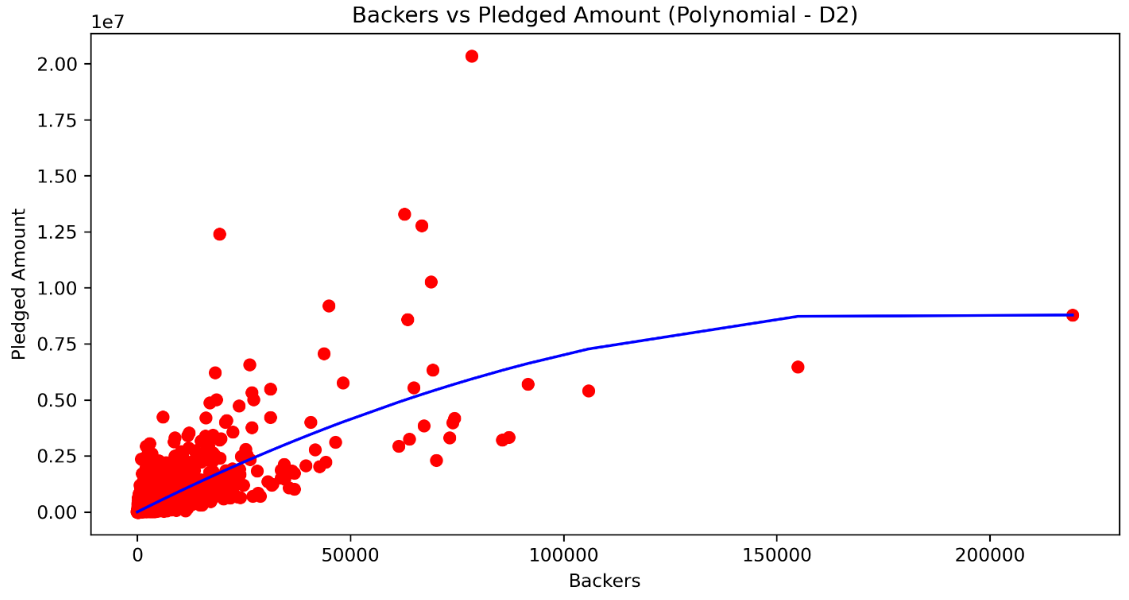

Figure 9.1.3. Polynomial model second degree.

Another model was trained which is Decision tree model, simply the goal here was to create a model that can predict the target variable value by learning the rules of the simple decision which was inferred from the training data.

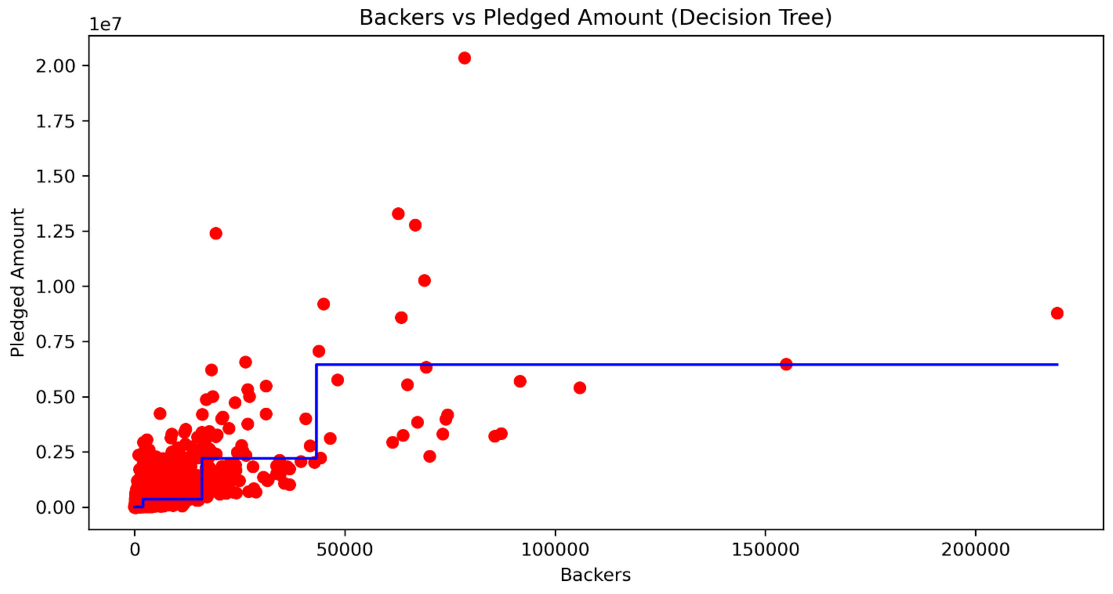

Figure 9.1.4. Decision Tree model.

Random forest was also trained and tested which gave accurate results. However, the model could be overfitting due to how accurate and precise it is.

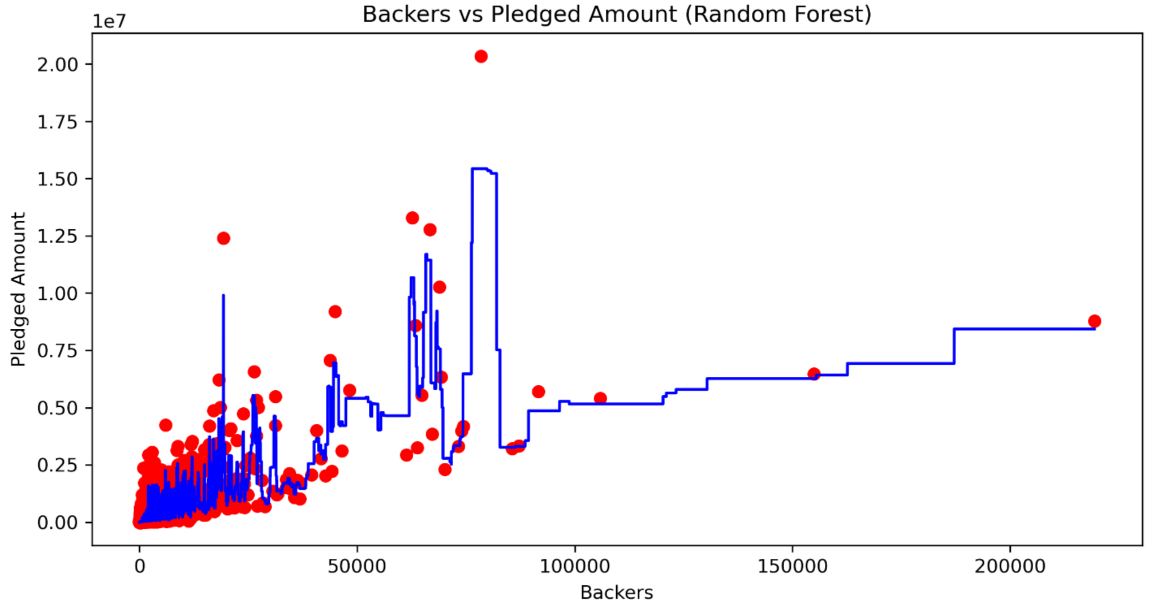

Figure 9.1.5. Random Forest model.

9.2. Model evaluation and testing

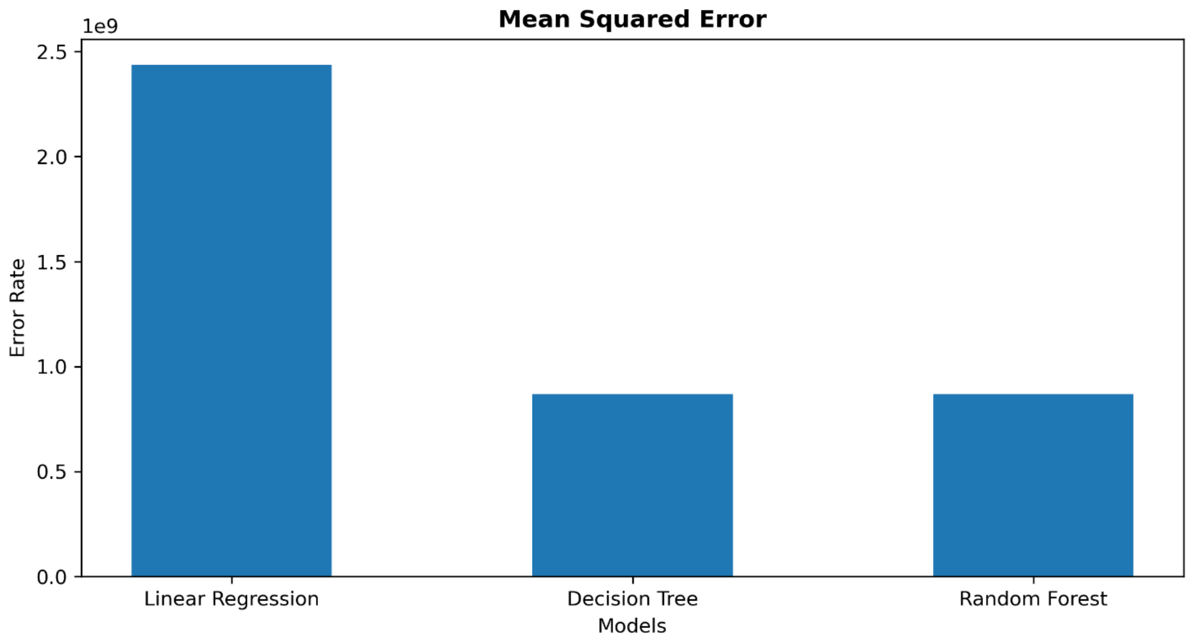

Figure 9.2.1. Mean Squared Error figure.

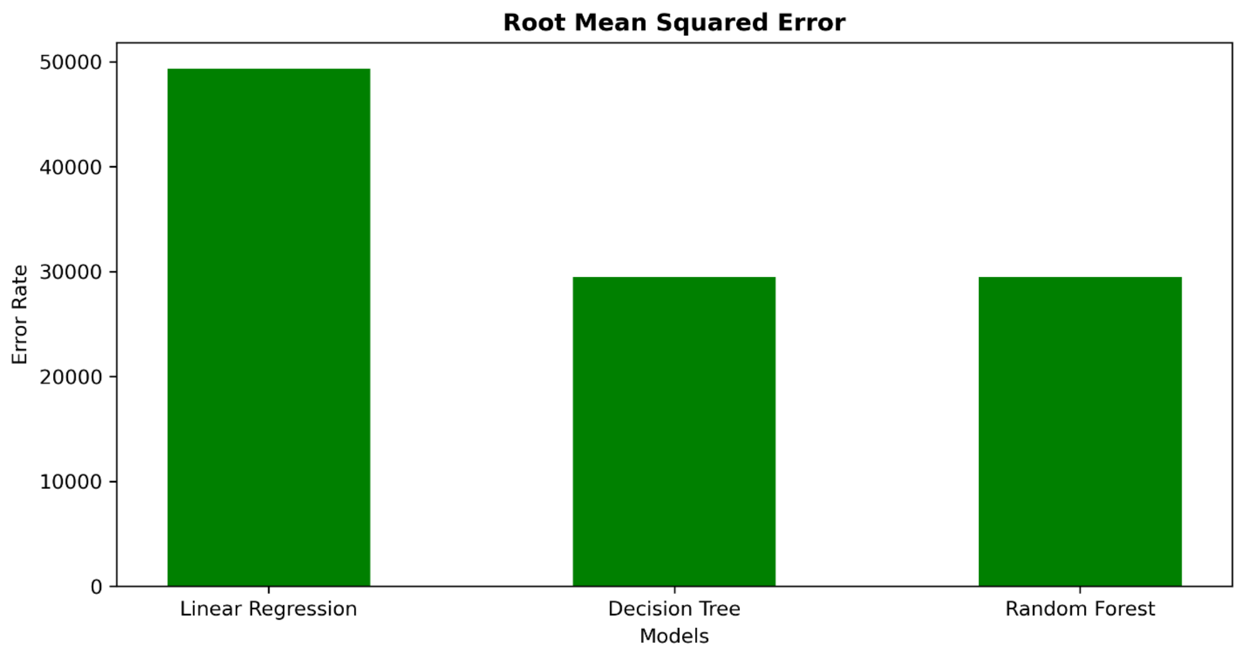

Figure 9.2.2. Root Mean Squared Error figure.

In the previous two figures we showed some error results for Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The MSE value shows how close is a fitted line to the data points, so it's basically measures the quality of an estimator, and RMSE is just the square root of these values.

10. Conclusion

In conclusion, many different analysis methods and tests has been applied such as Data visualization, Data preprocessing, Data transformation, correlation, modeling, etc. We were able to find clear answers for the questions that was asked previously in "Aim of the project" section. We updated our data, removed some columns, and added new columns. We took random sample of 30,000 projects and saw that it has close similarity in the data summaries to the population dataset. We showed the correlation between our data and we saw some have high correlation while others have low correlation. In the modeling section, we trained different models like linear regression, polynomial regression and decision tree model, and we saw the differences between them.

11. References

General references were from:
https://www.geeksforgeeks.org/

Data summaries:
https://www.dataanalytics.org.uk/data-analytics-knowledge-base-tips-tricks-r-excel/statistics-guide/data-summary/

Training and testing sets:
https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data
https://medium.com/@alexstrebeck/training-and-testing-machine-learning-models-e1f27dc9b3cb

Decision tree:

https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html#:~:text=The%20goal%20of%20using%20a,the%20root%20of%20the%20tree.

The recorded lectures were also a great help.

12. Appendix

```
#Importing Necessary Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sb
import random as r
from collections import Counter
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import VarianceThreshold
from random import choices



df = pd.read_csv('C:\\Users\\alhar\\.spyder-py3\\ks.csv') #Reading The Data

#----------PREPROCESSING----------

df['name'] = df['name'].replace(np.nan, 'unknown') #Change Null Values To
UNKNOWN

indexes_to_drop = list(df.loc[df['country'] == 'N,0"'].index) #Select Rows To Drop
columns_to_drop = ['currency', 'deadline', 'goal', 'launched', 'pledged', 'usd
pledged']

df = df.drop(indexes_to_drop) #Drop Undefined Rows
df = df.drop(columns_to_drop, axis = 1)

indexes_to_drop2 = list(df.loc[df['state'] == 'live'].index)
df = df.drop(indexes_to_drop2) #Drop Rows With Live State
```

```
df.loc[((df['state'] == 'suspended') | (df['state'] == 'canceled') | (df['state'] ==
'failed')), ['state']] = 'unsuccessful' #Formatting Data

df.reset_index(drop = True, inplace = True)

for a in list(df.columns.values):
    print(a)



def retriveSamples(a,df):
   return df.loc[df['ID'].isin(a)]

numberSamples = 30000

IDseries = df['ID']

# Sampling with repetition
sampleRep = retriveSamples(choices(IDseries, k=numberSamples),df)
print(sampleRep)

# Sampling without repetition
sampleNoRep = retriveSamples(r.sample(IDseries.tolist(), k=numberSamples),df)
print(sampleNoRep)

#----------VISUALIZATION----------

#Pie Chart
plt.figure(figsize = (10, 5), dpi= 300)

#Collect Count Of Each State
successful = df.loc[df['state'] == 'successful'].count()[0]
unsuccessful = df.loc[df['state'] == 'unsuccessful'].count()[0]

states = [successful, unsuccessful]
labels = ['Successful', 'Unsuccessful']
```

```
plt.pie(states, labels = labels, autopct = '%.2f %%')
plt.title('Project State Percentages', fontdict={'fontweight': 'bold'})
plt.show()

suc_df = df.loc[df['state'] == 'successful'] #Get Successfull Projects

alln = suc_df.count()[0] #Count Of All Successfull Projects

#Count The Countries And Sort Them
countries_sorted = [e for e, _ in Counter(suc_df['country']).most_common()]
countries_count = [e for _, e in Counter(suc_df['country']).most_common()]

#Bar Chart
plt.figure(figsize = (10, 5), dpi= 300)
bars = plt.bar(countries_sorted, countries_count)
plt.title('Distrubtion of Project Success Depending On Country',
fontdict={'fontweight': 'bold'})
plt.xlabel('Countries')
plt.ylabel('Number of Projects')

yticks = list(range(0, 120000, 10000))
plt.yticks(yticks)

#Add Percentage Above Bars
for bar in bars:
    y = bar.get_height()
    percent = round(((y/alln) * 100), 1)
    if (percent > 9):
        plt.text(bar.get_x() - 0.05, y + 1700, str(percent) + '%', fontsize = 8)
    else:
        plt.text(bar.get_x() + 0.05, y + 1700, str(percent) + '%', fontsize = 8)

plt.show()

#Heat Map
plt.figure(figsize = (10, 5), dpi= 300)
plt.title('Correlations Between Data', fontdict={'fontweight': 'bold'})
```

```
sb.heatmap(df.corr(), annot = True, linewidth = 0.5, cmap ='Blues')

#Stacked Bar Graph (High)

#Successes
successes = []
g = 25000
for i in range (0, 5):
    if(i != 5):
        successes.append(df.loc[(df['usd_goal_real'] > g) & (df['usd_goal_real'] <= g *
2) & (df['state'] == 'successful')].count()[0])
        g = g * 2
    else:
        successes.append(df.loc[(df['usd_goal_real'] > 400000) & (df['state'] ==
'successful')].count()[0])


#Failures
failures = []
g = 25000
for i in range (0, 5):
    if(i != 5):
        failures.append(df.loc[(df['usd_goal_real'] > g) & (df['usd_goal_real'] <= g * 2)
& (df['state'] == 'unsuccessful')].count()[0])
        g = g * 2
    else:
        failures.append(df.loc[(df['usd_goal_real'] > 400000) & (df['state'] ==
'unsuccessful')].count()[0])

labels = ['25000 < g <= 50000','50000 < g <= 100000','100000 < g <=
200000','200000 < g <= 400000', 'g > 400000']

plt.figure(figsize = (15, 7.5), dpi= 300)
plt.bar(labels, successes, 0.4, label = 'Successes')
plt.bar(labels, failures, 0.4, bottom = successes, label = 'Failures')
plt.yticks(list(range(0, 30001, 2500)))
plt.legend()
plt.ylabel('Number of Projects')
plt.xlabel('Project Goals')
plt.title('Distribution of Success Depending on Goals (High Goals)',
fontdict={'fontweight': 'bold'})
plt.show()

#Stacked Bar Graph (Low)

#Successes
sless5 = df.loc[(df['usd_goal_real'] > 5000) & (df['state'] == 'successful')].count()[0]
successes = [sless5]
g = 5000
for i in range (0, 4):
    successes.append(df.loc[(df['usd_goal_real'] > g) & (df['usd_goal_real'] <= g +
5000) & (df['state'] == 'successful')].count()[0])
    g = g + 5000


#Failures
lless5 = df.loc[(df['usd_goal_real'] > 5000) & (df['state'] ==
'unsuccessful')].count()[0]
failures = [lless5]
g = 5000
for i in range (0, 4):
    failures.append(df.loc[(df['usd_goal_real'] > g) & (df['usd_goal_real'] <= g +
5000) & (df['state'] == 'unsuccessful')].count()[0])
    g = g + 5000


labels = ['g > 5000','5000 < g <= 10000','10000 < g <= 15000','15000 < g <=20000',
'20000 < g <= 25000']

plt.figure(figsize = (15, 7.5), dpi= 300)
plt.bar(labels, successes, 0.4, label = 'Successes')
plt.bar(labels, failures, 0.4, bottom = successes, label = 'Failures')
plt.legend()
plt.ylabel('Number of Projects')
plt.xlabel('Project Goals')
plt.yticks(list(range(0,190001,10000)))
plt.title('Distribution of Success Depending on Goals (Low Goals)',
fontdict={'fontweight': 'bold'})
plt.show()

#Pie Chart Category
category_names = [e for e, _ in Counter(df['main_category']).most_common()]
category_counts = [e for _, e in Counter(df['main_category']).most_common()]

plt.figure(figsize = (20, 10), dpi= 300)
plt.pie(category_counts, labels = category_names, autopct = '%.2f %%',
textprops={'fontsize': 10})
plt.title('Main Category Percentage', fontdict={'fontweight': 'bold'})
plt.show()

#Pie Chart Category (Success)
category_names = [e for e, _ in
Counter(suc_df['main_category']).most_common()]
category_counts = [e for _, e in
Counter(suc_df['main_category']).most_common()]

plt.figure(figsize = (20, 10), dpi= 300)
plt.pie(category_counts, labels = category_names, autopct = '%.2f %%',
textprops={'fontsize': 10})
plt.title('Main Category Percentage Of Successful Projects', fontdict={'fontweight':
'bold'})
plt.show()

#Categorical Plot
c = sb.catplot(y="main_category", x="backers", data = df)
c.fig.set_size_inches(10, 5)
plt.title('Project Backers Depending on Category', fontdict={'fontweight': 'bold'})
plt.xticks(list(range(0,250001, 25000)))
plt.show()

c.savefig("catplot.png", dpi=500)
```

```
#Encoding For Modelling
ohe = pd.get_dummies(df.state) #Dummie Values for State
df = pd.concat([df, ohe], axis = 1) #Adding Them
df = df.drop(['unsuccessful', 'state'], axis = 1) #Removing Extra Columns

#Turning Country Column to Continent
eu = ['AT', 'BE', 'CH', 'DE', 'DK', 'ES', 'FR', 'GB', 'IE', 'IT', 'LU', 'NL', 'NO', 'SE']
na = ['CA', 'MX', 'US']
asia = ['HK', 'JP', 'SG']
oc = ['AU', 'NZ']

#Changing Values of Column
df.loc[df['country'].isin(eu), 'country'] = 'EU'
df.loc[df['country'].isin(na), 'country'] = 'NA'
df.loc[df['country'].isin(asia), 'country'] = 'AS'
df.loc[df['country'].isin(oc), 'country'] = 'OC'

df = df.rename(columns={'country': 'continent'}) #Renaming Column

ohe = pd.get_dummies(df.continent) #Dummie Values for Continent
df = pd.concat([df, ohe], axis = 1) #Adding Them
df = df.drop(['OC', 'continent'], axis = 1) #Removing Extra Columns

#----------TRANSFORMATION----------

#Scaling - Standardization
scaler = preprocessing.StandardScaler().fit(df[['usd_pledged_real', 'backers']])
standard_dev = scaler.transform(df[['usd_pledged_real', 'backers']])

min_max = preprocessing.MinMaxScaler().fit(df[['usd_pledged_real', 'backers']])
df_minmax = min_max.transform(df[['usd_pledged_real', 'backers']])

print('Mean after standardization:')
print('npledged={:.2f}, backers={:.2f}'
    .format(standard_dev[:,0].mean(), standard_dev[:,1].mean()))
```

```
print('\nStandard deviation after standardization:')
print('pledged={:.2f}, backers={:.2f}'
      .format(standard_dev[:,0].std(), standard_dev[:,1].std()))

print('\nMin-value after min-max scaling:')
print('npledged={:.2f}, backers={:.2f}'
      .format(df_minmax[:,0].min(), df_minmax[:,1].min()))

print('\nMax-value after min-max scaling:')
print('pledged={:.2f}, backers={:.2f}'
      .format(df_minmax[:,0].max(), df_minmax[:,1].max()))

#Aggregation - Adding 2 New Columns
df['success_amounts'] = df['usd_pledged_real'] / df['usd_goal_real']
df['pledge_goal_difference'] = df['usd_pledged_real'] - df['usd_goal_real']

#Heat Map
plt.figure(figsize = (10, 5), dpi= 300)
plt.title('Heatmap After Aggregation & Encoding', fontdict={'fontweight': 'bold'})
sb.heatmap(df.corr(), annot = True, linewidth = 0.5, cmap ='Blues')

#Sampling - After Aggregation & Encoding

#Population's Data Summaries
df.backers.describe()
df.usd_pledged_real.describe()
df.usd_goal_real.describe()
df.success_amounts.describe()
df.pledge_goal_difference.describe()

# Sampling without repetition
sampleNoRep = retriveSamples(r.sample(IDseries.tolist(), k=numberSamples),df)
print(sampleNoRep)

#Sample's Data Summaries
sampleNoRep.backers.describe()
sampleNoRep.usd_pledged_real.describe()
sampleNoRep.usd_goal_real.describe()
sampleNoRep.success_amounts.describe()
sampleNoRep.pledge_goal_difference.describe()



#df.to_csv('new_ks.csv', index=False)

#----------FEATURE SELECTION----------

#Low Variance
columns_to_drop = ['name', 'category', 'main_category'] #Removing Categorical
Features That Will Not Be Used Or Encoded
df = df.drop(columns_to_drop, axis=1)
selector = VarianceThreshold(threshold = 0.3)
selector.fit_transform(df)

df.columns[selector.get_support()] #Only Encoded Categorical Features, So No
Features Are Removed


#----------MODELLING / REGRESSION----------

x = df['backers'].values
y = df['usd_pledged_real'].values

x = x.reshape(-1, 1)
y = y.reshape(-1, 1)

msqe_errors = []
rmse_errors = []

#Train Test Split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state
= 0)

#Linear Regression - Before Scaling
```

```
regressor = LinearRegression()
regressor.fit(x_train,y_train)
y_pred = regressor.predict(x_test)

#Finding Error
msqe = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(msqe)

print(msqe)
print(rmse)

msqe_errors.append(msqe)
rmse_errors.append(rmse)

#Scaling The Data - Standardization
scaler = preprocessing.StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
y_train = scaler.fit_transform(y_train)
y_test = scaler.transform(y_test)

#Linear Regression - After Scaling
regressor = LinearRegression()
regressor.fit(x_train,y_train)
y_pred = regressor.predict(x_test)

#Training Set
plt.figure(figsize = (10, 5), dpi= 300)
plt.scatter(x_train, y_train, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color='blue')
plt.title("Backers vs Pledged Amount (Training Set - Standardized)")
plt.xlabel('Backers')
plt.ylabel('Pledged Amount')
plt.show()

#Testing Set
plt.figure(figsize = (10, 5), dpi= 300)
plt.scatter(x_test, y_test, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color='blue')
plt.title("Backers vs Pledged Amount (Testing Set - Standardized)")
plt.xlabel('Backers')
plt.ylabel('Pledged Amount')
plt.show()

#Polynomial Regression
poly_reg = PolynomialFeatures(degree = 2)
x_poly = poly_reg.fit_transform(x)
lin_reg2 = LinearRegression()
lin_reg2.fit(x_poly, y)

s_x = np.sort(x, axis=None).reshape(-1, 1) #Sorting

plt.figure(figsize = (10, 5), dpi= 300)
plt.scatter(x, y, color = 'red')
plt.plot(s_x, lin_reg2.predict(poly_reg.fit_transform(s_x)), color = 'blue')
plt.title('Backers vs Pledged Amount (Polynomial - D2)')
plt.xlabel('Backers')
plt.ylabel('Pledged Amount')
plt.show()

#Decision Tree Regression
regressor = DecisionTreeRegressor(random_state = 0, max_depth = 2)
regressor.fit(x, y)

y_pred = regressor.predict(x)

plt.figure(figsize = (10, 5), dpi= 300)
x_grid = np.arange(min(s_x), max(s_x), 0.01)
x_grid = x_grid.reshape((len(x_grid), 1))
plt.scatter(x, y, color = 'red')
plt.plot(x_grid, regressor.predict(x_grid), color = 'blue')
plt.title('Backers vs Pledged Amount (Decision Tree)')
plt.xlabel('Backers')
plt.ylabel('Pledged Amount')
plt.show()

#Random Forest
regressor = RandomForestRegressor(n_estimators = 20, random_state = 0)
regressor.fit(x, y)

y_pred = regressor.predict(x)

plt.figure(figsize = (10, 5), dpi= 300)
x_grid = np.arange(min(s_x), max(s_x), 0.01)
x_grid = x_grid.reshape((len(x_grid), 1))
plt.scatter(x, y, color = 'red')
plt.plot(x_grid, regressor.predict(x_grid), color = 'blue')
plt.title('Backers vs Pledged Amount (Random Forest)')
plt.xlabel('Backers')
plt.ylabel('Pledged Amount')
plt.show()

msqe = mean_squared_error(y, y_pred)
rmse = np.sqrt(msqe)

print(msqe)
print(rmse)

msqe_errors.append(msqe)
rmse_errors.append(rmse)

#Finding Error
msqe = mean_squared_error(y, y_pred)
rmse = np.sqrt(msqe)

print(msqe)
print(rmse)

msqe_errors.append(msqe)
rmse_errors.append(rmse)
```

```
plt.figure(figsize = (10, 5), dpi= 300)
plt.bar(['Linear Regression', 'Decision Tree', 'Random Forest'], msqe_errors,
width=0.5)
plt.title('Mean Squared Error', fontdict={'fontweight': 'bold'})
plt.xlabel('Models')
plt.ylabel('Error Rate')

plt.figure(figsize = (10, 5), dpi= 300)
plt.bar(['Linear Regression', 'Decision Tree', 'Random Forest'], rmse_errors,
width=0.5, color='green')
plt.title('Root Mean Squared Error', fontdict={'fontweight': 'bold'})
plt.xlabel('Models')
plt.ylabel('Error Rate')

#Modelling With Sample
#In sample modeling, we did same as population but this time for sampling, but
#we removed the codes because of the similarity.
```