Spring 2023

CMPE 257 - MACHINE LEARNING

PROJECT REPORT

On

"KICKSTARTER - SUCCESS RATE PREDICTION"

Instructor:

JAHAN GHOFRANIHA

Submitted By:

SRIKARI VEERUBHOTLA [015957032]
PRASHANSA EVANGELINE BONAPALLE [016678324]

Table of Contents

Executive Summary

This report presents a Kickstarter success rate prediction project aimed at assisting project creators in understanding the factors that contribute to campaign success and improving their chances of achieving funding goals. By analyzing a dataset of Kickstarter campaigns, we developed a prediction model to determine the likelihood of a campaign's success based on various features.

The project's key objectives were to develop a prediction model to accurately forecast the success or failure of Kickstarter campaigns.

Background

Since its launch in 2009, Kickstarter has transformed the process by which creators bring their ideas to life by giving them access to an online community of potential backers who can contribute money to their ventures. Creators can showcase their project ideas on Kickstarter and ask people who support their vision for funding.

The amount of money that creators need to raise in order to complete their project is specified in their fundraising goal when they create a Kickstarter project. In response, backers contribute money to the projects that resonate to them, frequently in exchange for gifts or other benefits.

A Kickstarter project's success is not assured, though. Several variables can affect whether a project meets its fundraising target or falls short of it.

Both creators and potential backers are very interested in understanding these aspects and precisely forecasting the end result of a project.

Our objective in this paper is to create a classification model that can predict whether a Kickstarter project will be successful or unsuccessful in attaining its funding target. To make these predictions, we will only use the data that was available at the time the initiative was launched. We will find key project characteristics—such as project type, funding target, duration, and other pertinent features—that are suggestive of project success by reviewing prior Kickstarter data.

We want to gain insights from this analysis that will help creators optimize their project launches and potential backers choose the projects they should support.

By harnessing the power of machine learning, we can leverage historical Kickstarter data to develop a robust predictive model that improves the overall efficiency and effectiveness of the crowdfunding ecosystem.

Problem Statement

Goal is to predict whether a kickstarter project proposal will succeed or fail in achieving its fundraising goals based on information from the project launch.

Differentiator/Contribution

· Unlike existing studies, in this project, we incorporated a combination of both traditional and novel features for predicting Kickstarter success. It includes not only campaign-specific features like funding goal and duration but also additional features such as the creator's previous successful campaigns, the average pledge amount.

· We have also created a new feature “percentage of pledged/goal”.

· By integrating these unique features into prediction models, the project aims to provide more accurate success rate predictions for kickstarter campaigns.

Methodology

The dataset from "kickstarter_data_with_features.csv" is loaded and preprocessed. In the preprocessing, handling missing values, encoding categorical variables, and scaling numerical features were done. Exploratory data analysis (EDA) is conducted to gain insights into the dataset and identify any patterns or relationships between variables.

For modeling, a machine learning algorithm such as a decision tree, logistic regression and Gradient Boosting Decision Tree is applied to build a prediction model. The dataset is split into training, validation, and testing sets. The model is trained on the training set and fine-tuned using techniques like cross-validation and hyperparameter tuning. The performance of the model is evaluated using metrics such as accuracy, precision, recall, and F1 score from the confusion matrix.

Data Preprocessing

1. Data Cleaning:

· Removed irrelevant or redundant features that do not contribute to the prediction task.

· Handled missing values by either deleting rows with missing values or applying imputation techniques such as mean, median, or regression imputation.

· Identified and handled outliers that may negatively impact the model's performance.

2. Feature Engineering:

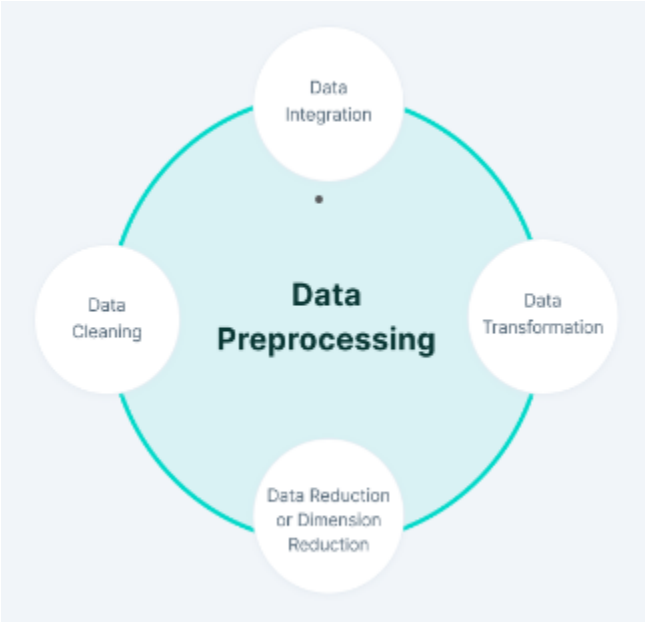
· Create new features that might be more informative for predicting Kickstarter success.

· Created a new feature to calculate percentage of pledged/goal.

· Extracted date, time, hour, minute, month and year from the deadline and created new features with them.

· Converted categorical variables into numerical representations using label encoding.

· Normalized numerical features to ensure they are on a similar scale, preventing any feature from dominating the model due to its larger magnitude.



3. Text Preprocessing:

· Mainly used for transforming raw textual data into a format suitable for analysis and machine learning algorithms. Here are some common techniques that were used:

· **Tokenization:** Splitting the text into individual tokens or words. This step helps break down the text into smaller units for further analysis.

· **Stopword Removal**: Removing commonly occurring words (such as "and," "the," "is") that do not carry significant meaning and are not useful for analysis.

· **Lowercasing:** Converting all text to lowercase. This step ensures that words with different capitalizations are treated as the same and avoids duplication of features.

· Removal of Punctuation: Removing punctuation marks such as periods, commas, and quotation marks that do not provide meaningful information for analysis.

· **Lemmatization:** Reducing words to their base or dictionary form (lemmas). For example, converting "running" and "ran" to "run." Lemmatization helps consolidate related words and reduces the dimensionality of the feature space.

· **Stemming:** Reducing words to their stem or root form. For example, converting "running" and "ran" to "run." Stemming is a simpler process than lemmatization and can sometimes lead to non-words.

· **Removal of Special Characters and Numbers:** Removing symbols, special characters, and numerical values from the text data, as they may not contribute significantly to the analysis.

· **Handling Contractions:** Expanding contractions like "don't" to "do not" or "won't" to "will not" to ensure consistent representation.

· **Removal of HTML Tags**: If the text data contains HTML tags, they can be removed to extract the actual text content.

· **Spell Checking:** Correcting spelling errors using techniques such as dictionary-based approaches or language models.

4. **Text Data Exploration:**

· Word clouds reveal frequently occurring words or phrases in successful or failed campaigns, providing initial insights into the significance of certain keywords.

· Sentiment analysis of project descriptions indicates positive sentiment is more prevalent in successful campaigns.

Exploratory Data Analysis (EDA):

EDA provides insights into the characteristics of the dataset and helps identify patterns, relationships, or anomalies in the data.

Data Profiling:

· The dataset contains 20632 records and 60 variables, including features such as campaign category, funding goal, duration, number of backers, and project description.

· Numerical variables, such as funding goal and duration, exhibit a wide range of values, with varying means and standard deviations.

· Categorical variables, such as campaign category and country, have different levels or categories with varying frequencies.

Correlation Matrix Heat Map:

· We created a correlation matrix heat map to visualize the relationships between numerical variables. The heat map helps identify potential correlations and dependencies among features.

· The heat map reveals a strong positive correlation between the number of backers and campaign success, indicating that campaigns with more backers tend to have higher success rates.

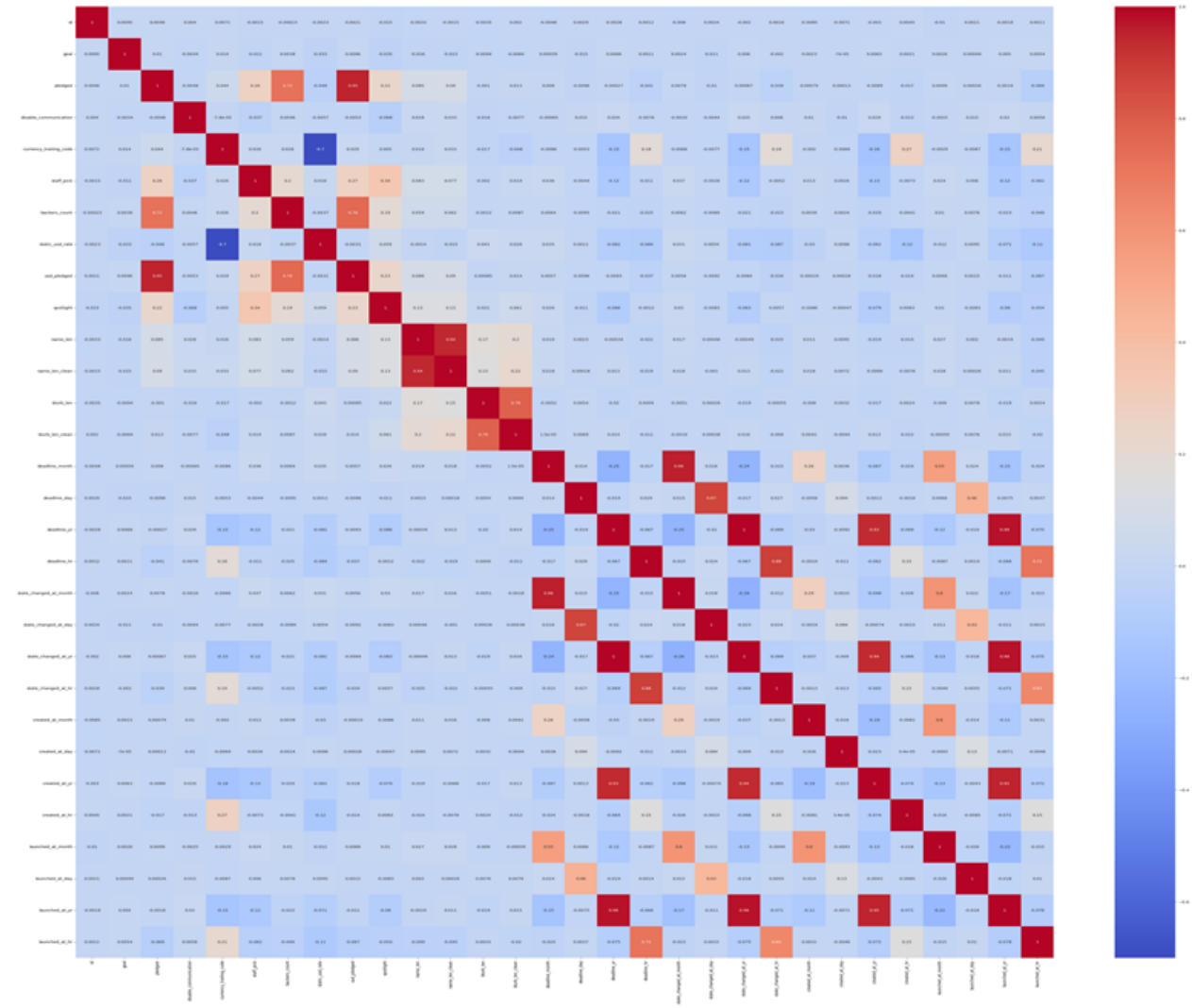


Fig. Heat Map

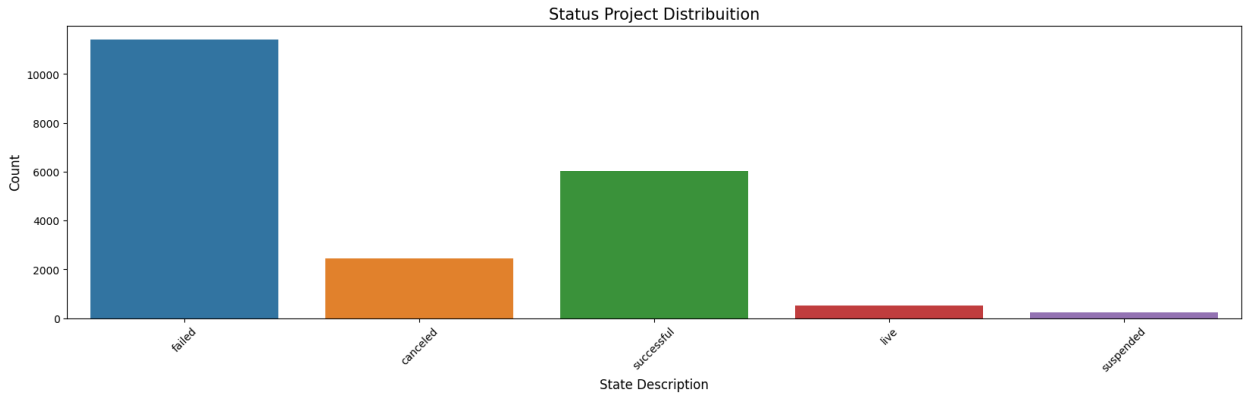
Visualization Techniques:

· The distribution of funding goals is right-skewed, indicating that most campaigns have relatively lower funding goals, while a few have extremely high goals.

· A bar plot of campaign categories shows that the most prevalent categories are "Technology," "Art," and "Film & Video."

· Scatter plots and correlation matrices reveal potential correlations between variables, such as a positive correlation between the number of backers and campaign success.

“State” Feature:



Bar graph for the Status feature in dataset. State Percentile in %:

Failed	55.33%
Successful	29.17%
Canceled	11.92%
Live	2.46%
Suspended	1.11%

The above graph shows the distribution of a variable called "state" in a dataset and visualizes it using a bar plot. The distribution shows the frequency of each unique value in the "state" column. The code then converts the frequencies into percentages and rounds them to two decimal places.

The resulting percentages are printed out, and a bar plot is created to visualize the distribution.

“Country” Feature:

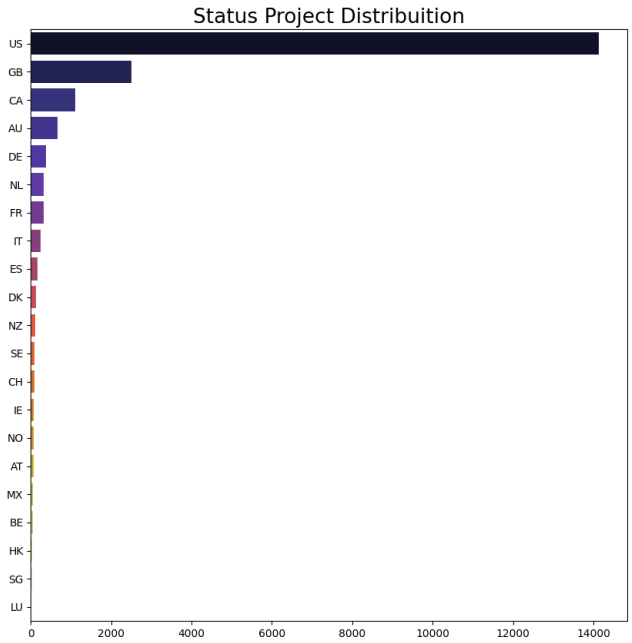


Fig. Status Project Distribution

The above graph shows the percentage distribution of different countries in a dataset and creates a bar plot to visualize the frequency of each country. The resulting percentages indicate the proportion of each country in the dataset. The bar plot allows us to easily compare the frequencies of different countries and observe any variations or patterns. This analysis helps us understand how the countries are distributed in the dataset and provides a clear visual representation of their frequencies.

“Normalization to understand the distribution of the pledge”:.

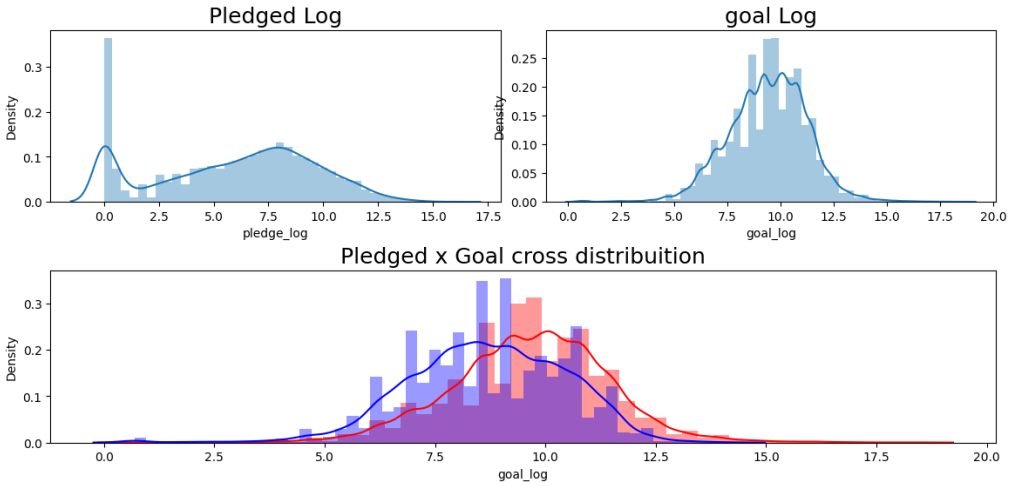


Fig. Pledged x Goal distribution plot

The first graph shows the distribution of the logarithm of pledged amounts ("pledge_log"). By taking the logarithm, the range of values is compressed, making it easier to observe the distribution pattern. This graph helps identify the central tendencies, spread, and potential outliers in the pledged amounts.

The second graph displays the distribution of the logarithm of goal amounts ("goal_log"). Again, taking the logarithm helps to reveal the distribution characteristics more clearly. This graph provides insights into the distribution of project goals, including the spread and central tendencies.

The third graph represents a cross-distribution of the logarithm of goals for failed (in red) and successful (in blue) projects. This graph allows for a direct comparison between the goal distributions of these two project outcomes. It helps identify any differences or overlaps in goal ranges between failed and successful projects.

“Understanding of Goal and pledged by its State”:

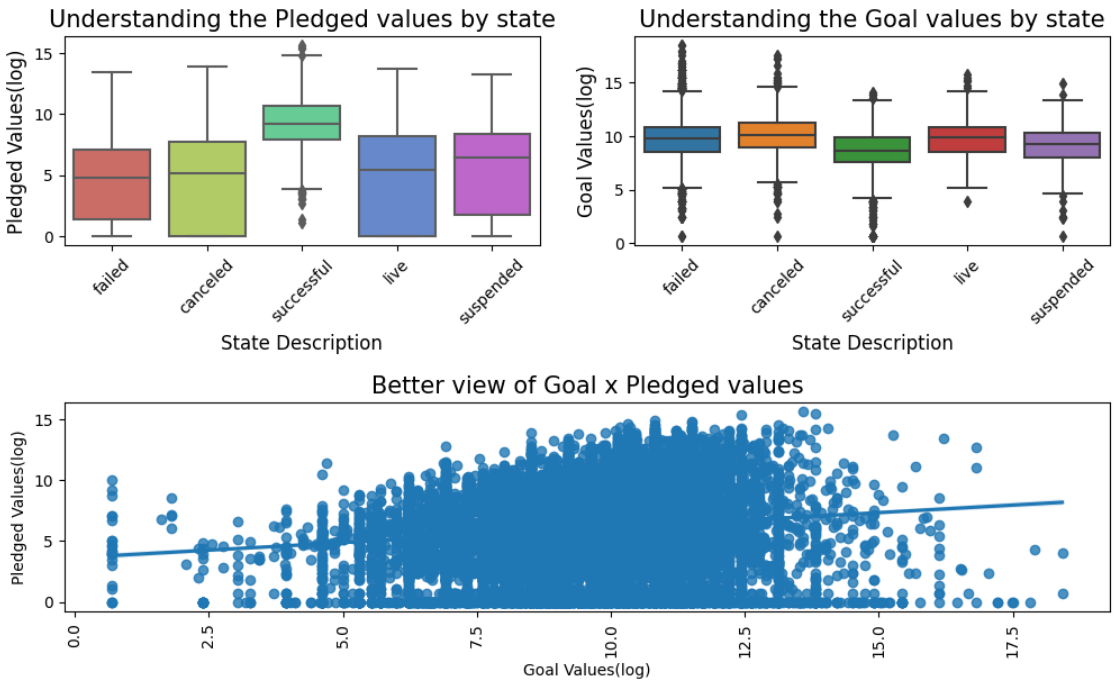


Fig. Scatter plot of Goal x Pledged values with regression line

The first graph is a box plot that shows the distribution of pledged values for each state. The y-axis represents the logarithm of pledged values ("pledge_log"), which helps to normalize the data. The x-axis represents the different states of the projects. This graph allows us to compare the median, quartiles, and outliers of the pledged values across different states. It provides a visual understanding of how the distribution of pledged values varies depending on the state of the project.

The second graph is a box plot that displays the distribution of goal values for each state. Similar to the first graph, the y-axis represents the logarithm of goal values ("goal_log"), and the x-axis represents the project states. This graph enables a comparison of the goal value distributions across different states. It helps identify any differences or similarities in goal values among the states.

The third graph is a scatter plot with a regression line. It shows the relationship between goal values (logarithm scale on the x-axis) and pledged values (logarithm scale on the y-axis) for all projects. The plot allows us to observe the overall trend between these two variables and identify any patterns or correlations. The regression line provides an estimation of the relationship between goal and pledged values.

How many categories Fail and Succeed?

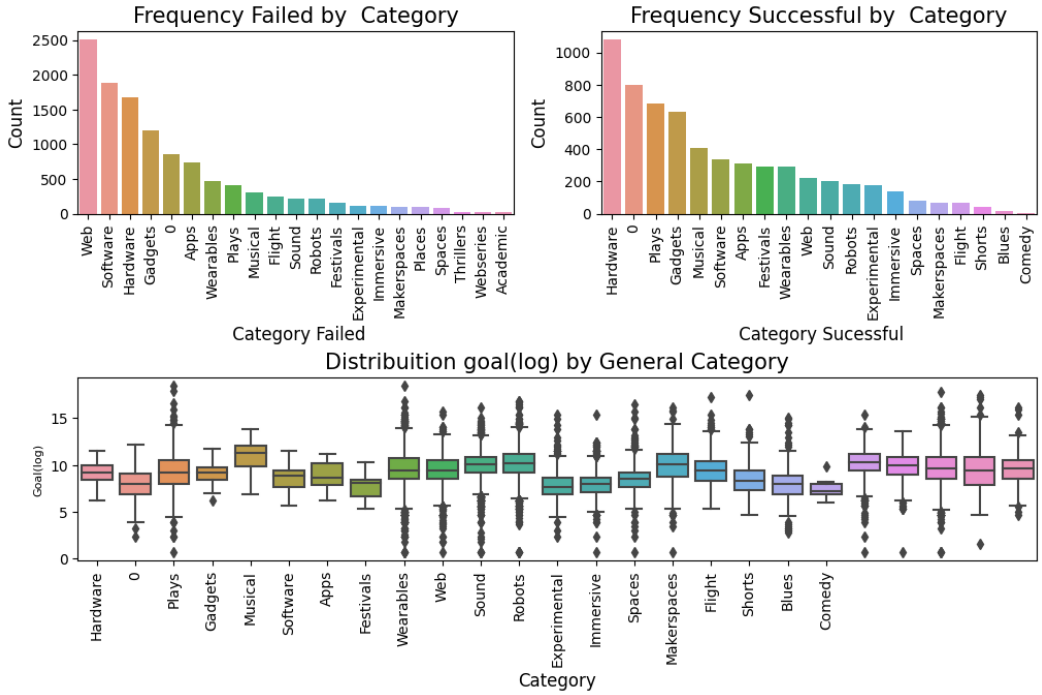


Fig. Box plots depicting the distribution goal(log) by General Category

The first graph is a bar plot that shows the frequency of categories in failed projects. Each bar represents a category, and the height of the bar represents the count of failed projects in that category. The x-axis represents the categories, and the y-axis represents the count of failed projects. This graph helps identify the categories that have a higher number of failed projects.

The second graph is a bar plot that displays the frequency of categories in successful projects. Similar to the first graph, each bar represents a category, and the height of the bar represents the count of successful projects in that category. The x-axis represents the categories, and the y-axis represents the count of successful projects. This graph helps identify the categories that have a higher number of successful projects.

The third graph is a box plot that illustrates the distribution of goal values (logarithm scale on the y-axis) for each category. The x-axis represents the categories, and the y-axis represents the logarithm of goal values. This graph allows us to compare the distributions of goal values across different categories. It helps identify any variations in goal values among different categories.

Distribution in category values as a success or failure:

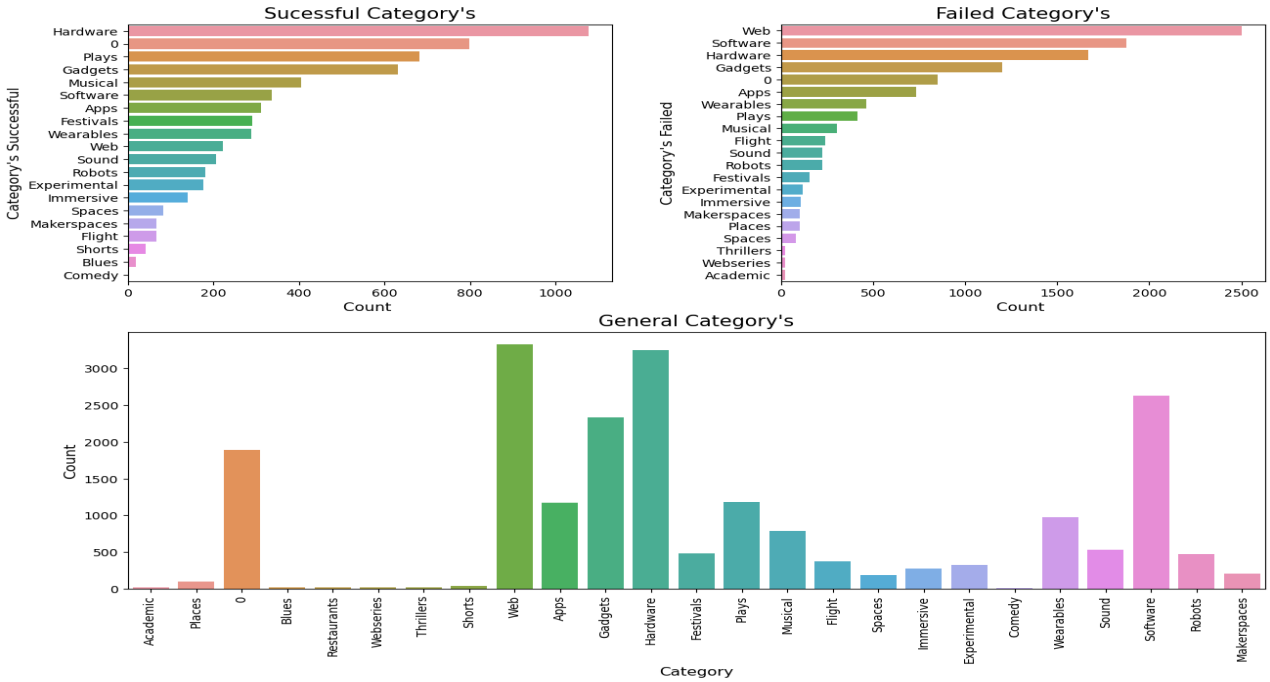


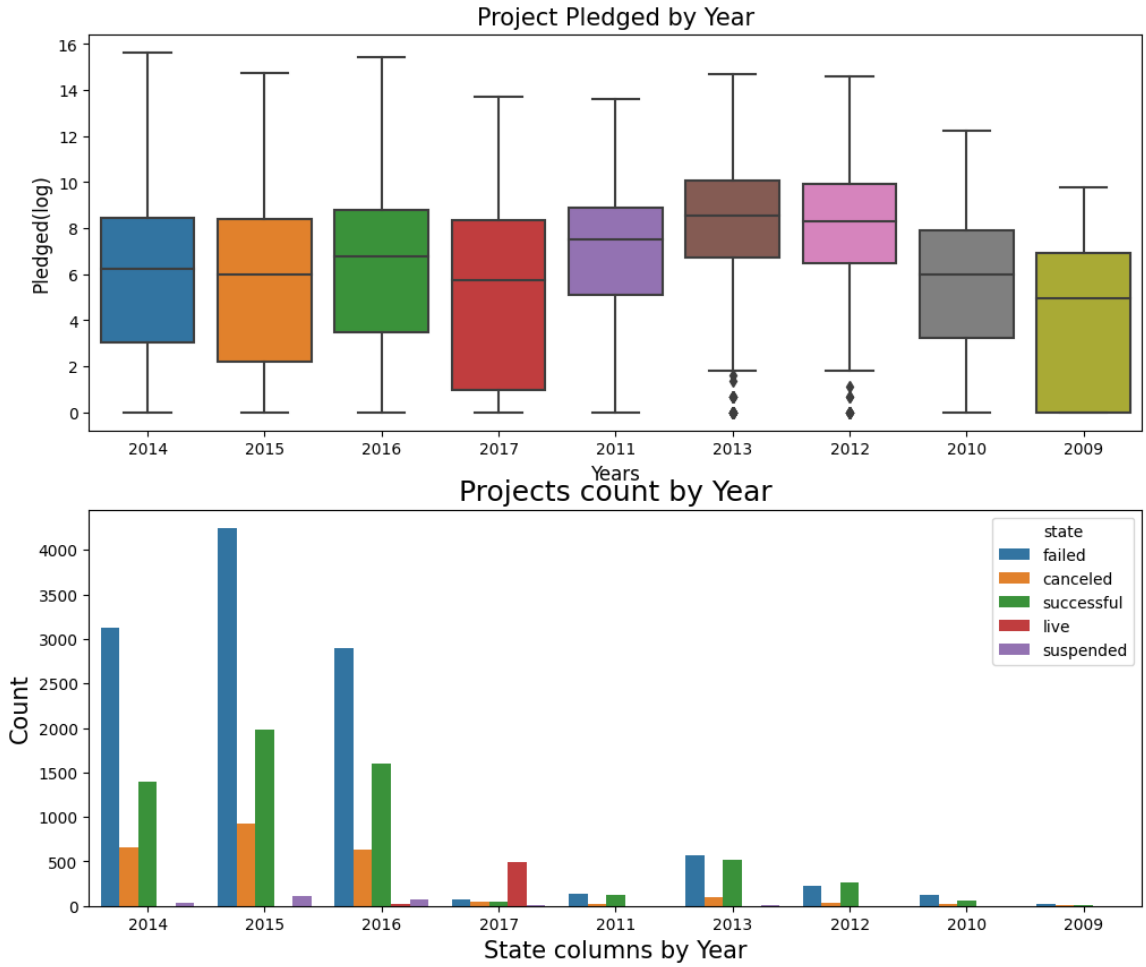
Fig. Bar plot depicting the most and least successful and failed categories

The first graph is a horizontal bar plot that displays the top 25 categories in terms of failed projects. Each bar represents a category, and the length of the bar represents the count of failed projects in that category. The y-axis represents the categories, and the x-axis represents the count of failed projects. This graph helps identify the categories that have a higher number of failed projects.

The second graph is a horizontal bar plot that shows the top 25 categories in terms of successful projects. Similar to the first graph, each bar represents a category, and the length of the bar represents the count of successful projects in that category. The y-axis represents the categories, and the x-axis represents the count of successful projects. This graph helps identify the categories that have a higher number of successful projects.

The third graph is a bar plot that illustrates the overall distribution of categories. Each bar represents a category, and the height of the bar represents the count of projects in that category, regardless of their success or failure. The x-axis represents the categories, and the y-axis represents the count of projects. This graph provides an overview of the distribution of projects across different categories.

Project Count by Year:



The first graph is a box plot that displays the distribution of pledged amounts (logarithm scale on the y-axis) for each year (x-axis). This graph helps understand the variation in pledged amounts across different years. The box plot shows the median, quartiles, and any outliers in the distribution of pledged amounts for each year.

The second graph is a count plot that shows the number of projects (y-axis) for each year (x-axis), categorized by their state (represented by different colors). This graph provides an overview of the count of projects in each state category (failed, successful, etc.) across different years. It allows us to observe any trends or patterns in project counts by year and state.

Creating a new feature to calculate percentage of pledged / goal

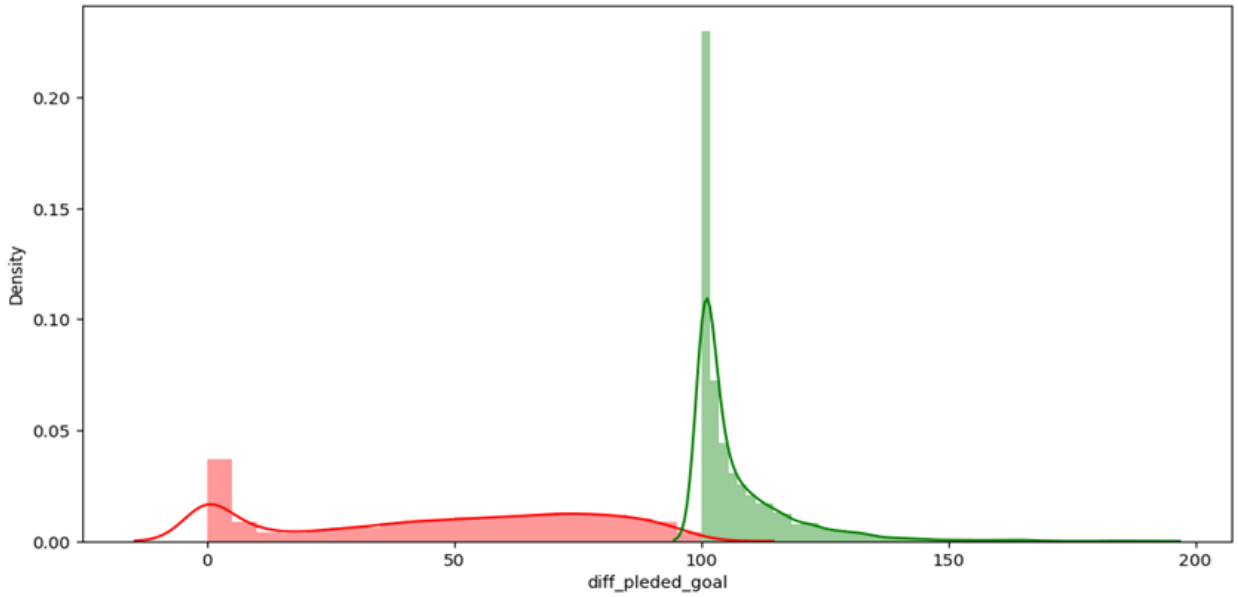


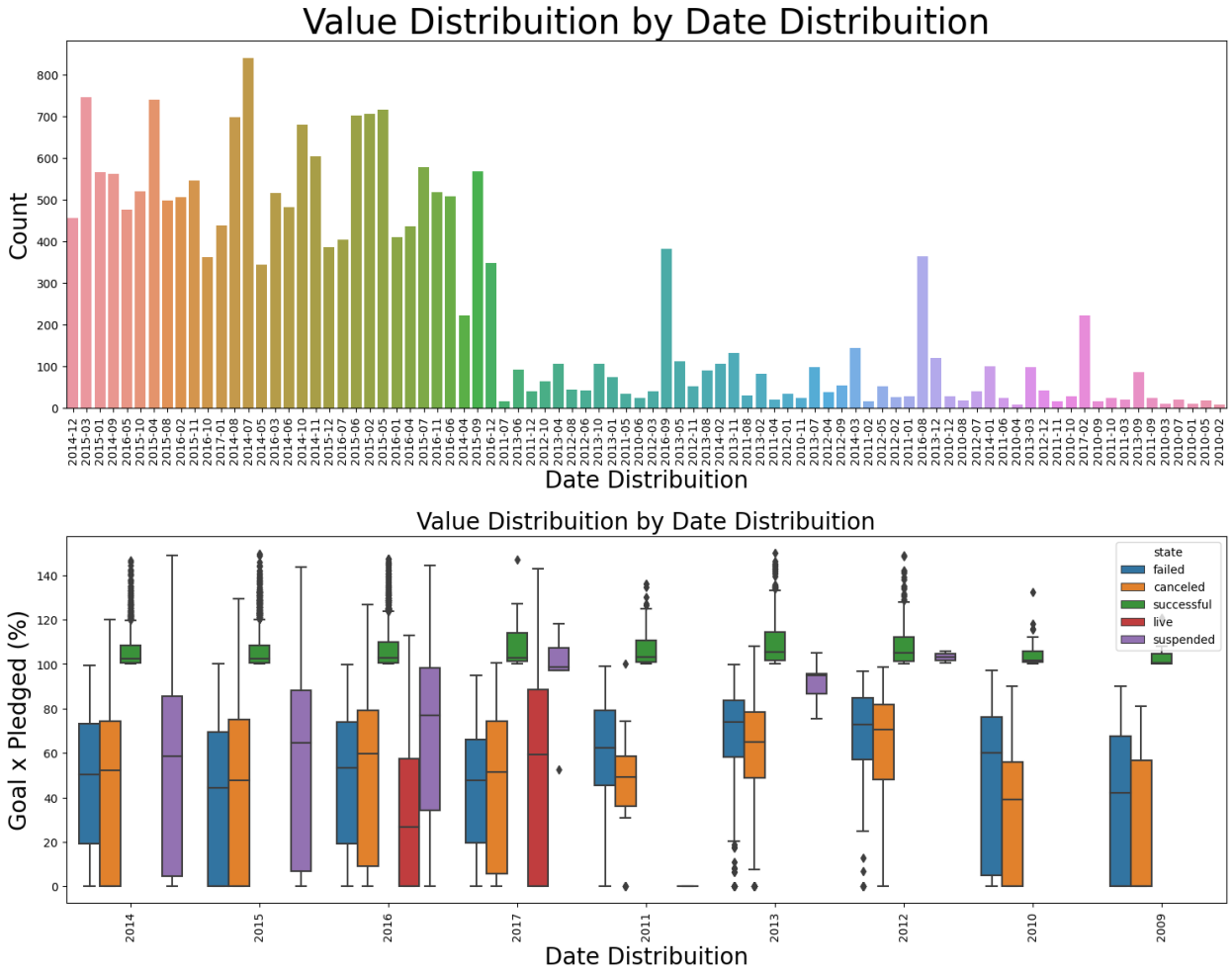
Fig. New feature calculates percentage of "pledged/goal"

A distribution plot to analyze the difference between pledged amounts and goal amounts in projects. The difference is calculated as a percentage by dividing the logarithm of the pledged amount by the logarithm of the goal amount and then multiplying by 100.

The graph displays two distributions, each represented by a colored curve. The red curve represents the distribution of the difference between pledged and goal amounts for failed projects, while the green curve represents the distribution for successful projects.

The x-axis represents the percentage difference between pledged and goal amounts, ranging from 0% to 200%. The y-axis represents the density or frequency of projects with a particular percentage difference.

Date Distribution:



The first graph is a count plot that displays the count of projects (y-axis) for each date distribution (x-axis). The date distribution is represented by months and years. The graph provides an overview of the number of projects launched in each month and year. The x-axis labels are rotated for better readability. This visualization helps identify any patterns or trends in the project launch dates.

The second graph is a box plot that shows the distribution of the percentage difference between pledged and goal amounts (y-axis) for different years (x-axis). The box plot is further categorized by the state of the projects, represented by different colors. The percentage difference is restricted to values below 150% to focus on the main distribution. The x-axis labels represent the years, and they are also rotated for better visibility. This graph allows for comparisons of the goal-to-pledged percentage difference across different years and states. It helps identify any variations or outliers in the performance of projects in meeting their funding goals.

Backers Count:

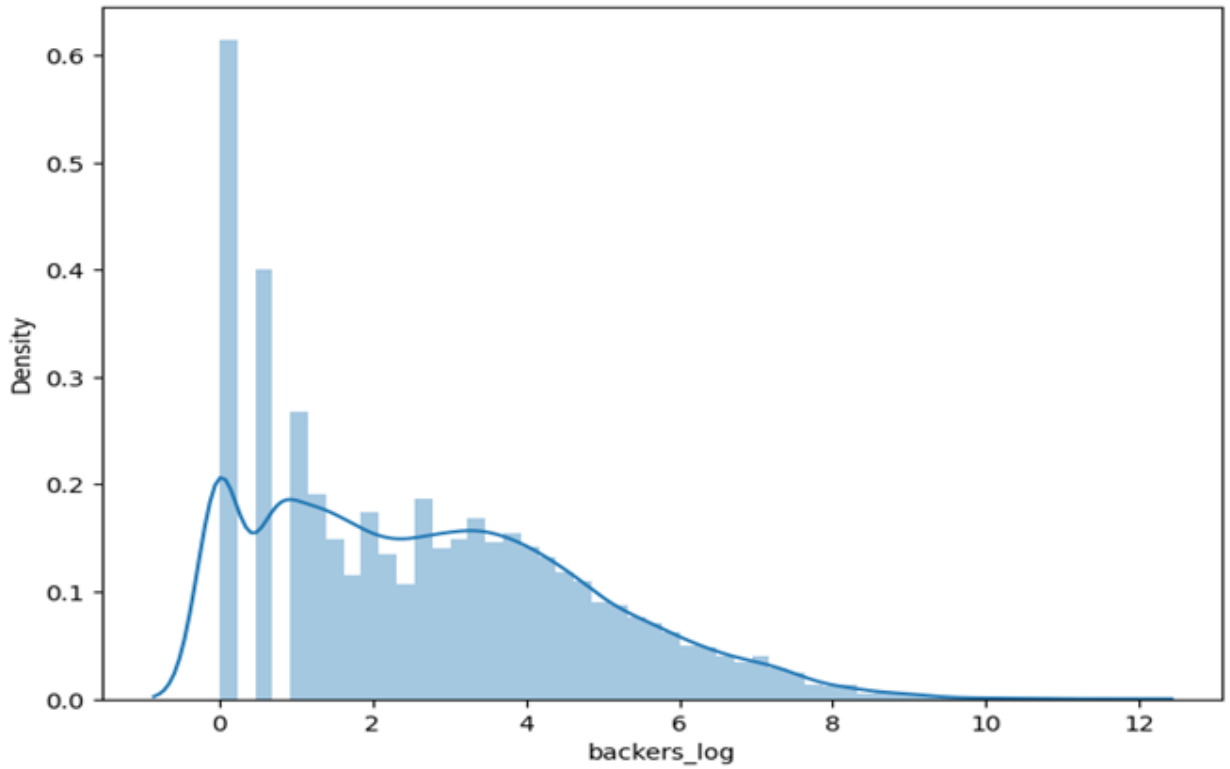
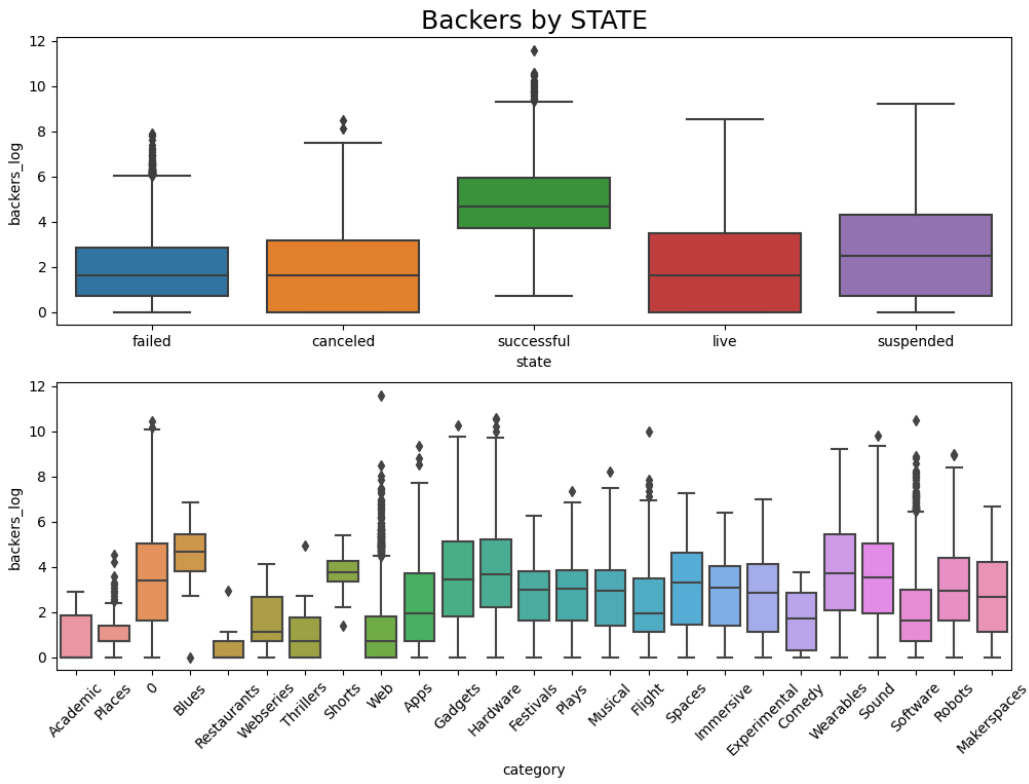


Fig. "Backers_count" in terms of logarithmic

The graph displays the distribution of the transformed number of backers (backers_log) on the x-axis and the density or frequency of projects on the y-axis. The density curve represents the shape of the distribution, while the histogram-like bars represent the frequency of projects at different values of the transformed number of backers.

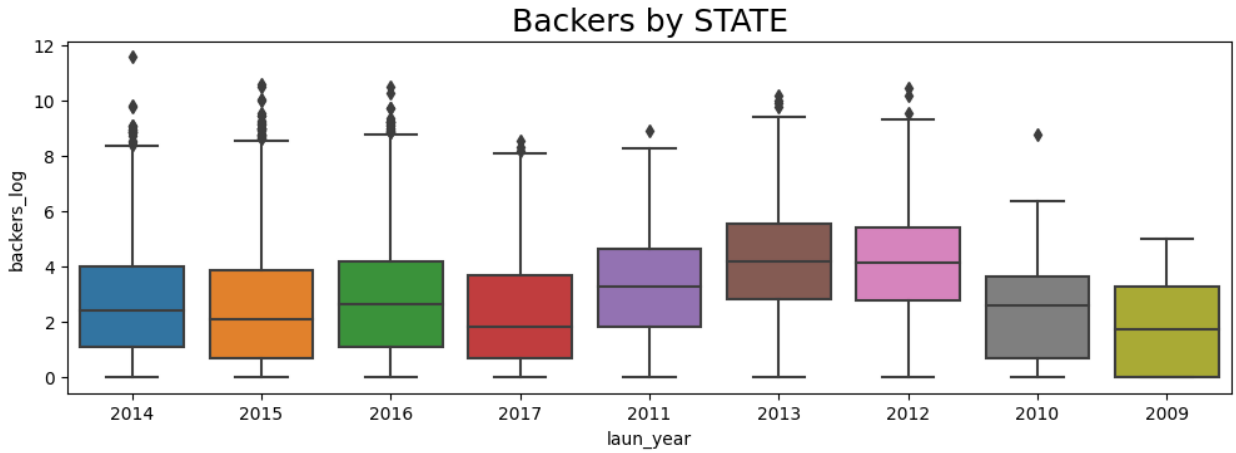
Backers with respect to the category:



The first graph, located in the top subplot (211), is a box plot that compares the distribution of the transformed number of backers for different project states (successful, failed, etc.). The x-axis represents the project state, while the y-axis represents the transformed number of backers. The box plot displays the median, quartiles, and any outliers for each project state. This visualization allows for a comparison of the distribution of backers among different project states, providing insights into the level of support and engagement for successful and unsuccessful projects.

The second graph, located in the bottom subplot (212), is a box plot that compares the distribution of the transformed number of backers for different project categories. The x-axis represents the project category, while the y-axis represents the transformed number of backers. The x-axis labels are rotated for better readability. This visualization helps to identify any variations or differences in the number of backers across different project categories. By comparing the median, quartiles, and outliers for each category, readers can gain insights into the level of interest and support for projects in different categories.

Backers with respect to Year:



The graph displays a box plot in the top subplot (211), where the x-axis represents the launch year of the projects, and the y-axis represents the transformed number of backers. The box plot provides a visual representation of the distribution of backers across different launch years. It shows the median, quartiles, and any outliers for each year.

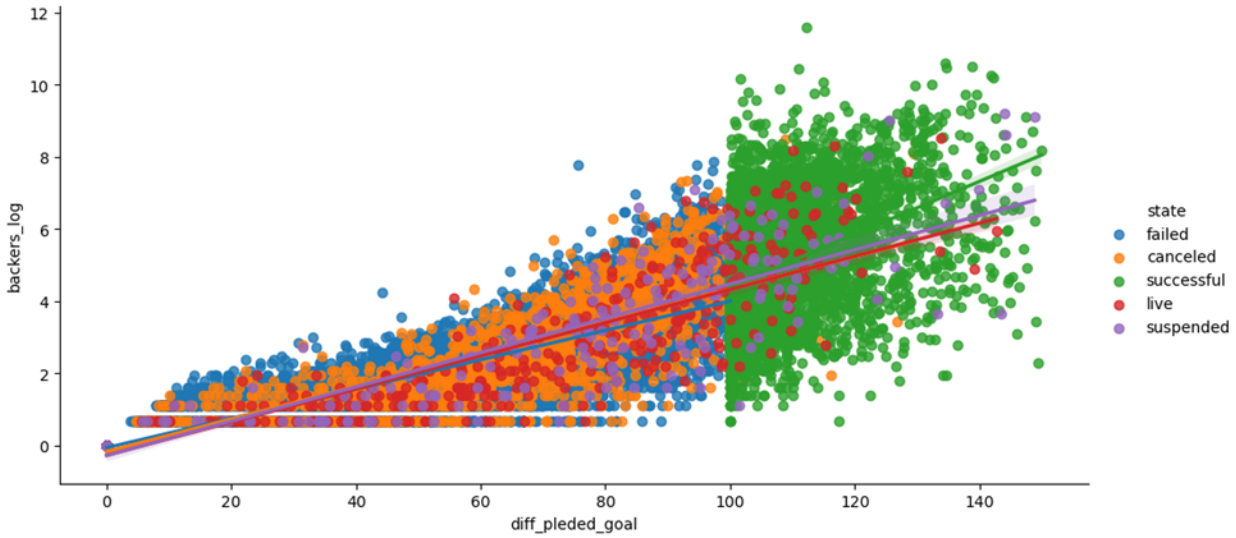


Fig. Relation of Backers and % of goal reached

The graph displays a scatter plot where each point represents a project. The x-axis represents the percentage of goal reached, indicating how much of the project's funding goal was achieved. The y-axis represents the transformed number of backers. Each point is colored based on the state of the project (successful or failed), allowing for visual differentiation between the two categories.

The linear regression line represents the overall trend or relationship between the percentage of goal reached and the number of backers. It provides an estimate of how the number of backers changes as the percentage of goal reached increases or decreases. By including this line, the graph allows readers to observe any general patterns or trends in backers' response as the project's funding goal is met or not.

This visualization helps in understanding the relationship between the percentage of goal reached and the number of backers. It can provide insights into how project progress toward its funding goal affects backers' engagement and support. Additionally, the differentiation of successful and failed projects allows for comparisons and analysis of how the relationship differs between these two states.

Implementation & Results:

The "kickstarter_data_with_features.csv" file is imported into a Python environment using libraries such as pandas and scikit-learn. Data preprocessing, feature engineering, model implementation, and result analysis are performed collaboratively by the team members.

After training and evaluating the model using the training and validation sets, it is applied to predict the success or failure of Kickstarter campaigns. The model's predictions are compared against the actual outcomes of campaigns in the testing set to assess its accuracy and effectiveness.

Srikari Veerubhotla: I have worked on the initial Data preprocessing. I have written codes for Decision trees and Gradient Boosting Decision trees. Data preprocessing is a critical step in machine learning, and it involves handling missing data, encoding categorical variables, and splitting the dataset into features and target variables. By completing this step, I have prepared the dataset for model training. Moving on to the models, Decision Trees utilize a recursive splitting approach based on features to make decisions, while Gradient Boosting Decision Trees iteratively enhance weak learners using gradient descent. These algorithms are powerful and widely used in various applications. Model evaluation, hyperparameter tuning, and model interpretation, has been performed to optimize and gain insights from these models.

Prashansa Evangeline Bonapalle: I have worked on Exploratory Data Analysis, Text preprocessing and Logistic regression. Exploratory Data Analysis (EDA), Text Preprocessing, and Logistic Regression. In EDA, I have explored the dataset using various statistical techniques and visualizations to understand its characteristics, identify patterns, and detect any outliers or missing values. Text preprocessing involves cleaning and transforming textual data by removing unnecessary elements, standardizing text formats, and preparing it for analysis. Lastly, Logistic Regression is a powerful algorithm for binary classification tasks, where you have likely built a model to predict the probability of an instance belonging to a specific class. These steps are crucial in gaining insights from the data, preparing it for analysis, and building predictive models.

Model Training and Evaluations:

1. Logistic Regression:

The logistic regression algorithm is chosen as the predictive modeling technique.

The dataset is split into training data (X_train, y_train) and testing data (X_test) to evaluate the model's performance.
The logistic regression model is initialized with the "l2" penalty, which helps prevent overfitting by adding a regularization term to the loss function.
The model is trained using the fit() method, which optimizes the model parameters based on the training data.
During training, the model learns the relationships between the input features (X_train) and the target variable (y_train), aiming to predict the state (successful or failed) of a project based on its characteristics.

Model Prediction:

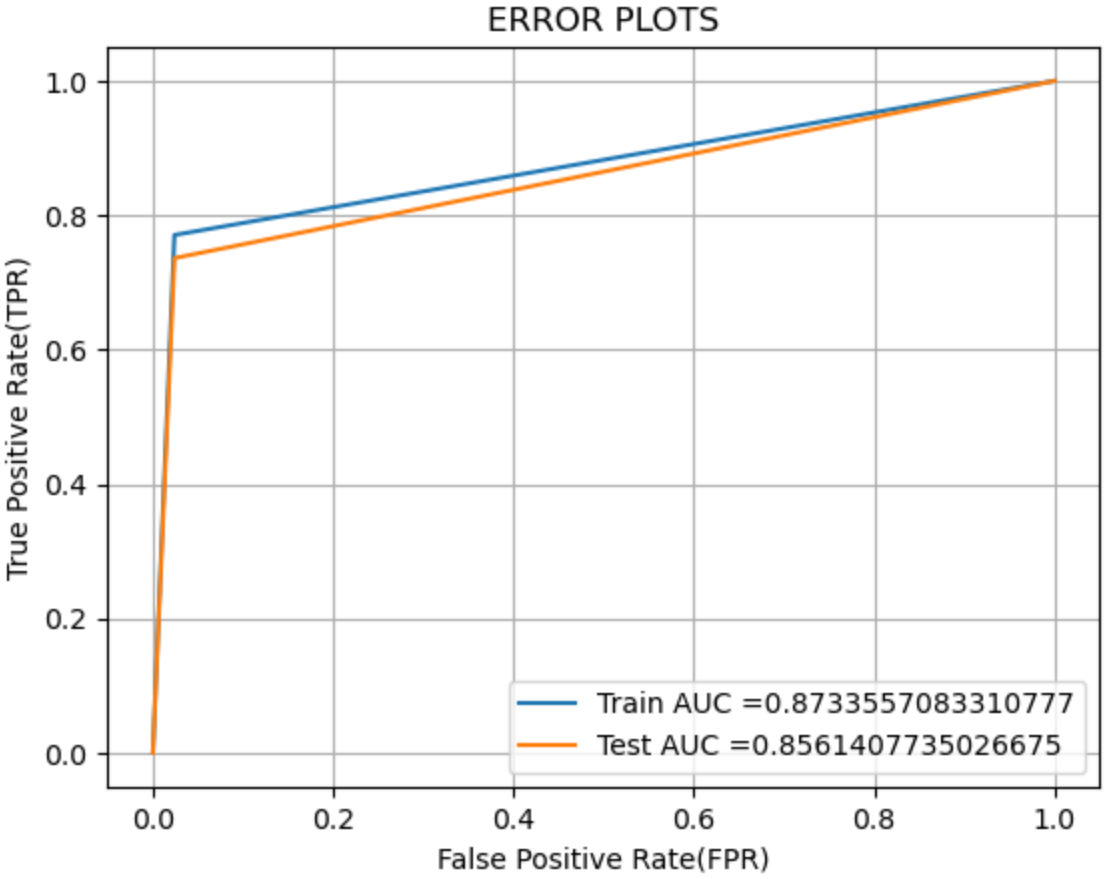
Once the logistic regression model is trained, it can make predictions on both the training and testing data.
The model uses the learned parameters to predict the state of the training data using the predict() method, resulting in y_train_pred.
Similarly, the model predicts the state of the testing data (X_test) using the predict() method, resulting in y_test_pred.

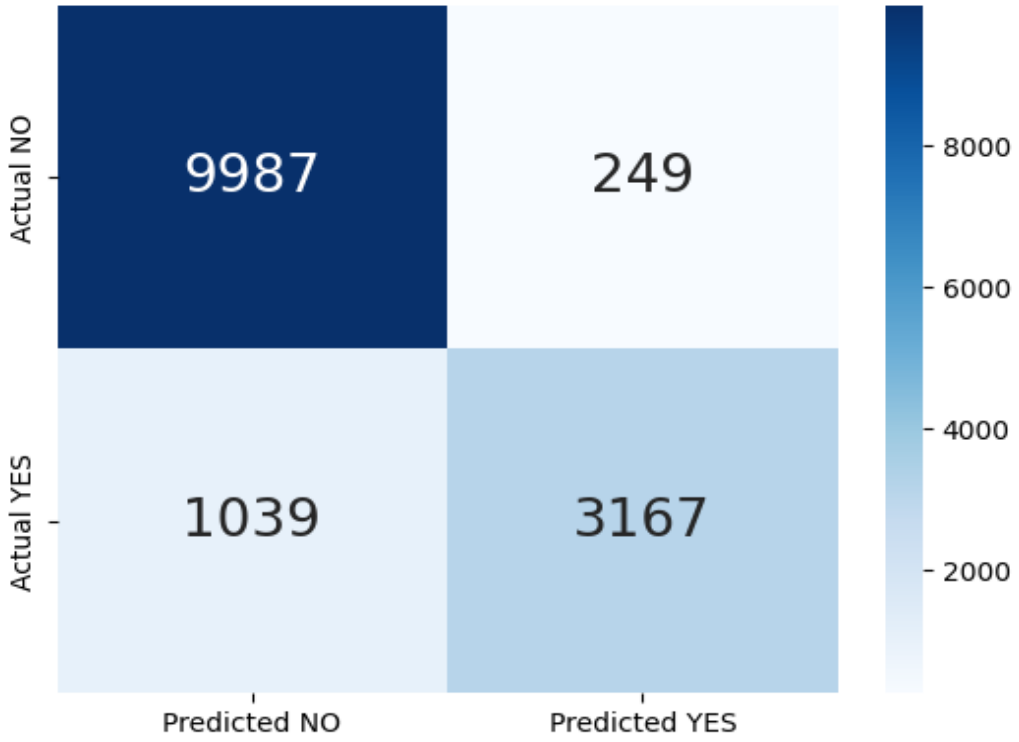
Evaluation Metrics:

To assess the performance of the logistic regression model, several evaluation metrics are computed.
The confusion matrix is calculated for both the training and testing data using the confusion_matrix() function from sklearn.metrics.
The confusion matrix provides a tabular representation of the model's predictions and the actual labels, allowing for the calculation of various metrics such as accuracy, precision, recall, and F1-score.
Additionally, the ROC curve is plotted to visualize the trade-off between the true positive rate (TPR) and false positive rate (FPR). The area under the curve (AUC) is computed as a measure of the model's discriminatory power.

Visualization:

The confusion matrices are visualized as heatmaps using the heatmap() function from the seaborn library.
Heatmaps provide a visual representation of the confusion matrices, with annotations to indicate the number of correct and incorrect predictions.
The ROC curve is plotted using the plot() function from matplotlib, displaying the TPR on the y-axis and the FPR on the x-axis. The AUC values are also displayed on the plot.


ERROR PLOTS
True Positive Rate(TPR)
False Positive Rate(FPR)
Train AUC =0.8733557083310777
Test AUC =0.8561407735026675
0.0
0.2
0.4
0.6
0.8
1.0


Actual NO
Actual YES
9987
249
1039
3167
Predicted NO
Predicted YES
8000
6000
4000
2000

2. Decision Tree Classifier:

Batch Prediction Function:
The function batch_predict is defined to make predictions in batches.
It takes a classifier (clf) and data as input.
It initializes an empty list, y_data_pred, to store the predicted probabilities.
The function loops through the data in batches of size 1000 (except for the remaining data points) and predicts the probabilities using the predict_proba() method of the classifier.
The predicted probabilities for each batch are appended to the y_data_pred list.
Finally, the function returns the list of predicted probabilities.

Hyperparameter Tuning with Decision Tree Classifier:
The Decision Tree Classifier is chosen as the algorithm for hyperparameter tuning.
Two hyperparameters, min_samples_split and max_depth, are specified with different values to be evaluated.
A loop is created to iterate through all combinations of min_samples_split and max_depth.
For each combination, a Decision Tree Classifier is initialized with the given hyperparameters and the 'balanced' class weight.
The classifier is fitted to the training data (X_train, y_train).
Using the batch_predict function, the predicted probabilities are obtained for both the training data (y_train_pred) and the testing data (y_test_pred).
The area under the ROC curve (AUC) is calculated using the roc_auc_score() function from sklearn.metrics for both the training and testing data.
The AUC values for each combination of hyperparameters are stored in the train_auc and test_auc lists.

Grid Search Cross-Validation:
GridSearchCV is used for performing grid search with cross-validation to find the best hyperparameters.
A Decision Tree Classifier with 'balanced' class weight is initialized.
The parameter grid is defined with different values for max_depth and min_samples_split.
GridSearchCV is executed with 3-fold cross-validation, using the roc_auc as the scoring metric.
The best hyperparameters are determined based on the highest mean_test_score obtained during cross-validation.

Visualization:

A heatmap is created to visualize the mean_train_score and mean_test_score for each combination of hyperparameters.

The mean_train_score and mean_test_score are grouped and displayed in separate subplots.

The values in the heatmap represent the AUC scores, and annotations are added to provide more information.

Model Evaluation:

The best estimator obtained from the grid search, dt_best, is fitted to the training data.

Using the batch_predict function, the predicted probabilities are obtained for both the training data (y_train_pred) and the testing data (y_test_pred).

The root mean squared error (rms) and the R-squared score are calculated to evaluate the performance of the model.

The predicted and actual states are stored in the DataFrame, xxx, for the testing data.

ROC Curve and Confusion Matrix:

The ROC curve is plotted using the true positive rate (TPR) and false positive rate (FPR) values obtained from the training and testing data.

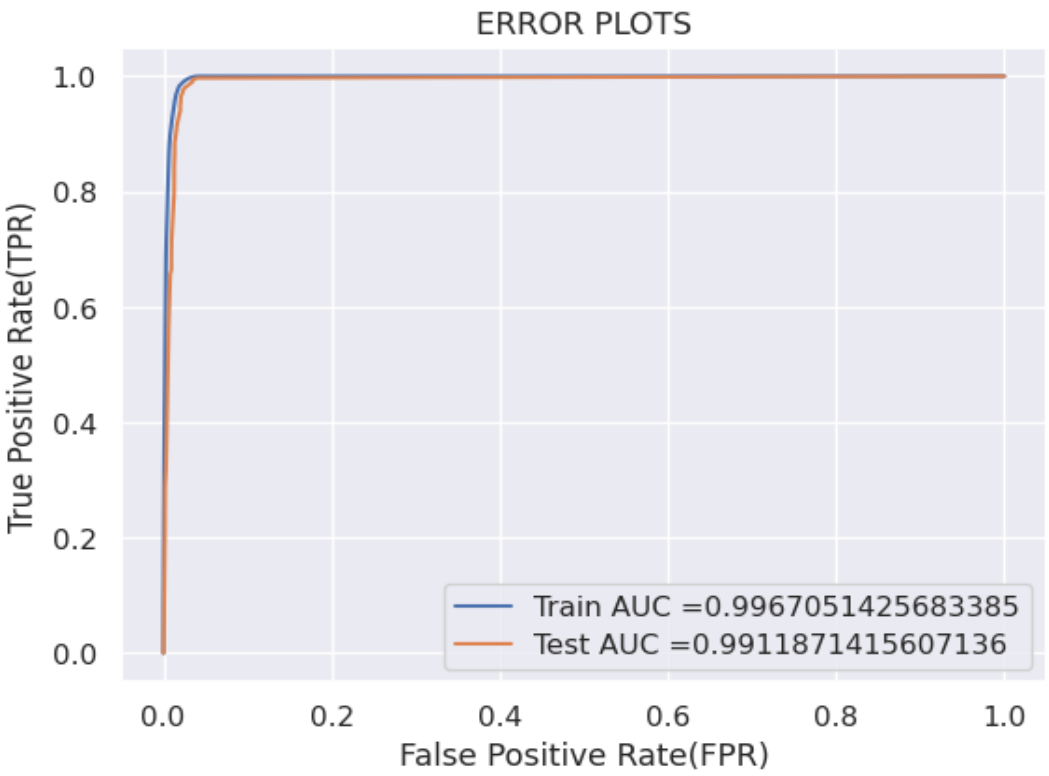
The area under the ROC curve (AUC) is calculated and displayed on the plot.

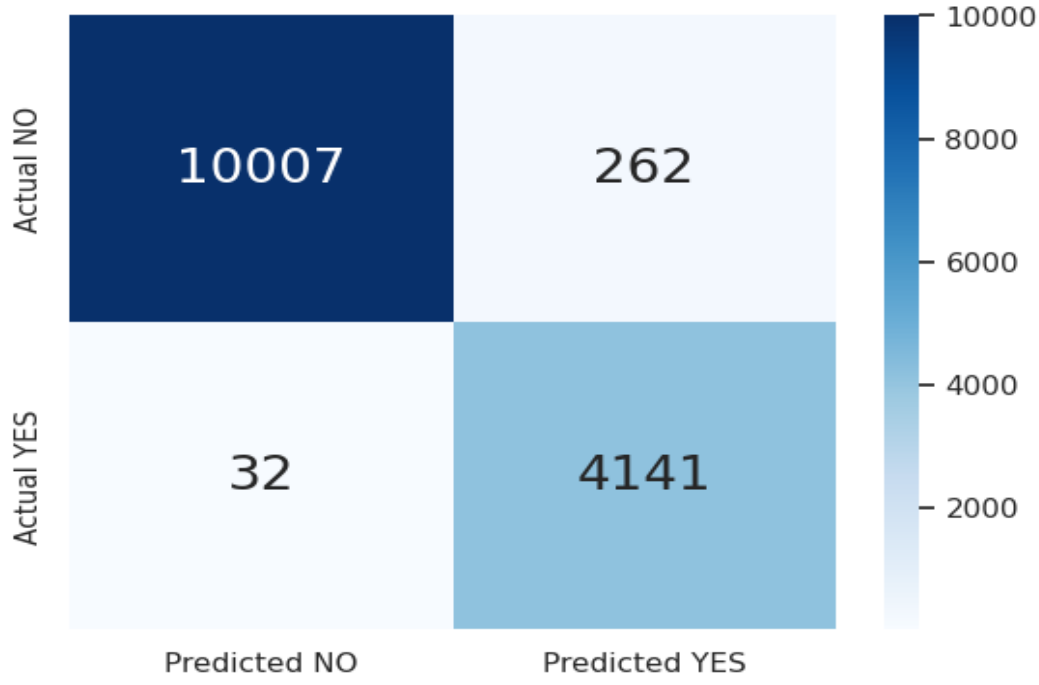
The find_best_threshold function is used to find the best threshold that maximizes the TPR*(1-FPR) value based on the training data.

The predict_with_best_t function is defined to assign the predicted states based on the best threshold.

Confusion matrices are computed for both the training and testing data using the confusion_matrix() function from sklearn.metrics.

The confusion matrices are visualized as heatmaps using the seaborn library.


ERROR PLOTS
True Positive Rate(TPR)
False Positive Rate(FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Train AUC =0.9967051425683385
Test AUC =0.9911871415607136


Actual NO
Actual YES
10007
262
32
4141
Predicted NO
Predicted YES
10000
8000
6000
4000
2000

3. Gradient Boosting Decision Tree Classifier:

Gradient Boosting Classifier and Parameter Grid:
The Gradient Boosting Classifier from the sklearn.ensemble module is imported.
Two hyperparameters, max_depth and learning_rate, are specified with different values to be evaluated.
The Gradient Boosting Classifier is initialized.
The parameter grid is defined with the values for max_depth and learning_rate.

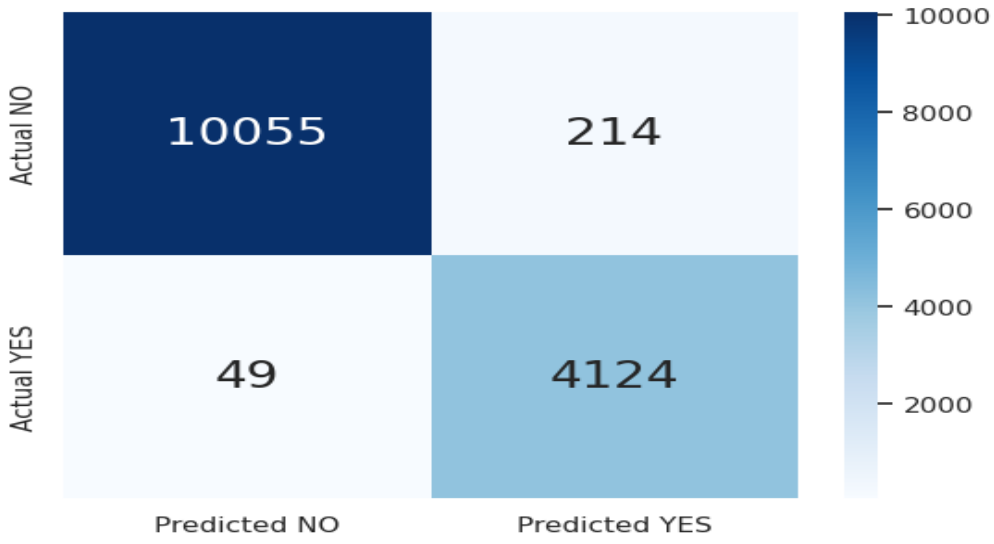
Grid Search Cross-Validation:
GridSearchCV is used for performing grid search with cross-validation to find the best hyperparameters.
GridSearchCV is initialized with the Gradient Boosting Classifier and the parameter grid.
The number of cross-validation folds is set to 3.
The scoring metric is roc_auc, which measures the area under the ROC curve.
GridSearchCV is executed to find the best hyperparameters based on the highest mean_test_score.

Visualization:
A heatmap is created to visualize the mean_train_score and mean_test_score for each combination of hyperparameters.
The mean_train_score and mean_test_score are grouped and displayed in separate subplots.
The values in the heatmap represent the AUC scores, and annotations are added to provide more information.

Model Evaluation:
The best estimator obtained from the grid search, gbdt_best, is fitted to the training data.
Using the batch_predict function, the predicted probabilities are obtained for both the training data (y_train_pred) and the testing data (y_test_pred).
The root mean squared error (rms) and the R-squared score are calculated to evaluate the performance of the model.
The predicted and actual states are stored in the DataFrame, xxx, for the testing data.

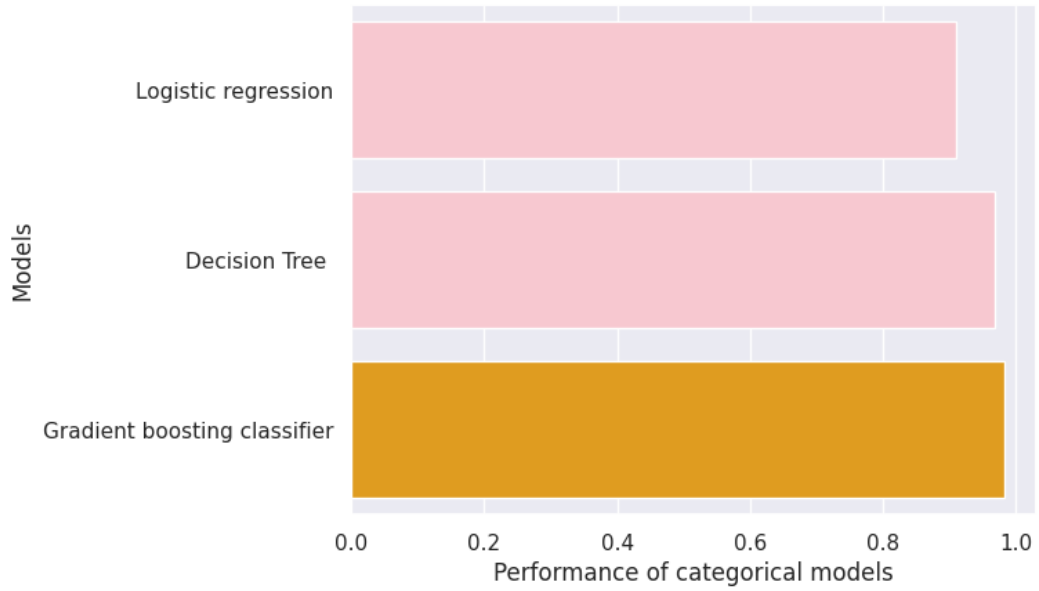
ROC Curve and Confusion Matrix:
The ROC curve is plotted using the true positive rate (TPR) and false positive rate (FPR) values obtained from the training and testing data.
The area under the ROC curve (AUC) is calculated and displayed on the plot.
The find_best_threshold function is used to find the best threshold that maximizes the TPR*(1-FPR) value based on the training data.

The predict_with_best_t function is defined to assign the predicted states based on the best threshold.
Confusion matrices are computed for both the training and testing data using the confusion_matrix() function from sklearn.metrics.
The confusion matrices are visualized as heatmaps using the seaborn library.



Conclusion

Based on the results obtained from the evaluation of the three models the gradient boosting decision classifier achieved the highest accuracy of 98.17%, followed by the decision tree classifier with an accuracy of 97.96%, and the logistic regression model with an accuracy of 92.03%.



Concluding from these results, we can say that the gradient boosting decision classifier outperformed both the decision tree classifier and logistic regression in terms of accuracy for predicting Kickstarter campaign success rates. It achieved the highest accuracy, indicating that it is more effective in distinguishing between successful and failed campaigns based on the selected features.

The decision tree classifier also performed well, with a high accuracy of 97.96%. Although slightly lower than the gradient boosting decision classifier, it still demonstrated strong predictive power. Decision trees are known for their ability to capture complex interactions and non-linear relationships in the data, making them suitable for this task.

On the other hand, the logistic regression model achieved the lowest accuracy of 92.03%. While still providing some predictive power, its performance was comparatively weaker than the decision tree and gradient boosting models. Logistic regression may have limitations in capturing non-linear relationships between features and the target variable, which could have affected its predictive performance in this particular context.

Appendix:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#load Library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")



# # Data Pre-processing

# In[2]:


import pandas as pd
df = pd.read_csv(r'C:\Users\Checkout\Downloads\kickstarter_data_with_features.csv')

df.head()



# In[3]:


df.shape


# In[4]:


df.info()


# In[5]:
```

```
df.describe()

# In[6]:

df.nunique()

# In[7]:

df.isnull().sum()

# In[8]:

df.fillna(0, inplace=True)

# In[9]:

df.drop('Unnamed: 0', axis=1, inplace=True)

# In[10]:

df.info()

# In[11]:

df.isnull().sum()

# # EDA
```

This method returns a square matrix where each cell represents the correlation between two variables in the DataFrame. The correlation coefficient can range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

In[12]:

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt



# Calculate the correlation matrix
corr_matrix = df.corr()

# Set up the figure and axes
fig, ax = plt.subplots(figsize=(50,50))

# Create the heatmap using the 'viridis' colormap
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)

# Show the plot
plt.show()
```

state

In[13]:

```
df["state"].value_counts()
```

In[14]:

```
percentile_success = round(df["state"].value_counts() / len(df["state"]) * 100,2)

print("State Percentile in %: ")
print(percentile_success)

plt.figure(figsize = (20,5))
```

```
ax1 = sns.countplot(x="state", data=df)
ax1.set_xticklabels(ax1.get_xticklabels(),rotation=45)
ax1.set_title("Status Project Distribuition", fontsize=15)
ax1.set_xlabel("State Description", fontsize=12)
ax1.set_ylabel("Count", fontsize=12)

plt.show()


# # Country

# In[15]:


percentile_success = round(df["country"].value_counts() / len(df["country"]) * 100,2)

print("country Percentile in %: ")
print(percentile_success)

top = df["country"].value_counts()
plt.figure(figsize=(10,10))
gv = sns.barplot(y = top.index, x = top.values,data = df, palette = "CMRmap")
plt.title("Status Project Distribuition",fontsize = 19)
plt.show()


# # Category

# In[16]:


df["category"].value_counts()


# In[17]:


percentile_success = round(df["category"].value_counts() / len(df["category"]) * 100,2)

print("category Percentile in %: ")
print(percentile_success)

top = df["category"].value_counts()
plt.figure(figsize=(10,50))
```

```
gv = sns.barplot(y = top.index, x = top.values,data = df, palette = "CMRmap")
plt.title("Status Project Distribuition",fontsize = 19)
plt.show()



# # Currency

# In[18]:


df["currency"].value_counts()


# In[19]:


percentile_success = round(df["currency"].value_counts() / len(df["currency"]) * 100,2)

print("currency Percentile in %: ")
print(percentile_success)

top = df["currency"].value_counts()
plt.figure(figsize=(10,5))
gv = sns.barplot(y = top.index, x = top.values,data = df, palette = "CMRmap")
plt.title("Status Project Distribuition",fontsize = 19)
plt.show()



# In[20]:


# Normalization to understand the distribuition of the pledge

df["pledge_log"] = np.log(df["pledged"] + 1)
df["goal_log"] = np.log(df["goal"]+ 1)

df_failed = df[df["state"] == "failed"]
df_success = df[df["state"] == "successful"]
df_suspended = df[df["state"] == "suspended"]

plt.figure(figsize = (14,6))
plt.subplot(221)
```

```
g = sns.distplot(df["pledge_log"])
g.set_title("Pledged Log", fontsize=18)

plt.subplot(222)
g1 = sns.distplot(df["goal_log"])
g1.set_title("goal Log", fontsize=18)

plt.subplot(212)
g2 = sns.distplot(df_failed['goal_log'], color='r')
g2 = sns.distplot(df_success['goal_log'], color='b')
g2.set_title("Pledged x Goal cross distribuition", fontsize=18)

plt.subplots_adjust(wspace = 0.1, hspace = 0.4,top = 0.9)

plt.show()



# In[ ]:




# # Description of Goal and Pledged values

# In[21]:



print("Min Goal and Pledged values")
print(df[["goal", "pledged"]].min())
print("")
print("Mean Goal and Pledged values")
print(round(df[["goal", "pledged"]].mean(),2))
print("")
print("Median Goal and Pledged values")
print(df[["goal", "pledged"]].median())
print("")
print("Max Goal and Pledged values")
print(df[["goal", "pledged"]].max())
print("")
print("Std Goal and Pledged values")
print(round(df[["goal", "pledged"]].std(),2))
```

Understanding of Goal and pleged by its State
#

In[22]:


```
plt.figure(figsize = (12,8))
plt.subplots_adjust(hspace = 0.75, top = 0.75)

ax1 = plt.subplot(221)
ax1 = sns.boxplot(x="state", y="pledge_log", data=df, palette="hls")
ax1.set_xticklabels(ax1.get_xticklabels(),rotation=45)
ax1.set_title("Understanding the Pledged values by state", fontsize=15)
ax1.set_xlabel("State Description", fontsize=12)
ax1.set_ylabel("Pledged Values(log)", fontsize=12)

ax2 = plt.subplot(222)
ax2 = sns.boxplot(x="state", y="goal_log", data=df)
ax2.set_xticklabels(ax2.get_xticklabels(),rotation=45)
ax2.set_title("Understanding the Goal values by state", fontsize=15)
ax2.set_xlabel("State Description", fontsize=12)
ax2.set_ylabel("Goal Values(log)", fontsize=12)

ax0 = plt.subplot(212)
ax0 = sns.regplot(x="goal_log", y="pledge_log", data=df, x_jitter=False)
ax0.set_title("Better view of Goal x Pledged values", fontsize=15)
ax0.set_xlabel("Goal Values(log)")
ax0.set_ylabel("Pledged Values(log)")
ax0.set_xticklabels(ax0.get_xticklabels(),rotation=90)
plt.show()
```

In[ ]:

In[ ]:

How many category Fail and Succeed?
#

In[23]:


```
main_cats = df["category"].value_counts()
main_cats_failed = df[df["state"] == "failed"]["category"].value_counts()
main_cats_sucess = df[df["state"] == "successful"]["category"].value_counts()

plt.figure(figsize = (12,8))
plt.subplots_adjust(hspace = 0.9, top = 0.75)

ax0 = plt.subplot(221)
ax0 = sns.barplot(x=main_cats_failed.index, y= main_cats_failed.values, orient='v')
ax0.set_xticklabels(ax0.get_xticklabels(),rotation=90)
ax0.set_title("Frequency Failed by  Category", fontsize=15)
ax0.set_xlabel(" Category Failed", fontsize=12)
ax0.set_ylabel("Count", fontsize=12)

ax1 = plt.subplot(222)
ax1 = sns.barplot(x=main_cats_sucess.index, y = main_cats_sucess.values, orient='v')
ax1.set_xticklabels(ax1.get_xticklabels(),rotation=90)
ax1.set_title("Frequency Successful by  Category", fontsize=15)
ax1.set_xlabel(" Category Sucessful", fontsize=12)
ax1.set_ylabel("Count", fontsize=12)

ax2 = plt.subplot(212)
ax2 = sns.boxplot(x="category", y="goal_log", data=df)
ax2.set_xticks(ax1.get_xticks())
ax2.set_xticklabels(ax1.get_xticklabels(), rotation=90)
#ax2.set_xticklabels(ax1.get_xticklabels(),rotation=90)
ax2.set_title("Distribuition goal(log) by General Category", fontsize=15)
ax2.set_xlabel("Category", fontsize=12)
ax2.set_ylabel("Goal(log)", fontsize=8)
plt.show()
```

Look Goal and Pledged Means by State
#

In[24]:

```
print("Looking Goal and Pledged Mean by state ")
print(round(df.groupby(["state"])["goal", "pledged"].mean(),2))


# # Distribuition in category values as a success or failure
#

# In[ ]:




# In[ ]:




# In[25]:


categorys_failed = df[df["state"] == "failed"]["category"].value_counts()[:25]
categorys_sucessful = df[df["state"] == "successful"]["category"].value_counts()[:25]

fig, ax = plt.subplots(ncols=2, figsize=(15,20))
plt.subplots_adjust(wspace = 0.35, top = 0.5)

g1 = plt.subplot(222)
g1 = sns.barplot(x= categorys_failed.values, y=categorys_failed.index, orient='h')
g1.set_title("Failed Category's", fontsize=15)
g1.set_xlabel("Count ", fontsize=12)
g1.set_ylabel("Category's Failed", fontsize=12)

g2 = plt.subplot(221)
g2 = sns.barplot(x= categorys_sucessful.values, y=categorys_sucessful.index, orient='h')
g2.set_title("Sucessful Category's", fontsize=15)
g2.set_xlabel("Count ", fontsize=12)
g2.set_ylabel("Category's Successful", fontsize=12)

ax2 = plt.subplot(212)
ax2 = sns.countplot(x="category", data=df)
```

```
ax2.set_xticks(ax2.get_xticks())
ax2.set_xticklabels(ax2.get_xticklabels(),rotation=90)
ax2.set_title("General Category's", fontsize=15)
ax2.set_xlabel("Category", fontsize=12)
ax2.set_ylabel("Count", fontsize=12)

plt.show()


# # Date & Time

# In[26]:



df['launched_at'] = pd.to_datetime(df['launched_at'])
df['laun_month_year'] = df['launched_at'].dt.to_period("M")
df['laun_year'] = df['launched_at'].dt.to_period("A")

df['deadline'] = pd.to_datetime(df['deadline'])
df['dead_month_year'] = df['deadline'].dt.to_period("M")
df['dead_year'] = df['deadline'].dt.to_period("A")



# In[27]:


df.laun_month_year = df.laun_month_year.dt.strftime('%Y-%m')
df.laun_year = df.laun_year.dt.strftime('%Y')



# In[ ]:




# In[28]:


year = df['laun_year'].value_counts()
month = df['laun_month_year'].value_counts()
```

```
fig, ax = plt.subplots(2,1, figsize=(12,10))

ax1 = sns.boxplot(x="laun_year", y='pledge_log', data=df, ax=ax[0])
ax1.set_title("Project Pledged by Year", fontsize=15)
ax1.set_xlabel("Years", fontsize=12)
ax1.set_ylabel("Pledged(log)", fontsize=12)

ax2 = sns.countplot(x="laun_year", hue='state', data=df, ax=ax[1])
ax2.set_title("Projects count by Year", fontsize=18)
ax2.set_xlabel("State columns by Year", fontsize=15)
ax2.set_ylabel("Count", fontsize=15)

plt.show()

print("Descriptive status count by year")
print(pd.crosstab(df.laun_year, df.state))



# In[ ]:




# # Creating a new feature to calculate percentage of pledged / goal
#

# In[29]:


df['diff_pleded_goal'] = round(df['pledge_log'] / df['goal_log'] * 100,2)
df['diff_pleded_goal'] = df['diff_pleded_goal'].astype(float)



# In[30]:


plt.figure(figsize = (12,6))
sns.distplot(df[(df['diff_pleded_goal'] < 200) & (df['state'] == 'failed')]['diff_pleded_goal'], color='r')
sns.distplot(df[(df['diff_pleded_goal'] < 200) & (df['state'] == 'successful')]['diff_pleded_goal'],color='g')
plt.show()
```

In[31]:

```
plt.figure(figsize = (18,15))

plt.subplots_adjust(hspace = 0.35, top = 0.8)

g1 = plt.subplot(211)
g1 = sns.countplot(x="laun_month_year", data=df[df['laun_month_year'] >= '2010-01'])
g1.set_xticklabels(g1.get_xticklabels(),rotation=90)
g1.set_title("Value Distribuition by Date Distribuition", fontsize=30)
g1.set_xlabel("Date Distribuition", fontsize=20)
g1.set_ylabel("Count", fontsize=20)

g2 = plt.subplot(212)
g2 = sns.boxplot(x="laun_year", y="diff_pleded_goal",data=df[df['diff_pleded_goal'] < 150],
hue="state")
g2.set_xticklabels(g2.get_xticklabels(),rotation=90)
g2.set_title("Value Distribuition by Date Distribuition", fontsize=20)
g2.set_xlabel("Date Distribuition", fontsize=20)
g2.set_ylabel("Goal x Pledged (%)", fontsize=20)
plt.show()
```

In[32]:

```
df['backers_log'] = np.log(df['backers_count'] + 1 )
#The + 1 is to normalize the zero or negative values

plt.figure(figsize = (8,6))
sns.distplot(df['backers_log'])

plt.show()
```

In[33]:

```
plt.figure(figsize = (12,8))
```

```
plt.subplot(211)
g = sns.boxplot(x='state',y='backers_log', data=df)
g.set_title("Backers by STATE", fontsize=18)

plt.subplot(212)
g = sns.boxplot(x='category',y='backers_log', data=df)
g.set_xticklabels(g.get_xticklabels(),rotation=45)

plt.show()


# In[34]:


plt.figure(figsize = (12,8))

plt.subplot(211)
g = sns.boxplot(x='laun_year',y='backers_log',data=df)
g.set_title("Backers by STATE", fontsize=18)

plt.show()



# In[35]:


#Looking the relation of Backers and % of goal reached
sns.lmplot(x='diff_pleded_goal', y ='backers_log', data=df[df['diff_pleded_goal'] < 150], aspect =
2,hue='state')
plt.show()



# In[36]:


df.head()


# In[37]:


df.info()
```

In[38]:

```
df['Dead_Dates'] = pd.to_datetime(df['deadline']).dt.date
df['Dead_Time'] = pd.to_datetime(df['deadline']).dt.time
df['dead_hour'] = pd.to_datetime(df['deadline']).dt.hour
df['dead_minute'] = pd.to_datetime(df['deadline']).dt.minute
df['dead_day'] = pd.to_datetime(df['deadline']).dt.day
df['dead_month'] = pd.to_datetime(df['deadline']).dt.month
df['dead_year'] = pd.to_datetime(df['deadline']).dt.year
```

In[39]:

```
df['launch_Dates'] = pd.to_datetime(df['launched_at']).dt.date
df['launch_Time'] = pd.to_datetime(df['launched_at']).dt.time
df['launch_hour'] = pd.to_datetime(df['launched_at']).dt.hour
df['launch_minute'] = pd.to_datetime(df['launched_at']).dt.minute
df['launch_day'] = pd.to_datetime(df['launched_at']).dt.day
df['launch_month'] = pd.to_datetime(df['launched_at']).dt.month
df['launch_year'] = pd.to_datetime(df['launched_at']).dt.year
```

In[40]:

```
df['deadline'] = df['deadline'].apply(pd.to_datetime)
df['launched'] = df['launched_at'].apply(pd.to_datetime)

df['duration'] = df['deadline'] - df['launched_at']
df['duration'] = df['duration'].dt.days
```

In[41]:

```
df['duration'] = df['duration']
```

In[42]:

```
df.drop(columns=["id","photo","blurb","slug","disable_communication","currency_symbol","creator","location","profile","spotlight","urls","source_url","friends","is_starred","is_backing","permissions","deadline","launched","pledge_log","goal_log","laun_month_year","laun_year","dead_month_year","diff_pleded_goal","backers_log","Dead_Dates","Dead_Time","launch_Dates","launch_Time"],inplace=True)
```

In[43]:

```
df.head()
```

In[44]:

```
df=df.drop(['currency_trailing_code'],axis=1)
```

In[45]:

```
df=df.drop(['staff_pick'],axis=1)
```

In[46]:

```
df=df.drop(['state_changed_at','created_at','launched_at'],axis=1)
```

In[47]:

```
df=df.drop(['deadline_weekday','state_changed_at_weekday','created_at_weekday','launched_at_weekday'],axis=1)
```

In[48]:

```
df=df.drop(['create_to_launch','launch_to_deadline','launch_to_state_change'],axis=1)

# In[49]:

df.head()

# In[50]:

df.info()

# In[51]:

df.describe()

# In[52]:

df.info()

# # Text Preprocessing
#
# In[53]:

get_ipython().system('pip install matplotlib-venn')

# In[54]:

get_ipython().system('pip install stop-words')

# In[55]:
```

```
pip install wordcloud


# In[56]:


import nltk
nltk.download('stopwords')
import nltk.tokenize as word_tokenize
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import re
from nltk.stem.porter import *
from nltk.tokenize import sent_tokenize
from stop_words import get_stop_words


# In[57]:


from wordcloud import WordCloud, STOPWORDS

stopwords = set(STOPWORDS)

wordcloud = WordCloud(
    background_color='white',
    stopwords=stopwords,
    max_words=500,
    max_font_size=200,
    width=1000, height=800,
    random_state=42,
).generate(" ".join(df['name'].dropna().astype(str)))

print(wordcloud)
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

In[58]:

```
import nltk
nltk.download('punkt')
```

In[59]:

```
# convert sentance to word contain special characters.
def sentance_to_word(x):
    x=re.sub("[^A-Za-z0-9]"," ",x)
    words=nltk.word_tokenize(x)
    return words

# convert whole eassy to word list
def essay_to_word(essay):
    essay = essay.strip()
    tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
    raw = tokenizer.tokenize(essay)
    final_words=[]
    for i in raw:
        if(len(i)>0):
            final_words.append(sentance_to_word(i))
    return final_words

# Calculate number of words in essay
def number_Of_Words(essay):
    count=0
    for i in essay_to_word(essay):
        count+=len(i)
    return count

# calculate number of character in essay
def number_Of_Char(essay):
    count=0
    for i in essay_to_word(essay):
        for j in i:
            count+=len(j)
    return count

# calculate average of words in essay
```

```
def avg_word_len(essay):
    return number_Of_Char(essay)/number_Of_Words(essay)


# In[60]:


# update dataset by calculating char_count, word_count, avg_word_len
df = df.copy()
df['name_char_count'] = df['name'].apply(number_Of_Char)
df['name_word_count'] = df['name'].apply(number_Of_Words)
df['name_avg_word_len'] = df['name'].apply(avg_word_len)
df.head()


# # Encoding
#

# In[61]:


df= df.loc[df['state'].isin(['failed','successful','canceled','suspended','live'])]


# In[62]:


target = {'failed': 0, 'successful': 1 , 'canceled': 0, 'live':0,'suspended':0 }
df['state'] = df['state'].map(target)



# In[63]:


df['state'].value_counts()


# In[64]:


#Mapping countries and replacing 'N,0"' according to currency
country = {'USD':'US', 'AUD':'AU', 'CAD':'CA', 'GBP':'GB', 'EUR':'DE', 'SEK':'SE', 'DKK':'DK',
'NZD':'NZ', 'NOK':'NO', 'CHF':'CH'}
```

```
invalid = df[df['country'] == 'N,0"']
invalid['country'] = invalid['currency'].map(country)

#Placing it in original data
invalid_country = invalid['country'].iloc[:].values
j=0
for i in invalid.index:
   df['country'].iloc[i] =invalid_country[j]
   j=j+1


# In[65]:


df['category'] = df['category'].astype(str)
df['currency'] = df['currency'].astype(str)
df['country'] = df['country'].astype(str)


# In[66]:


from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()
df['category'] = labelencoder.fit_transform(df['category'])
df['currency'] = labelencoder.fit_transform(df['currency'])
df['country'] = labelencoder.fit_transform(df['country'])


# In[67]:


df['category'] = df['category'].astype(float)
df['currency'] = df['currency'].astype(float)
df['country'] = df['country'].astype(float)



# In[68]:


df.head(1)
```

In[69]:

df.drop(columns=['name'],inplace=True)

In[70]:

df.head(1)

In[71]:

df.info()

Model Selection and Evaluation

In[72]:

df.to_csv("final_preprocessed.csv")

In[73]:

print(df.info())

In[74]:

print(list(df.columns))

In[ ]:

In[75]:

```
y = df['state'].values
X = df.drop(['state'],axis=1)
```

In[76]:

```
# Remove missing values from both X and y
X = X.dropna()
y = y[X.index]
```

In[77]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.30)
```

In[78]:

```
df.head(1)
```

In[79]:

```
df.info()
```

Logistic Regression

In[80]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

```
logistic_reg = LogisticRegression(penalty="l2")
logistic_reg.fit(X_train,y_train)



# In[81]:


y_train_pred = logistic_reg.predict(X_train)
y_test_pred = logistic_reg.predict(X_test)

d = pd.DataFrame({'Actual state': y_test, 'Predicted state': y_test_pred})
d.head(5)



# In[82]:


from sklearn.metrics import roc_curve,auc

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC ="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC ="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))

plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()



# In[83]:


def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t
```

```
def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions



# In[84]:


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfidf)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))



# In[85]:


confusion_matrix_train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
y_true = ['NO','YES']
y_pred = ['NO','YES']
confusion_matrix_train = pd.DataFrame(confusion_matrix_train, columns=np.unique(y_true), index =
np.unique(y_true))
confusion_matrix_train.index = ['Actual NO', 'Actual YES']
confusion_matrix_train.columns = ['Predicted NO','Predicted YES']
sns.heatmap(confusion_matrix_train, annot=True,annot_kws={"size": 20},fmt="d",cmap='Blues')


# In[86]:


confusion_matrix_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
y_true = ['NO','YES']
y_pred = ['NO','YES']
```

```
confusion_matrix_test = pd.DataFrame(confusion_matrix_test, columns=np.unique(y_true), index =
np.unique(y_true))
confusion_matrix_test.index = ['Actual NO', 'Actual YES']
confusion_matrix_test.columns = ['Predicted NO','Predicted YES']
sns.heatmap(confusion_matrix_test, annot=True,annot_kws={"size": 20},fmt="d",cmap='Blues')


# In[87]:


from sklearn.metrics import precision_score
precision_score_train = precision_score(y_train, predict_with_best_t(y_train_pred, best_t))
print(print("Precision_Score of Train: ",precision_score_train))



# In[88]:


from sklearn.metrics import recall_score
recall_score_train = recall_score(y_train, predict_with_best_t(y_train_pred, best_t))
print(print("Recall_Score of Train: ",recall_score_train))



# In[89]:


from sklearn.metrics import f1_score
f1_score_train  = f1_score(y_train, predict_with_best_t(y_train_pred, best_t))
print("F1_Score of Train: ",f1_score_train)


# In[90]:


from sklearn.metrics import accuracy_score
auc_score_train_lr  = accuracy_score(y_train, predict_with_best_t(y_train_pred, best_t))
print("Accuracy_Score of Train: ",auc_score_train_lr)



# In[ ]:
```

In[ ]:

In[91]:

```
from sklearn.metrics import accuracy_score
auc_score_test_lr  = accuracy_score(y_test, predict_with_best_t(y_test_pred, best_t))
print("Accuracy_Score of Test: ",auc_score_test_lr)
```

In[ ]:

Decision Tree

In[92]:

```
def batch_predict(clf, data):

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In[93]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score

min_samples_split=[5, 10, 100, 500]
max_depth=[1, 5, 10, 50]
train_auc = []
test_auc = []
for i in min_samples_split:
  for j in max_depth:
    print("min_samples_split = ", i ,'and max_depth = ',j)
    dt = DecisionTreeClassifier(min_samples_split=i, max_depth=j,class_weight='balanced')
    dt.fit(X_train, y_train)

    y_train_pred = batch_predict(dt, X_train)
    y_test_pred = batch_predict(dt, X_test)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    test_auc.append(roc_auc_score(y_test, y_test_pred))

print('='*100)
print('Train AUC : ',train_auc)
print('='*100)
print('Test AUC : ',test_auc)
print('='*100)


# In[94]:


from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

min_samples_split=[5, 10, 100, 500]
max_depth= [1, 5, 10, 50]

dt=DecisionTreeClassifier(class_weight='balanced')
params={'max_depth': [1, 5, 10], 'min_samples_split':[5, 10, 100]}
```

```
cross_val=GridSearchCV(estimator=dt, param_grid=params, cv=3,
scoring='roc_auc',return_train_score=True,verbose=2,n_jobs=-1)

cross_val.fit(X_train,y_train)
train_auc= cross_val.cv_results_['mean_train_score']
train_auc_std= cross_val.cv_results_['std_train_score']
test_auc = cross_val.cv_results_['mean_test_score']
test_auc_std= cross_val.cv_results_['std_test_score']

print('='*100)
print('The Best Parameters are : ',cross_val.best_params_)
print('='*100)



# In[95]:



sns.set()
max_scores = pd.DataFrame(cross_val.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()


# In[96]:



#
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curv
e
print(cross_val.best_params_)
dt_best = cross_val.best_estimator_
dt_best.fit(X_train, y_train)

y_train_pred = batch_predict(dt_best, X_train)
y_test_pred = batch_predict(dt_best, X_test)

from sklearn.metrics import mean_squared_error as mse , r2_score
from math import sqrt
```

```
rms = sqrt(mse(y_test, y_test_pred))
score = r2_score(y_test, y_test_pred)

print(rms)
print(score)

xxx = pd.DataFrame({'Actual state': y_test, 'Predicted state': y_test_pred})
xxx.head(5)


# In[97]:


from sklearn.metrics import roc_curve, roc_auc_score, auc

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC ="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC ="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))

plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()



# In[98]:


# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
```

```
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In[99]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfidf)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

In[100]:

```
confusion_matrix_train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
y_true = ['NO','YES']
y_pred = ['NO','YES']
confusion_matrix_train = pd.DataFrame(confusion_matrix_train, columns=np.unique(y_true), index =
np.unique(y_true))
confusion_matrix_train.index = ['Actual NO', 'Actual YES']
confusion_matrix_train.columns = ['Predicted NO','Predicted YES']
sns.heatmap(confusion_matrix_train, annot=True,annot_kws={"size": 20},fmt="d",cmap='Blues')
```

In[101]:

```
confusion_matrix_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
y_true = ['NO','YES']
y_pred = ['NO','YES']
confusion_matrix_test = pd.DataFrame(confusion_matrix_test, columns=np.unique(y_true), index =
np.unique(y_true))
confusion_matrix_test.index = ['Actual NO', 'Actual YES']
```

```
confusion_matrix_test.columns = ['Predicted NO','Predicted YES']
sns.heatmap(confusion_matrix_test, annot=True,annot_kws={"size": 20},fmt="d",cmap='Blues')
```

In[102]:

```
from sklearn.metrics import precision_score
precision_score_train = precision_score(y_train, predict_with_best_t(y_train_pred, best_t))
print(print("Precision_Score of Train: ",precision_score_train))
```

In[103]:

```
from sklearn.metrics import recall_score
recall_score_train = recall_score(y_train, predict_with_best_t(y_train_pred, best_t))
print(print("Recall_Score of Train: ",recall_score_train))
```

In[104]:

```
from sklearn.metrics import f1_score
f1_score_train  = f1_score(y_train, predict_with_best_t(y_train_pred, best_t))
print("F1_Score of Train: ",f1_score_train)
```

In[105]:

```
from sklearn.metrics import accuracy_score
auc_score_train_dt  = accuracy_score(y_train, predict_with_best_t(y_train_pred, best_t))
print("Accuracy_Score of Train: ",auc_score_train_dt)
```

In[ ]:

In[106]:

```
from sklearn.metrics import accuracy_score
auc_score_test_dt  = accuracy_score(y_test, predict_with_best_t(y_test_pred, best_t))
print("Accuracy_Score of Test: ",auc_score_test_dt)
```

In[ ]:

In[ ]:

Gradient Boosting Decision Tree
#

In[107]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

max_depth= [1, 2, 3]
learning_rate=[0.001,0.01,1]

gbdt=GradientBoostingClassifier()
params={'max_depth': [1, 2, 3], 'learning_rate':[0.001,0.01,1]}

grid=GridSearchCV(estimator=gbdt, param_grid=params, cv=3,
scoring='roc_auc',return_train_score=True,verbose=2,n_jobs=-1)

grid.fit(X_train,y_train)
train_auc= grid.cv_results_['mean_train_score']
train_auc_std= grid.cv_results_['std_train_score']
test_auc = grid.cv_results_['mean_test_score']
test_auc_std= grid.cv_results_['std_test_score']
```

```
print('='*100)
print('The Best Parameters are : ',grid.best_params_)
print('='*100)


# In[108]:


sns.set()
max_scores = pd.DataFrame(grid.cv_results_).groupby(['param_max_depth',
'param_learning_rate']).max().unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()



# In[109]:


gbdt_best = grid.best_estimator_
gbdt_best.fit(X_train, y_train)

y_train_pred = batch_predict(gbdt_best, X_train)
y_test_pred = batch_predict(gbdt_best, X_test)

from sklearn.metrics import mean_squared_error as mse , r2_score
from math import sqrt
rms = sqrt(mse(y_test, y_test_pred))
score = r2_score(y_test, y_test_pred)

print(rms)
print(score)

xxx = pd.DataFrame({'Actual state': y_test, 'Predicted state': y_test_pred})
xxx.head(5)



# In[110]:
```

```
train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC ="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC ="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))

plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()


# In[111]:


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfidf)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))


# In[112]:


confusion_matrix_train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
y_true = ['NO','YES']
y_pred = ['NO','YES']
confusion_matrix_train = pd.DataFrame(confusion_matrix_train, columns=np.unique(y_true), index =
np.unique(y_true))
confusion_matrix_train.index = ['Actual NO', 'Actual YES']
confusion_matrix_train.columns = ['Predicted NO','Predicted YES']
sns.heatmap(confusion_matrix_train, annot=True,annot_kws={"size": 20},fmt="d",cmap='Blues')


# In[113]:
```

```
confusion_matrix_test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
y_true = ['NO','YES']
y_pred = ['NO','YES']
confusion_matrix_test = pd.DataFrame(confusion_matrix_test, columns=np.unique(y_true), index =
np.unique(y_true))
confusion_matrix_test.index = ['Actual NO', 'Actual YES']
confusion_matrix_test.columns = ['Predicted NO','Predicted YES']
sns.heatmap(confusion_matrix_test, annot=True,annot_kws={"size": 20},fmt="d",cmap='Blues')


# In[114]:


from sklearn.metrics import precision_score
precision_score_train = precision_score(y_train, predict_with_best_t(y_train_pred, best_t))
print(print("Precision_Score of Train: ",precision_score_train))



# In[115]:


from sklearn.metrics import recall_score
recall_score_train = recall_score(y_train, predict_with_best_t(y_train_pred, best_t))
print(print("Recall_Score of Train: ",recall_score_train))



# In[116]:


from sklearn.metrics import f1_score
f1_score_train  = f1_score(y_train, predict_with_best_t(y_train_pred, best_t))
print("F1_Score of Train: ",f1_score_train)


# In[117]:


from sklearn.metrics import accuracy_score
auc_score_train_gbdt  = accuracy_score(y_train, predict_with_best_t(y_train_pred, best_t))
print("Accuracy_Score of Train: ",auc_score_train_gbdt)
```

In[ ]:

In[118]:

```
from sklearn.metrics import accuracy_score
auc_score_test_gbdt  = accuracy_score(y_test, predict_with_best_t(y_test_pred, best_t))
print("Accuracy_Score of Test: ",auc_score_test_gbdt)
```

In[ ]:

In[119]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Define the data
my_data = pd.DataFrame({
    'cat': ['Logistic regression', 'Decision Tree ', 'Gradient boosting classifier'],
    'val': [auc_score_train_lr, auc_score_train_dt, auc_score_train_gbdt],
    'color': ['#FFC0CB', '#FFC0CB', '#FFA500']
})

# Create the barplot
ax = sns.barplot(x='val', y='cat', data=my_data, palette=my_data['color'])

# Set the axis labels
ax.set(xlabel='Performance of categorical models', ylabel='Models')

# Show the plot
plt.show()
```

References:

1.A. Géron, "Hands-On Machine Learning with Scikit-Learn and TensorFlow," 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2017.

2. R. Predum, "Kickstarter Campaign Success Prediction," GitHub. [Online]. Available: https://github.com/rileypredum/Kickstarter-Campaign-Success-Prediction/blob/master/kickstarter.ipynb. [Accessed: 15 May 2023].

3.RStudio, "Kickstarter: What Drives a Successful Campaign?" [Online]. Available: https://rstudio-pubs-static.s3.amazonaws.com/750966_5cf514a25ba74fc586f45cba9c9c2c8f.html . [Accessed:15 May 2023].

4.K. Schall et al., "Kickpredict: Predicting Kickstarter Success," in Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, Australia, 2015, pp. 1779-1788.

5.D. Grbovic et al., "Predicting Kickstarter Success with Natural Language Processing and Deep Learning," in Proceedings of the 12th ACM Conference on Recommender Systems, Vancouver, BC, Canada, 2018, pp. 280-284.

6."scikit-learn: Machine Learning in Python," [Online]. Available:https://scikit-learn.org/stable/modules/classes.html. [Accessed:15 May 2023].

7."Trends in Pricing and Duration," Kickstarter Blog. [Online]. Available: https://www.kickstarter.com/blog/trends-in-pricing-and-duration. [Accessed:15 May 2023].