1.INTRODUCTION

1...OVERVIEW

creators bring their innovative ideas to life. However, not all Kickstarter projects are successful, and understanding the factors that contribute to success or failure can be valuable for both creators and investors alike.

In this dataset, we have collected information on a large number of Kickstarter projects and whether they ultimately succeeded Kickstarter is a popular crowdfunding platform that has helped thousands of entrepreneurs and or failed to meet their funding goals. This dataset includes a wide range of project types, including technology startups, creative arts endeavors, and social impact initiatives, among others. By, analyzing this dataset, researchers and analysts can gain insights into the characteristics of successful and unsuccessful Kickstarter projects, such as funding targets, project categories, and funding sources. This information can be used to inform investment decisions and guide future crowdfunding campaigns. Overall, this dataset provides a comprehensive look at the Kickstarter ecosystem and can serve as a valuable resource for anyone interested in understanding the dynamics of crowdfunding and the factors that contribute project success or failure.

2..PURPOSE

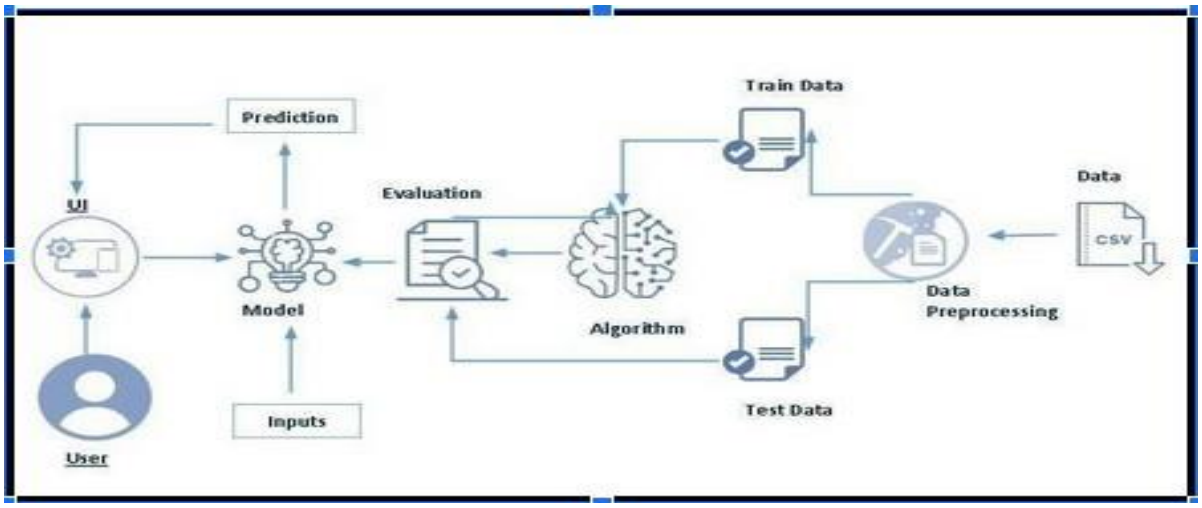
An automated kickstart campaign is not a standard or widely recognized term, so its purpose can vary depending on the context. However, I can provide information on a few possible interpretations and purposes of the term:

1. Kickstarter Campaign Automation: In the context of crowdfunding, such as Kickstarter, an
   automated kickstart campaign could refer to using automated tools or software to help manage and promote a Kickstarter campaign. This might involve automating tasks like social media posts, email updates to backers, or managing the campaign page itself. The purpose would be to streamline and optimize the campaign process.
   Kickstart for Software or Systems: In the context of technology or IT, "kickstart" often refers to
2. the automated installation and configuration process for operating systems or software.
   automated kickstart campaign might be related to automating the deployment and management of software or systems, aiming to save time and effort.

   Automated Fundraising Campaign: In a more general sense, an automated kickstart campaign

3. could refer to any fundraising campaign, be it for a charity, a political cause, or a startup, where automation tools or strategies are used to streamline the fundraising process. The purpose could be to reach a wider audience, optimize donations, or reduce administrative overhead

3..Technical Architecture



4..LITERATURE SURVEY

1...EXISTING PROBLEM

1. Over-Automation: Excessive automation can lead to a lack of personalization and authenticity. Campaigns that feel overly automated may turn off potential backers or customers.
2. Lack of Human Touch: Automated campaigns may lack the human touch, making it challenging to build personal relationships or trust with backers or customers.
3. Targeting and Segmentation: If not properly set up, automated campaigns may not effectively target the right audience or segment customers, leading to wasted resources and lower conversion rates.
4. Data Privacy and Compliance: Automated campaigns need to comply with data protection regulations. Mishandling personal data can lead to legal and reputational issues.
5. Content Quality: Relying solely on automation for content creation can result in low-quality or generic content that doesn't engage the audience effectively.
6. Technical Issues: Technical glitches or errors in automated systems can disrupt campaigns, causing delays or even data loss.

2...PROPOSED SOLUTIONS

Balance Automation with Personalization:
Solution: Customize automated messages and content to add a personal touch. Use segmentation and data analysis to create personalized experiences for different audience segments.
Human Interaction and Engagement:
Solution: Ensure that there are opportunities for human interaction, such as responding to customer inquiries or concerns promptly. Maintain an active social media presence to engage with backers or customers directly.
Improved Targeting and Segmentation:
Solution: Regularly review and update your targeting criteria. Use data analytics to refine your audience segments and ensure your messages are reaching the right people.
Data Privacy and Compliance:
Solution: Stay informed about data protection regulations, implement robust data protection measures, and seek legal advice if needed. Clearly communicate your data handling and privacy policies to your audience.
Quality Content Creation:
Solution: Balance automated content with high-quality, original content. Invest in content creation and editing to ensure your messages are engaging and well-crafted.
Technical Reliability:
Solution: Implement redundancy and monitoring systems to ensure the reliability of your automated systems. Regularly test and troubleshoot to prevent technical issues

5.SOCIAL AND BUSINESS IMPACTS

Social Impact: - Helps take informed and better decisions: By providing accurate and up-to-date information on various affecting parameters affecting the life of a Kickstarter, a classification project can help individuals make more informed decisions about weather or not they should carry forward with their project.

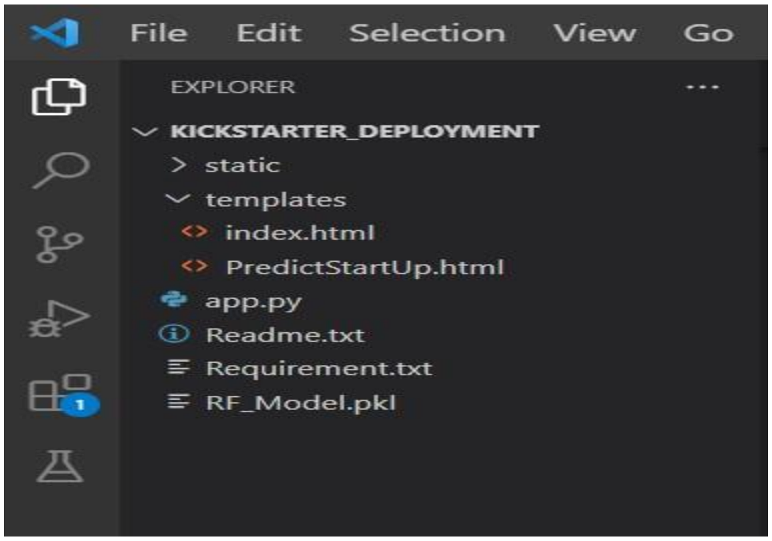
Business Model/Impact: - Emergence of new Kickstarter: By providing information on the properties and interactions of different factors such as category, goal, pledge, backers, country etc. a Kickstarter classification project can assist in the development of new horizons amongst
Kick-starters.

PROJECT FLOW :

- User interacts with the UI to enter the input.

- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI
- To accomplish this, we have to complete all the activities listed below,
- Define Problem / Problem Understanding
- Specify the business problem Data Collection & Preparation
    - Collect the dataset ○ Data Preparation
- Exploratory Data Analysis
    - Descriptive statistical
    - Visual Analysis
- Model Building
    - Training the model in multiple algorithms
    - Testing the model
- Performance Testing & Hyperparameter Tuning
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning ● Model Deployment
    - Save the best model
    - Integrate with Web Framework
- Project Demonstration & Documentation
    - Record explanation Video for project end to end solution
    - Project Documentation-Step by step project development procedure

PROJECT STRUCTURE:



```
KICKSTARTER_DEPLOYMENT
  > static
  v templates
      index.html
      PredictStartUp.html
  app.py
  Readme.txt
  Requirement.txt
  RF_Model.pkl
```

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- RF_model.pkl is our saved model. Further we will use this model for flask integration. ● Training folder contains a model training file.

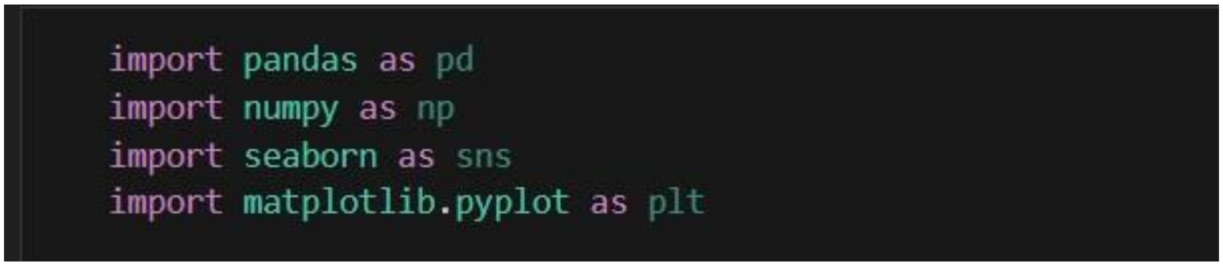
6.RESULT

DATA COLLECTION AND PREPARATION:

There are many open sources you can collect the data from

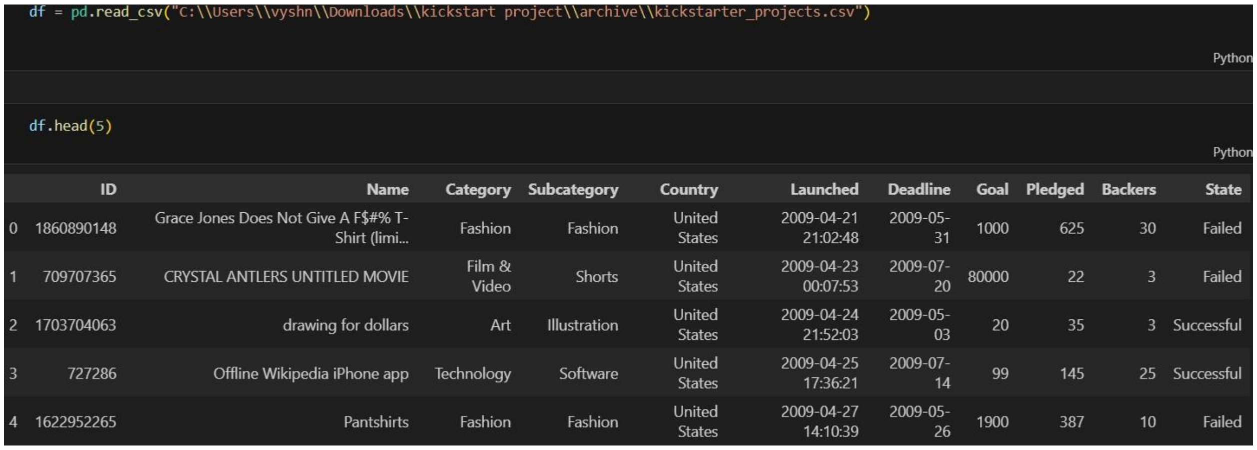
Link:https://www.kaggle.com/datasets/ulrikthygepedersen/kickstarter-projects

Case 1:importing the libraries import the nessasary libraries as show in the figure



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

Case 2:Read the dataset



```python
df = pd.read_csv("C:\\Users\\vyshn\\Downloads\\kickstart project\\archive\\kickstarter_projects.csv")
```

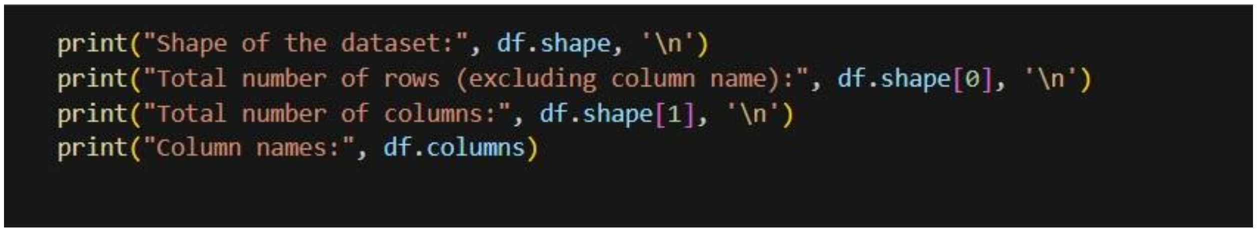
Python

```python
df.head(5)
```

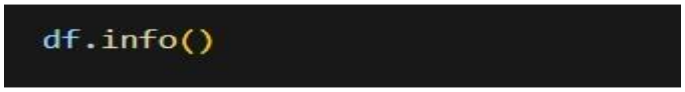
Python

	ID	Name	Category	Subcategory	Country	Launched	Deadline	Goal	Pledged	Backers	State
0	1860890148	Grace Jones Does Not Give A F$#% T-Shirt (limi...	Fashion	Fashion	United States	2009-04-21 21:02:48	2009-05-31	1000	625	30	Failed
1	709707365	CRYSTAL ANTLERS UNTITLED MOVIE	Film & Video	Shorts	United States	2009-04-23 00:07:53	2009-07-20	80000	22	3	Failed
2	1703704063	drawing for dollars	Art	Illustration	United States	2009-04-24 21:52:03	2009-05-03	20	35	3	Successful
3	727286	Offline Wikipedia iPhone app	Technology	Software	United States	2009-04-25 17:36:21	2009-07-14	99	145	25	Successful
4	1622952265	Pantshirts	Fashion	Fashion	United States	2009-04-27 14:10:39	2009-05-26	1900	387	10	Failed

Case 3:Data cleaning



```python
print("Shape of the dataset:", df.shape, '\n')
print("Total number of rows (excluding column name):", df.shape[0], '\n')
print("Total number of columns:", df.shape[1], '\n')
print("Column names:", df.columns)
```



```python
df.info()
```

EXPLORING DATA ANALYSIS

Case 4:Data preparation

Data preparation is a crucial step in the data analysis and modeling process. It involves cleaning, transforming, and organizing your raw data into a format suitable for analysis, machine learning, or other data-driven tasks. Proper data preparation is essential for ensuring the accuracy and effectiveness of your analysis or modeling efforts

1.Handling missing values (Null values), checking for zero values and finding duplicate values

2.Exploring the total number of values or data present for specific countries

3.Evaluating descriptive Statistics of Backers attribute, finding the 10 largest backer row values and visualizing the details of the highest column.

4.Creating a new column by splitting the existing column Launched and then deleting the 4 columns contributing the least towards prediction.

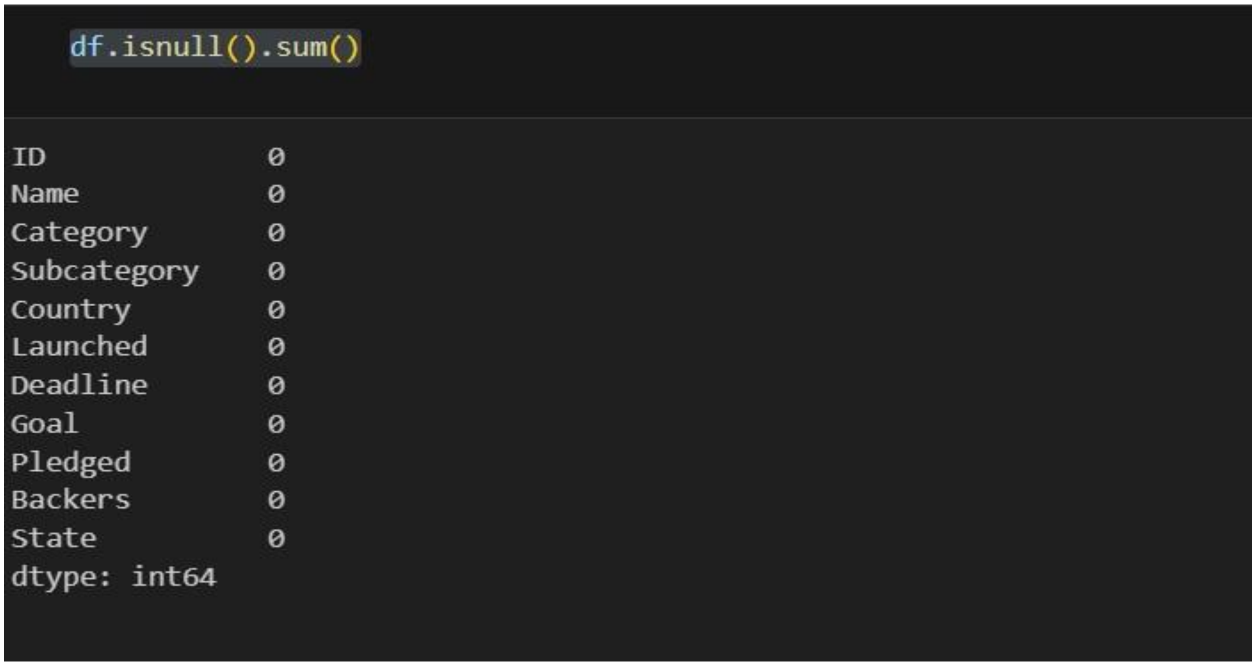
5.Exploring rows where goal amount is greater than pledged amount for successful kick-starters.

6. Finding the per person mean of individual countries.

7.Determining the correlation between independent variables

1.Handling missing values (Null values), checking for zero values and finding duplicate values Handling missing values (null values), checking for zero values, and finding duplicate values are essential data preprocessing steps to ensure the quality and reliability of your dataset.

The code df.isnull().sum() is used to identify and count missing (null) values in a DataFrame. It provides a summary of the number of missing values for each column in the DataFrame.

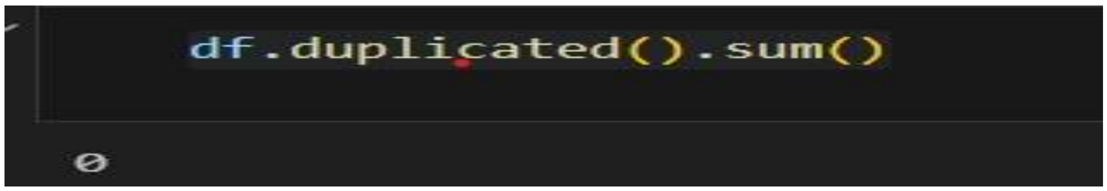


```
df.isnull().sum()
```

```
ID             0
Name           0
Category       0
Subcategory    0
Country        0
Launched       0
Deadline       0
Goal           0
Pledged        0
Backers        0
State          0
dtype: int64
```

The code df.duplicated().sum() is used to find and count duplicate rows in a DataFrame. It provides the sum of rows that are duplicates in the DataFrame. Here's an explanation of the code: df: This variable represents the DataFrame you are working with, which contains your dataset. .duplicated(): This method is applied to the DataFrame and returns a boolean Series that identifies duplicate rows. Each element in the Series is True if the corresponding row is a duplicate of a previous row and False otherwise.

.sum(): After applying .duplicated(), the .sum() method is used to calculate the sum of True values in the boolean Series. Since True is equivalent to 1 and False is equivalent to 0 when summing boolean values, this effectively counts the number of duplicate rows in the DataFrame.

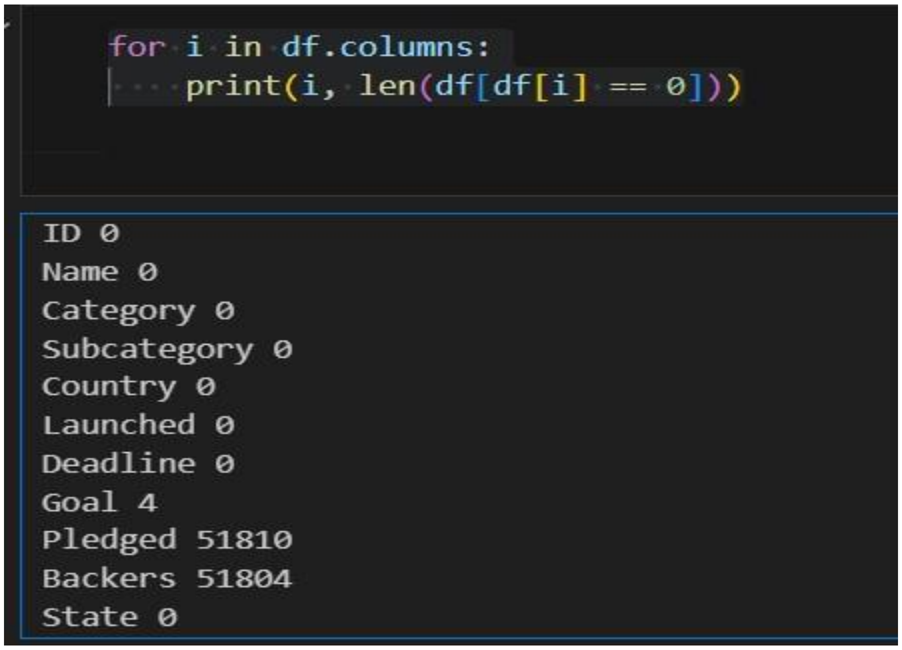


```
df.duplicated().sum()
```

```
0
```

The code snippet for i in df.columns: print(i, len(df[df[i] == 0])) is a loop that iterates through the columns of a DataFrame (df) and counts the number of occurrences where each column's value is equal to

0. Here's the breakdown of the code: for i in df.columns:: This is a loop that iterates through the column names of the DataFrame df. It is using the .columns attribute to obtain a list of column names. print(i, len(df[df[i] == 0])): Within the loop, it prints two pieces of information: i: This represents the name of the current column being examined in the loop iteration. len(df[df[i] == 0]): This is an expression that calculates the length (count) of rows where the values in the current column i are equal to 0. This effectively counts the number of occurrences of the value 0 in that column.



```
for i in df.columns:
    print(i, len(df[df[i] == 0]))
```

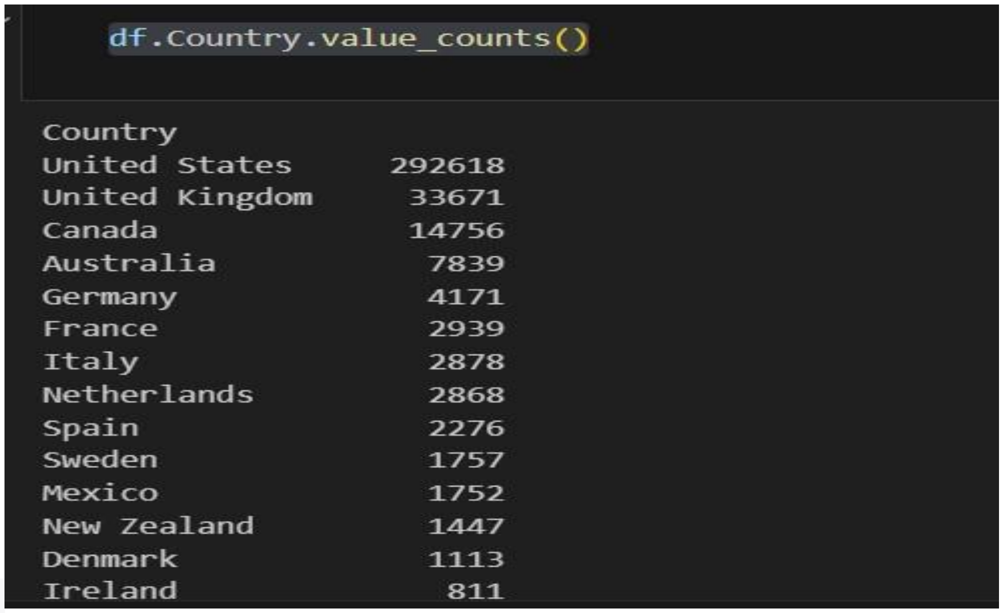
```
ID 0
Name 0
Category 0
Subcategory 0
Country 0
Launched 0
Deadline 0
Goal 4
Pledged 51810
Backers 51804
State 0
```

2.Exploring the total number of values or data present for specific countries

Exploring the total number of values or data present for specific countries typically involves counting the occurrences of each unique country in a dataset.

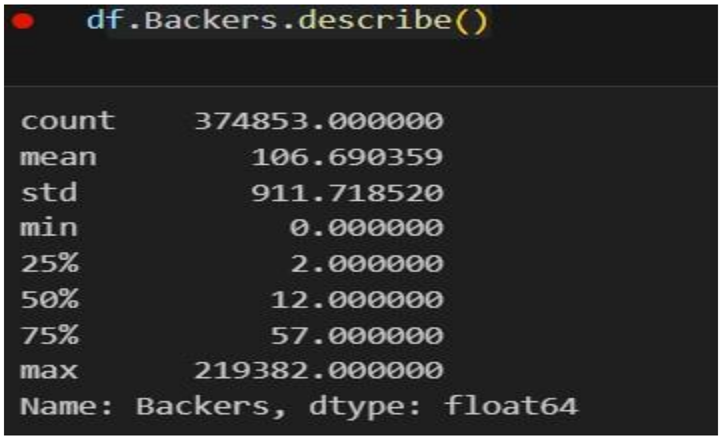
The code df.Country.value_counts() is used to count the occurrences of each unique value in the "Country" column of a DataFrame. Here's an explanation of the code: df: This variable represents the DataFrame you are working with, which contains your dataset.

.Country: This part of the code accesses the "Country" column in the DataFrame df. .value_counts(): This is a method that is applied to the "Country" column and returns a Series that counts the number of occurrences of each unique value in the column.

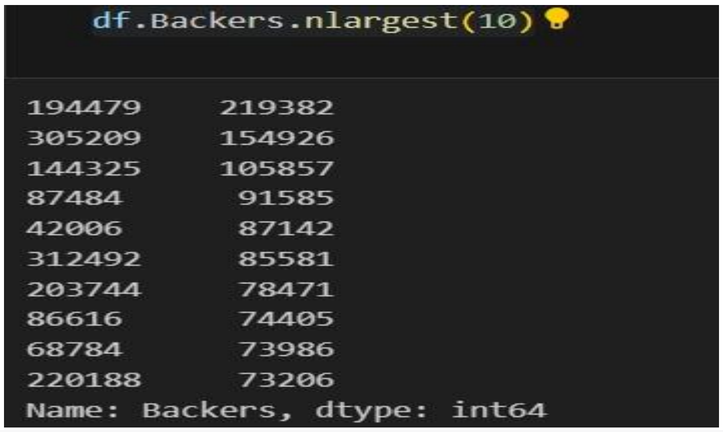

```
df.Country.value_counts()

Country
United States     292618
United Kingdom     33671
Canada             14756
Australia           7839
Germany             4171
France              2939
Italy               2878
Netherlands         2868
Spain               2276
Sweden              1757
Mexico              1752
New Zealand         1447
Denmark             1113
Ireland              811
```

3.Evaluating descriptive Statistics of Backers attribute, finding the 10 largest backer row values and visualizing the details of the highest column.

The code Backers.describe() is used to calculate and display descriptive statistics for a numerical attribute, in this case, the "Backers" attribute. Here's an explanation of the code:
Backers: This is assumed to be a column or attribute in a DataFrame, representing the number of backers for a particular project or campaign.
.describe(): This is a method in pandas that can be applied to a numerical column, and it computes various statistical summary measures for that column.


```
df.Backers.describe()

count    374853.000000
mean        106.690359
std         911.718520
min           0.000000
25%           2.000000
50%          12.000000
75%          57.000000
max      219382.000000
Name: Backers, dtype: float64
```

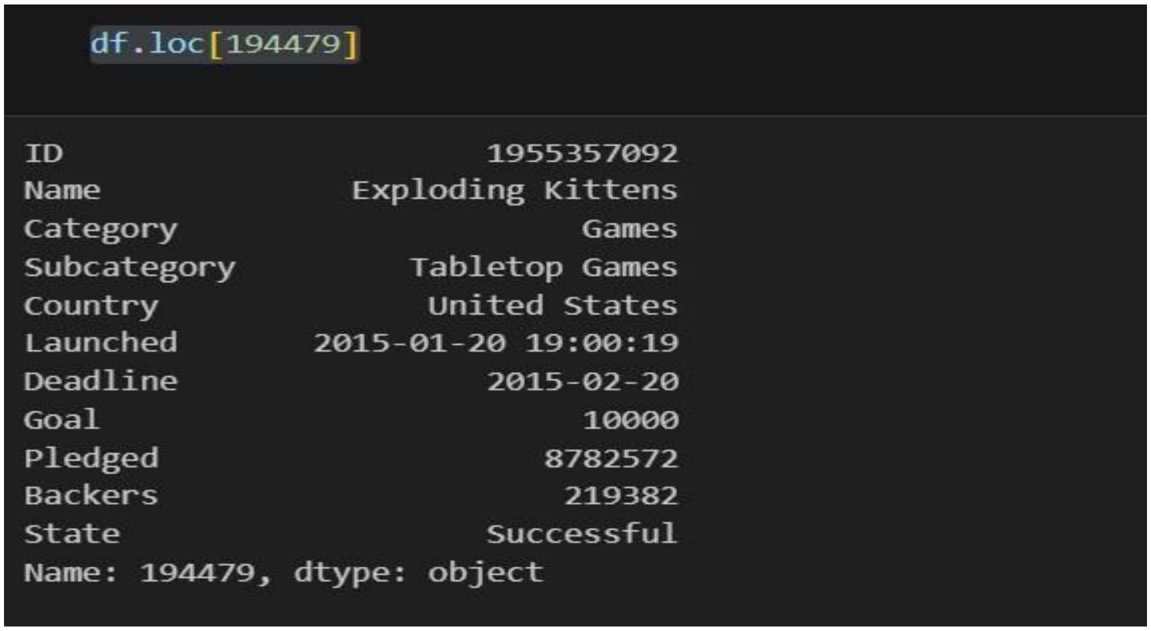
The code df.Backers.nlargest(10) is used to identify and retrieve the ten largest values from the "Backers" column in a DataFrame (df). Here's an explanation of the code: df: This variable represents the DataFrame you are working with, which contains your dataset.
.Backers: This is used to access the "Backers" column within the DataFrame.
.nlargest(10): This is a method applied to the "Backers" column. It selects the ten largest values from the column, preserving the order of the original rows in the DataFrame.
The result of df.Backers.nlargest(10) is a DataFrame containing the top ten rows from the original DataFrame (df) where the "Backers" attribute has the largest values. These rows are selected based on the magnitude of the "Backers" values, with the largest values at the top of the list.



```
df.Backers.nlargest(10)
```

```
194479    219382
305209    154926
144325    105857
87484      91585
42006      87142
312492     85581
203744     78471
86616      74405
68784      73986
220188     73206
Name: Backers, dtype: int64
```

The code df.loc[194479] is used to locate and access a specific row within a df: This variable represents the DataFrame you are working with, which contains your dataset. loc[194479]: This code is using the .loc accessor, which is used for label-based indexing. It selects and retrieves the row with the label (index) "194479" from the DataFrame.
The result of df.loc[194479] is a Series or DataFrame that represents the specific row with the label
"194479" from the original DataFrame (df). This allows you to access and examine the data associated with that particular row.



```
df.loc[194479]
```

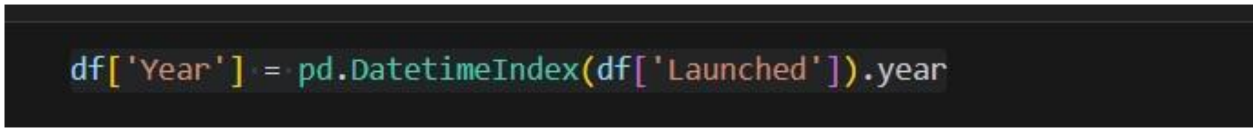
```
ID                          1955357092
Name                 Exploding Kittens
Category                         Games
Subcategory             Tabletop Games
Country                  United States
Launched           2015-01-20 19:00:19
Deadline                    2015-02-20
Goal                             10000
Pledged                        8782572
Backers                         219382
State                       Successful
Name: 194479, dtype: object
```

4.Creating a new column by splitting the existing column Launched and then deleting the 4 columns contributing the least towards prediction.

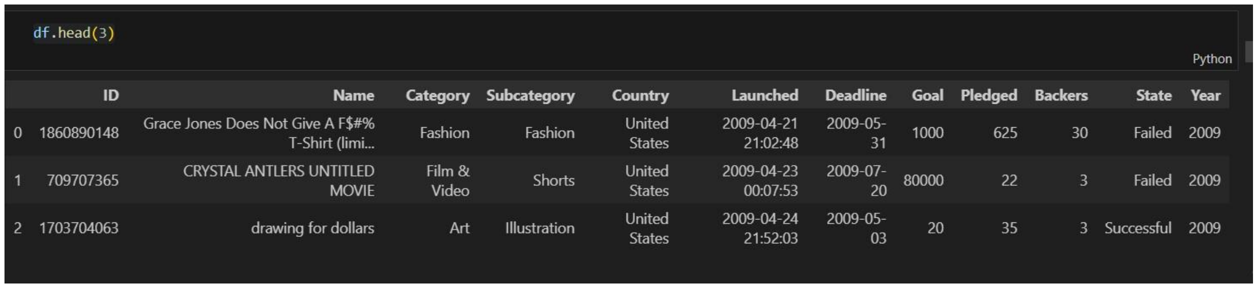
Creating a new column by splitting an existing column and then deleting the four columns contributing the least towards prediction typically involves data preprocessing and feature selection in the context of a machine learning task.

The code df['Year'] = pd.DatetimeIndex(df['Launched']).year is used to create a new column called "Year" in the DataFrame df by extracting the year component from a date column, in this case,
"Launched." Here's an explanation of the code:
df: This variable represents the DataFrame you are working with, which contains your dataset.
['Year']: This part of the code specifies the name of the new column you want to create, which will hold the extracted year values. pd.DatetimeIndex(df['Launched']): This code extracts the datetime information from the "Launched" column and converts it into a DatetimeIndex object. This allows you to access various datetime components (such as year, month, day, etc.).
.year: This is used to extract the year component from the DatetimeIndex object, and the result is a Series of year values.
The result of df['Year'] = pd.DatetimeIndex(df['Launched']).year is a DataFrame with an additional column "Year," containing the year values extracted from the "Launched" column.
This code is particularly useful when you have date or timestamp data in your DataFrame and you want to create new features based on the components of those dates. In this case, you are interested in extracting the year of the "Launched" dates, which can be valuable for time-based analysis or for creating timerelated features in your dataset.

After executing this code, you can use the "Year" column for various data analysis, visualization, and modeling tasks.



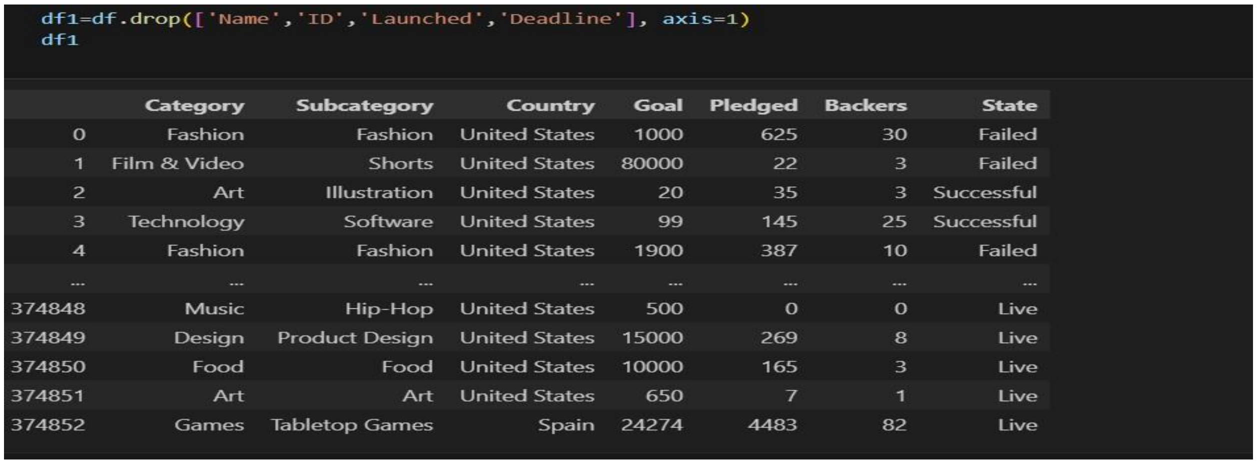
```
df['Year'] = pd.DatetimeIndex(df['Launched']).year
```



```
df.head(3)
```

```
Python
           ID                                Name  Category  Subcategory  Country    Launched   Deadline   Goal  Pledged  Backers       State  Year
0  1860890148  Grace Jones Does Not Give A F$#%     Fashion      Fashion   United  2009-04-21   2009-05-   1000      625       30      Failed  2009
                                T-Shirt (limi...                           States    21:02:48         31
1   709707365         CRYSTAL ANTLERS UNTITLED     Film &       Shorts   United  2009-04-23   2009-07-  80000       22        3      Failed  2009
                                           MOVIE     Video                 States    00:07:53         20
2  1703704063                 drawing for dollars       Art  Illustration  United  2009-04-24   2009-05-     20       35        3  Successful  2009
                                                                           States    21:52:03         03
```

The code df1 = df.drop(['Name', 'ID', 'Launched', 'Deadline'], axis=1) is used to create a new DataFrame (



```
df1=df.drop(['Name','ID','Launched','Deadline'], axis=1)
df1
```

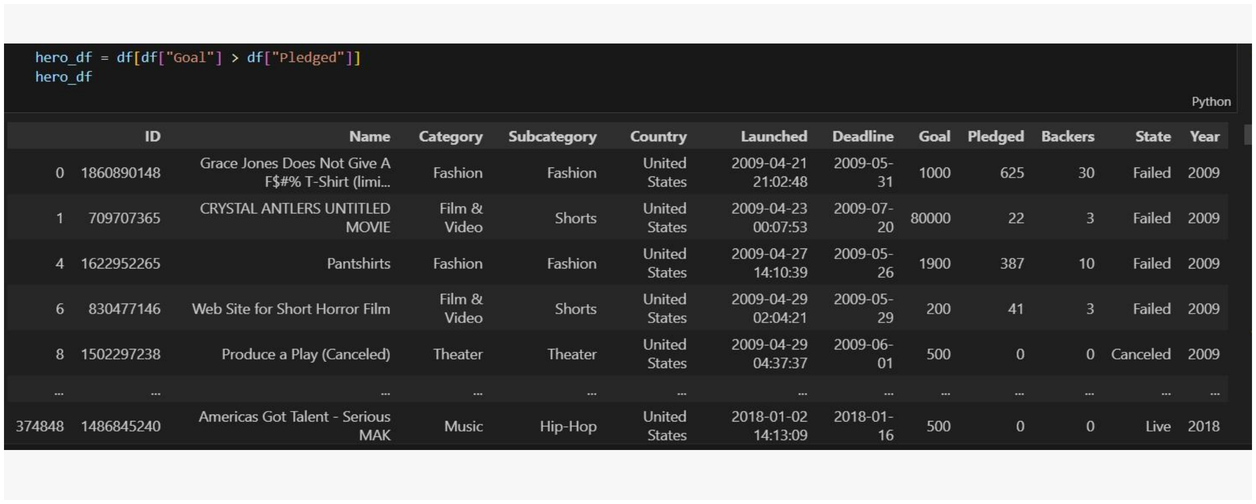
```
            Category       Subcategory        Country   Goal  Pledged  Backers       State
     0       Fashion           Fashion  United States   1000      625       30      Failed
     1  Film & Video            Shorts  United States  80000       22        3      Failed
     2           Art      Illustration  United States     20       35        3  Successful
     3    Technology          Software  United States     99      145       25  Successful
     4       Fashion           Fashion  United States   1900      387       10      Failed
   ...           ...               ...            ...    ...      ...      ...         ...
374848         Music           Hip-Hop  United States    500        0        0        Live
374849        Design    Product Design  United States  15000      269        8        Live
374850          Food              Food  United States  10000      165        3        Live
374851           Art               Art  United States    650        7        1        Live
374852         Games   Tabletop Games           Spain  24274     4483       82        Live
```

df1) by removing (dropping) specific columns from the original DataFrame df.

5.Exploring rows where goal amount is greater than pledged amount for successful kick-starters.

The code hero_df = df[df["Goal"] > df["Pledged"]] is a data filtering operation using pandas in Python. Here's the theory behind this code:

df: This variable represents the original DataFrame you are working with, which contains your dataset. ["Goal"] and ["Pledged"]: These are column selections, specifying that you want to access the "Goal" and
"Pledged" columns within the DataFrame.
df["Goal"] > df["Pledged"]: This is a conditional comparison. It compares each row's "Goal" value with the corresponding row's "Pledged" value. The result is a boolean Series, where each element is True if the
"Goal" is greater than the "Pledged" for that specific row, and False otherwise. hero_df: This is a new DataFrame created by applying the boolean Series as an indexing filter to the original DataFrame df. In other words, it includes only the rows for which the condition ("Goal" > "Pledged") is met.
The purpose of this code is to filter your dataset and create a new DataFrame, hero_df, that contains only the rows where the funding goal ("Goal") is greater than the amount pledged ("Pledged").



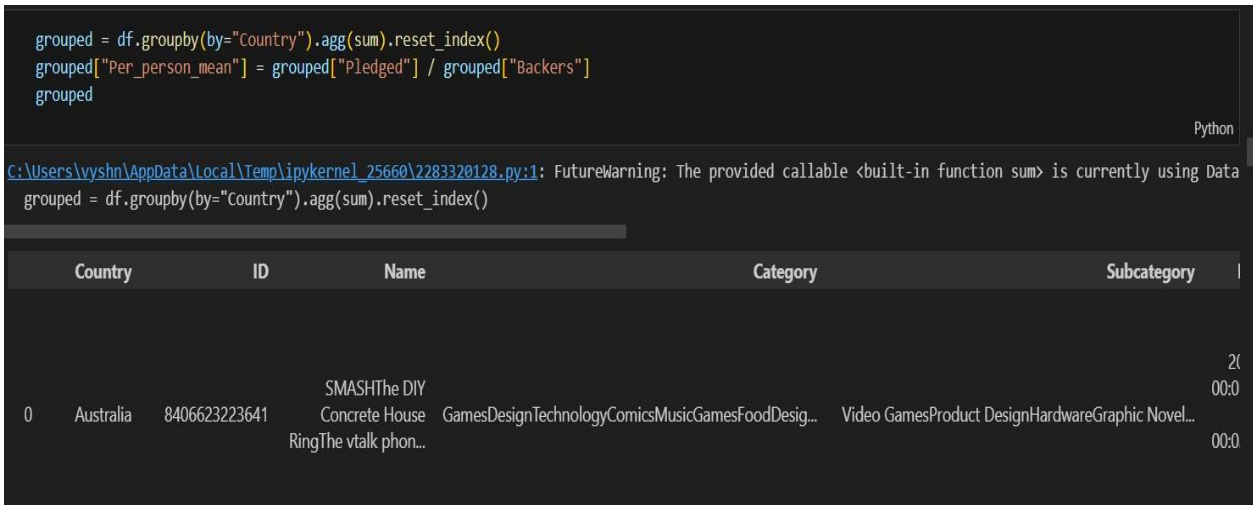
```python
hero_df = df[df["Goal"] > df["Pledged"]]
hero_df
```

Python

	ID	Name	Category	Subcategory	Country	Launched	Deadline	Goal	Pledged	Backers	State	Year
0	1860890148	Grace Jones Does Not Give A F$#% T-Shirt (limi...	Fashion	Fashion	United States	2009-04-21 21:02:48	2009-05-31	1000	625	30	Failed	2009
1	709707365	CRYSTAL ANTLERS UNTITLED MOVIE	Film & Video	Shorts	United States	2009-04-23 00:07:53	2009-07-20	80000	22	3	Failed	2009
4	1622952265	Pantshirts	Fashion	Fashion	United States	2009-04-27 14:10:39	2009-05-26	1900	387	10	Failed	2009
6	830477146	Web Site for Short Horror Film	Film & Video	Shorts	United States	2009-04-29 02:04:21	2009-05-29	200	41	3	Failed	2009
8	1502297238	Produce a Play (Canceled)	Theater	Theater	United States	2009-04-29 04:37:37	2009-06-01	500	0	0	Canceled	2009
...	...	...	...	...	...	...	...	...	...	...	...	...
374848	1486845240	Americas Got Talent - Serious MAK	Music	Hip-Hop	United States	2018-01-02 14:13:09	2018-01-16	500	0	0	Live	2018

6. Finding the per person mean of individual countries. grouped = df.groupby(by="Country").agg(sum).reset_index(): df: This variable represents the original DataFrame containing your dataset.
.groupby(by="Country"): This part of the code groups the data in the DataFrame by the "Country" column. It creates groAfterups where rows with the same "Country" value are placed in the same group. .agg(aggregatessum): This the data in each group by summing the values in each group. It effectively calculates the "talPledged" and "Backers" for each country.
.reset_index(): aggregation, it resets the index of the resulting DataFrame, which makes "Country" become a toregular column instead of an index.
The resugraoupedlt is a new DataFrame called "grouped" that contains one row per country, with the total
"Pledged" nd "Backers" for each country.

["Per_person_mean"] = grouped["Pledged"] / grouped["Backers"]: grouped: This is the DataFrame obtained from the previous step, which contains the aggregated data for each country. grouped["Per_person_mean"]: This code creates a new column called "Per_person_mean" in the
"grouped" DataFrame. grouped["Pledged"] / grouped["Backers"]: This calculates the average amount pledged per backer for
each country. It divides the total "Pledged" by the total "Backers" for each country.
The result is an additional column in the "grouped" DataFrame that represents the average amount pledged per backer for each country.
The purpose of this code is to aggregate data at the country level and calculate a meaningful metric ("Per_person_mean") related to crowdfunding campaigns in each country. This kind of aggregation and computation is valuable for generating insights, comparing performance across countries, and understanding the crowdfunding dynamics in different regions.



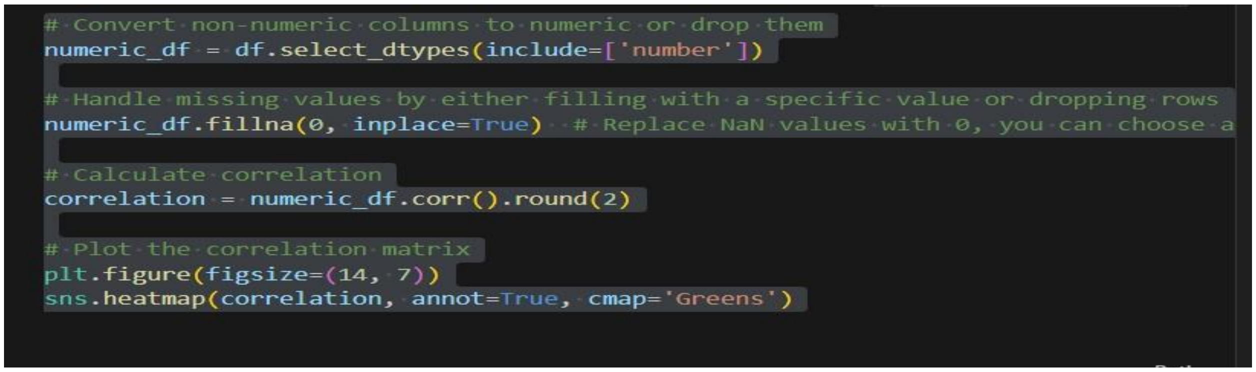
```python
grouped = df.groupby(by="Country").agg(sum).reset_index()
grouped["Per_person_mean"] = grouped["Pledged"] / grouped["Backers"]
grouped
```

Python

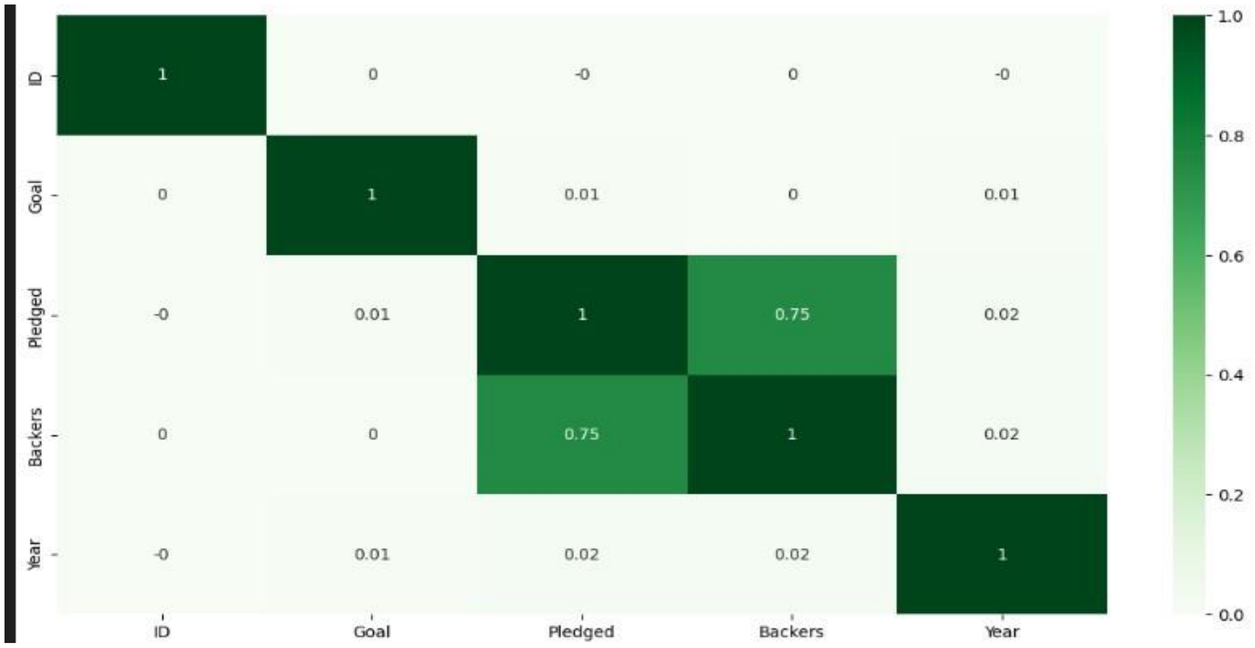
```
C:\Users\vyshn\AppData\Local\Temp\ipykernel_25660\2283320128.py:1: FutureWarning: The provided callable <built-in function sum> is currently using Data
  grouped = df.groupby(by="Country").agg(sum).reset_index()
```

	Country	ID	Name	Category	Subcategory	I
0	Australia	8406623223641	SMASHThe DIY Concrete House RingThe vtalk phon...	GamesDesignTechnologyComicsMusicGamesFoodDesig...	Video GamesProduct DesignHardwareGraphic Novel...	2(00:0 00:0

7.Determining the correlation between independent variables
numeric_df = df.select_dtypes(include=['number']): df: This variable
represents the original DataFrame containing your dataset.
.select_dtypes(include=['number']): This method selects columns in the DataFrame that have numeric data types (e.g., integers and floating-point numbers). It filters out non-numeric columns.
The result is a new DataFrame called "numeric_df" that includes only the columns with numeric data types. Non-numeric columns are dropped from this DataFrame.
numeric_df.fillna(0, inplace=True):

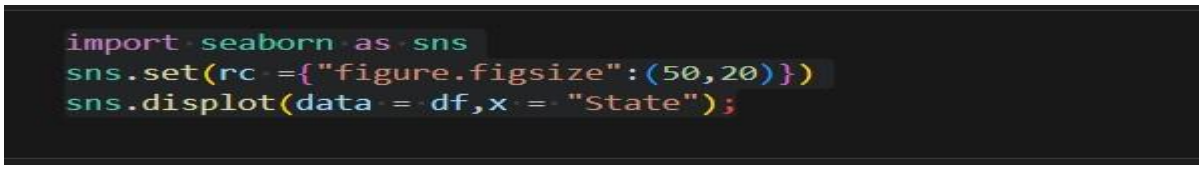


```
# Convert non-numeric columns to numeric or drop them
numeric_df = df.select_dtypes(include=['number'])

# Handle missing values by either filling with a specific value or dropping rows
numeric_df.fillna(0, inplace=True)  # Replace NaN values with 0, you can choose a

# Calculate correlation
correlation = numeric_df.corr().round(2)

# Plot the correlation matrix
plt.figure(figsize=(14, 7))
sns.heatmap(correlation, annot=True, cmap='Greens')
```

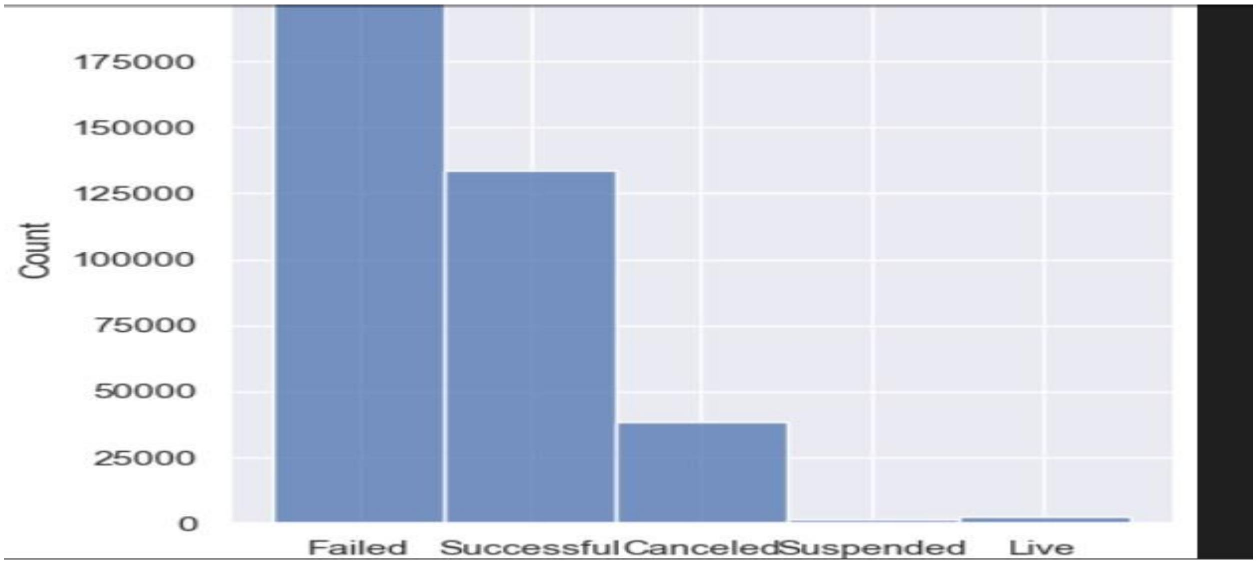


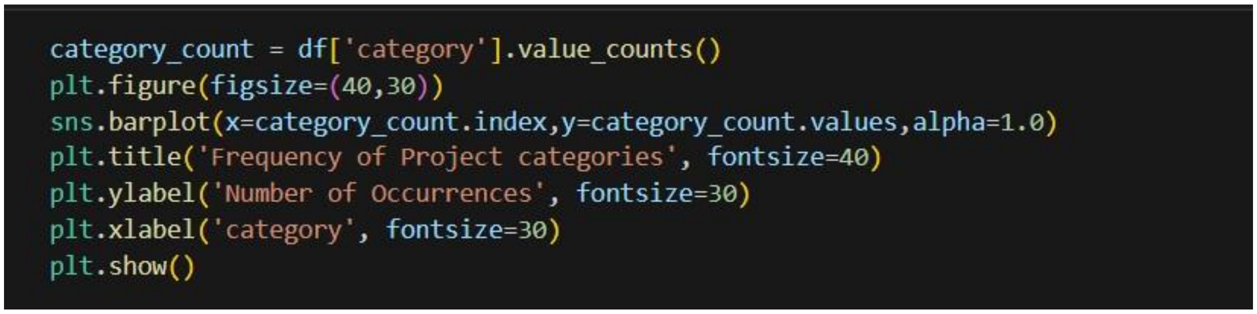
1.univarient analysis import seaborn as sns: This line imports the Seaborn library, which is commonly used for data visualization and is built on top of Matplotlib. Seaborn provides a high-level interface for creating informative and attractive statistical graphics. sns.set(rc={"figure.figsize": (50, 20)}): This code sets the figure size for the upcoming plot. The sns.set() function is used to customize the appearance of Seaborn plots. In this case, it specifies a large figure size with a width of 50 units and a height of 20 units, making the plot quite large. sns.displot(data=df, x="State"): data=df: This specifies the DataFrame (df) from which the data for the plot will be taken. x="State": This specifies the variable ("State") from the DataFrame to

be plotted on the x-axis. The sns.displot() function is used to create a distribution plot (histogram) of the "State" column in the DataFrame. It visualizes the distribution of values in the "State" column, showing how frequently each unique value occurs. Each bar in the histogram represents a specific "State" value, and the height of the bar indicates the frequency of that value.
The purpose of this code is to provide a visual representation of the distribution of the "State" column in your dataset. The large figure size (50 units in width and 20 units in height) is chosen to make the plot more readable and informative, especially if you have a large number of unique "State" values or if you want to see fine-grained details of the distribution. This can be useful for understanding the distribution of states or categories in your dataset.

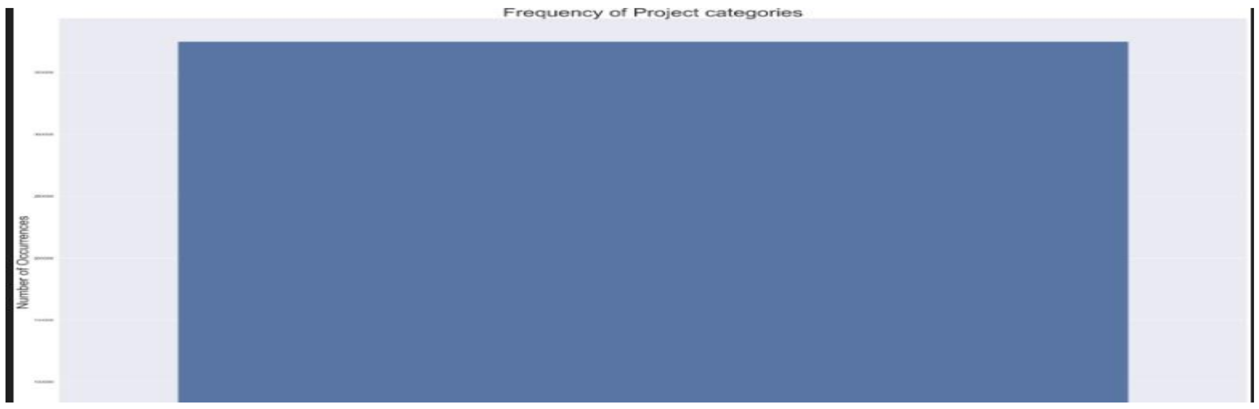


```
import seaborn as sns
sns.set(rc ={"figure.figsize":(50,20)})
sns.displot(data = df,x = "State");
```



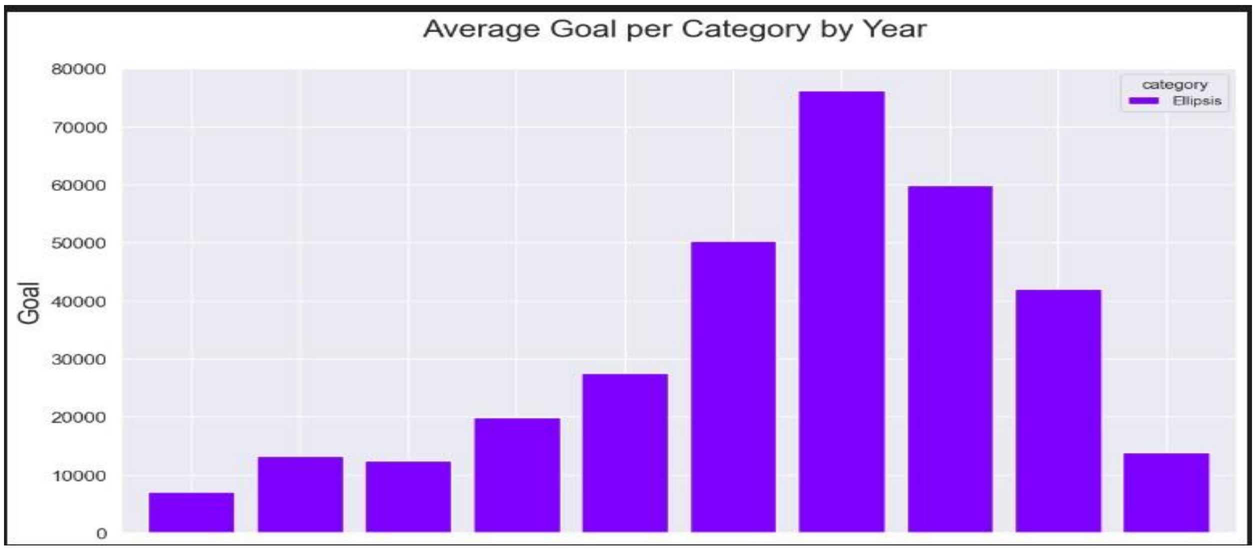


```
category_count = df['category'].value_counts()
plt.figure(figsize=(40,30))
sns.barplot(x=category_count.index,y=category_count.values,alpha=1.0)
plt.title('Frequency of Project categories', fontsize=40)
plt.ylabel('Number of Occurrences', fontsize=30)
plt.xlabel('category', fontsize=30)
plt.show()
```

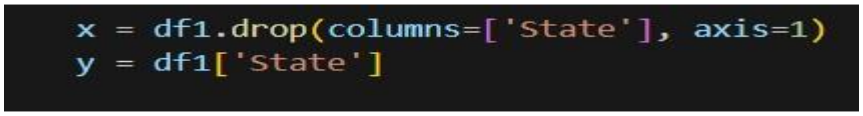


2.Bivaient analysis

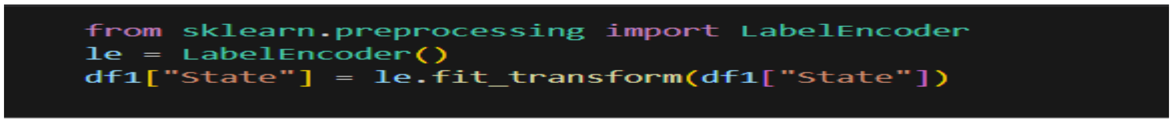
Bivariate analysis, also known as two-variable analysis, is a statistical method used to examine the relationship between two variables or factors. The primary goal of bivariate analysis is to determine whether there is a correlation or association between the two variables and to understand the nature of this relationship. Bivariate analysis is a fundamental step in data analysis and is often a precursor to more advanced multivariate analyses.



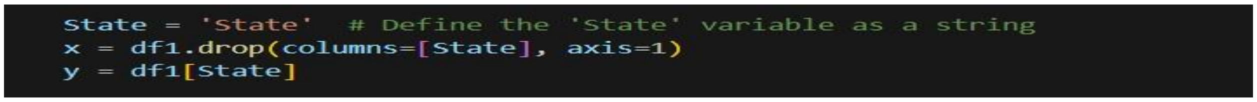
7.classification



```python
x = df1.drop(columns=['State'], axis=1)
y = df1['State']
```

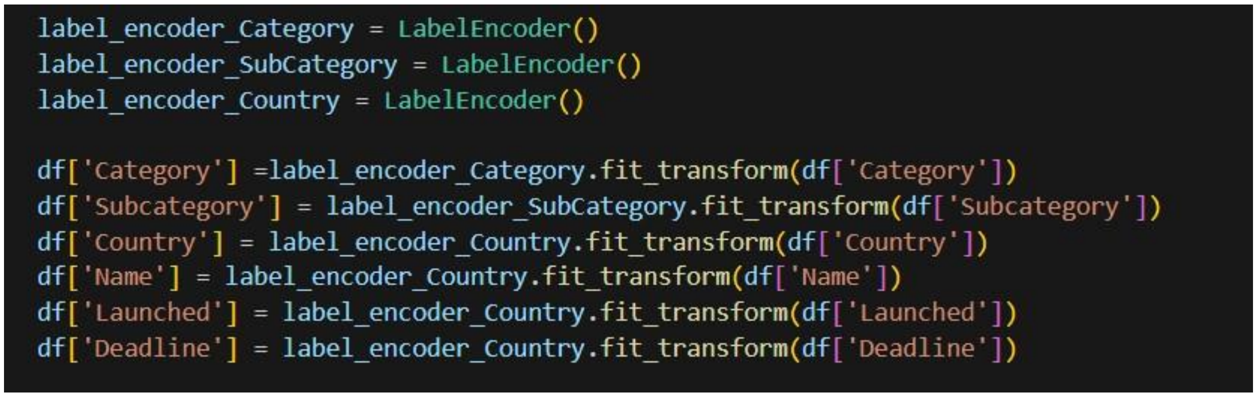


```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df1["State"] = le.fit_transform(df1["State"])
```



```python
State = 'State'  # Define the 'State' variable as a string
x = df1.drop(columns=[State], axis=1)
y = df1[State]
```

State = 'State': This line defines a string variable State with the value 'State'. It essentially stores the column name you want to isolate in a variable for convenience. x = df1.drop(columns=[State], axis=1): Here, you are creating a new DataFrame x by dropping the column specified by the variable State, which is 'State' in this case, from the DataFrame df1. The axis=1 parameter indicates that you are specifying a column operation. As a result, x will contain all columns from df1 except for the 'State' column. y = df1[State]: This line creates a Series y by selecting only the column specified by the variable State, which is 'State' in this case, from the DataFrame df1. y will contain the values from the 'State' column. The three different versions of the code are doing the same thing: they are separating the features (independent variables) into x and the target variable (dependent variable) into y, using different variable names for the 'State' column. The code is often used to prepare data for machine learning or data analysis, where you need to distinguish between the predictors and the variable you want to predict. The choice of variable names for the target column can be useful for code readability and flexibility when working with different datasets or column names.



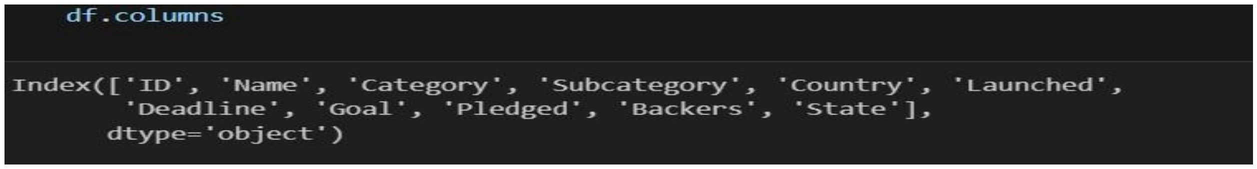
```python
label_encoder_Category = LabelEncoder()
label_encoder_SubCategory = LabelEncoder()
label_encoder_Country = LabelEncoder()

df['Category'] =label_encoder_Category.fit_transform(df['Category'])
df['Subcategory'] = label_encoder_SubCategory.fit_transform(df['Subcategory'])
df['Country'] = label_encoder_Country.fit_transform(df['Country'])
df['Name'] = label_encoder_Country.fit_transform(df['Name'])
df['Launched'] = label_encoder_Country.fit_transform(df['Launched'])
df['Deadline'] = label_encoder_Country.fit_transform(df['Deadline'])
```

df['Category'] = label_encoder_Category.fit_transform(df['Category']): This line encodes

the'Category'column of your DataFrame. The fit_transform method fits the label encoder to the

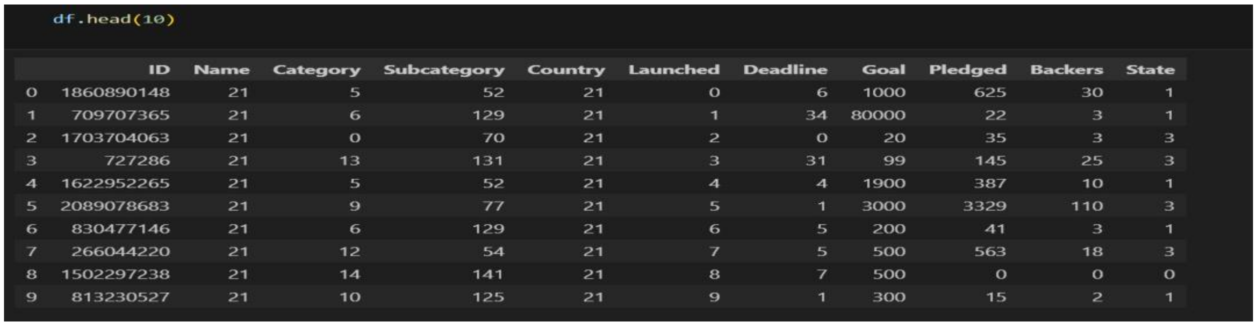
unique values in the 'Category' column and then transforms those values into numerical labels.

It's important to note that label encoding can introduce an ordinal relationship among categories that might not exist in the original data. If there is no inherent order among the categories, another encoding method like one-hot encoding should be considered to avoid misinterpretation by machine learning algorithms. Additionally, the code seems to contain some errors in the last three lines, where it attempts to apply label encoding to columns like 'Name,' 'Launched,' and 'Deadline' using the 'Country' label encoder, which doesn't seem to be intended. These columns should have their own separate label encoders if needed.



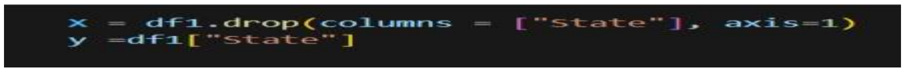
```
df.columns
```

```
Index(['ID', 'Name', 'Category', 'Subcategory', 'Country', 'Launched',
       'Deadline', 'Goal', 'Pledged', 'Backers', 'State'],
      dtype='object')
```



```
df.head(10)
```

```
           ID  Name  Category  Subcategory  Country  Launched  Deadline   Goal  Pledged  Backers  State
0  1860890148    21         5           52       21         0         6   1000      625       30      1
1   709707365    21         6          129       21         1        34  80000       22        3      1
2  1703704063    21         0           70       21         2         0     20       35        3      3
3      727286    21        13          131       21         3        31     99      145       25      3
4  1622952265    21         5           52       21         4         4   1900      387       10      1
5  2089078683    21         9           77       21         5         1   3000     3329      110      3
6   830477146    21         6          129       21         6         5    200       41        3      1
7   266044220    21        12           54       21         7         5    500      563       18      3
8  1502297238    21        14          141       21         8         7    500        0        0      0
9   813230527    21        10          125       21         9         1    300       15        2      1
```



```
x = df1.drop(columns = ["State"], axis=1)
y =df1["State"]
```

8.Applying Scaling and Splitting in Train and Test

Applying Scaling and Splitting in Train and Test is a crucial part of the data preprocessing pipeline in machine learning:

Scaling:

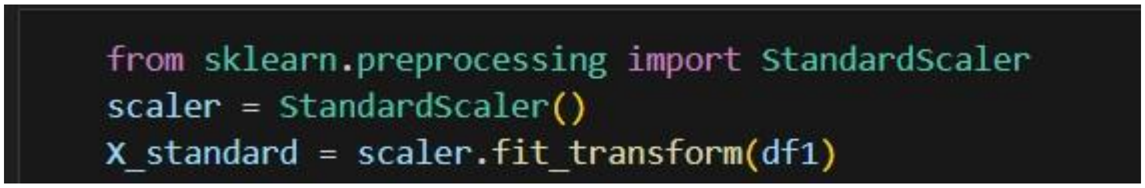
Purpose: Ensure all features have the same scale for machine learning models.

Methods: Standardization (mean=0, std=1), Min-Max scaling (range 0-1), Robust scaling (resistant to outliers), etc.

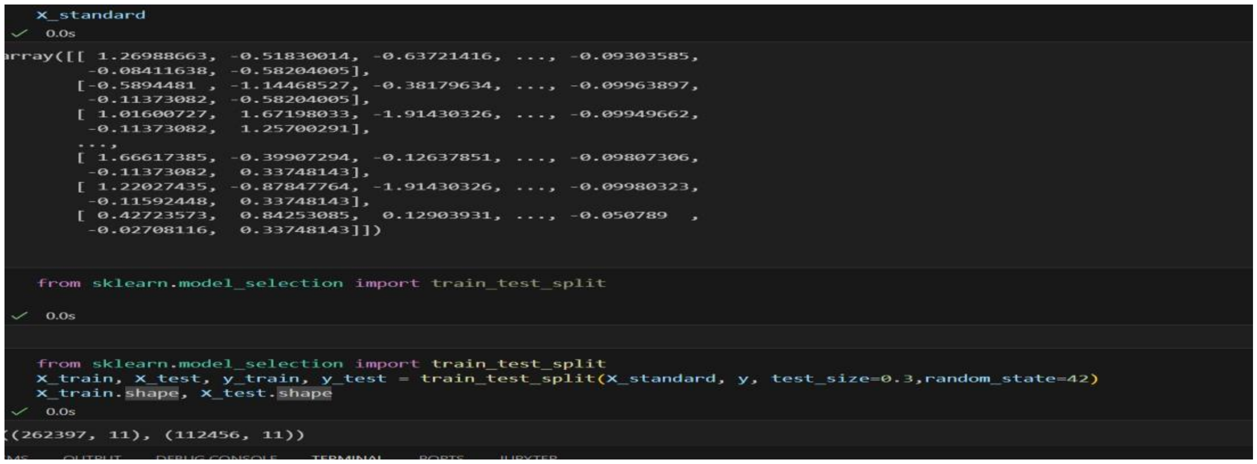
Splitting into Train and Test Sets:

Purpose: Evaluate model performance on unseen data and prevent overfitting.

Methods: Train-Test Split (dividing data into training and testing subsets), Cross-Validation (multiple train-test splits for robust evaluation), Stratified Sampling (ensuring representative class distribution). In practice, data is first split into training and testing sets, and then scaling is applied to the features, keeping the two sets independent to assess model generalization effectively.



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_standard = scaler.fit_transform(df1)
```



```
X_standard
✓ 0.0s
array([[ 1.26988663, -0.51830014, -0.63721416, ..., -0.09303585,
        -0.08411638, -0.58204005],
       [-0.5894481 , -1.14468527, -0.38179634, ..., -0.09963897,
        -0.11373082, -0.58204005],
       [ 1.01600727,  1.67198033, -1.91430326, ..., -0.09949662,
        -0.11373082,  1.25700291],
       ...,
       [ 1.66617385, -0.39907294, -0.12637851, ..., -0.09807306,
        -0.11373082,  0.33748143],
       [ 1.22027435, -0.87847764, -1.91430326, ..., -0.09980323,
        -0.11592448,  0.33748143],
       [ 0.42723573,  0.84253085,  0.12903931, ..., -0.050789  ,
        -0.02708116,  0.33748143]])
```

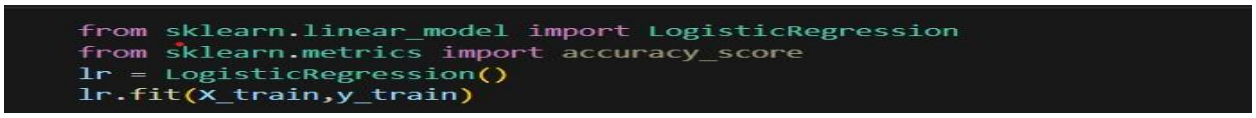
```
from sklearn.model_selection import train_test_split
✓ 0.0s
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_standard, y, test_size=0.3,random_state=42)
X_train.shape, X_test.shape
✓ 0.0s
((262397, 11), (112456, 11))
```

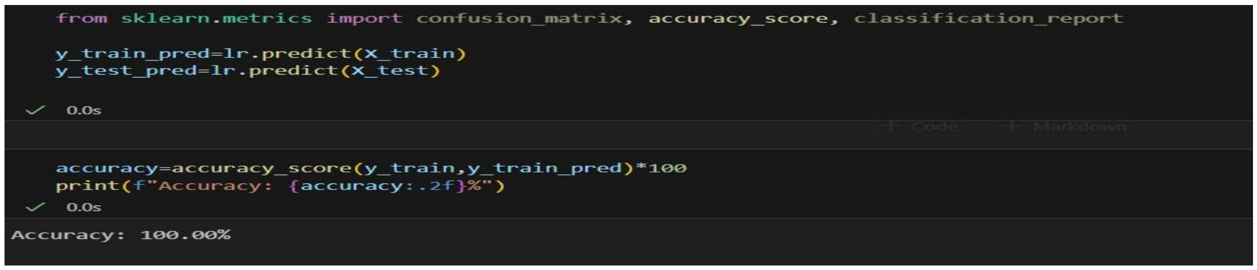
MODEL BUILDING

9.Training The Model In Multiple Algorithms

1.Logistic regression is a statistical method used for binary classification, which means it's primarily used when the dependent variable is a binary outcome, such as 0/1, True/False, Yes/No, or any other two-class labels. It is a type of generalized linear model (GLM) and is widely used in various fields, including machine learning, statistics, and social sciences.



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

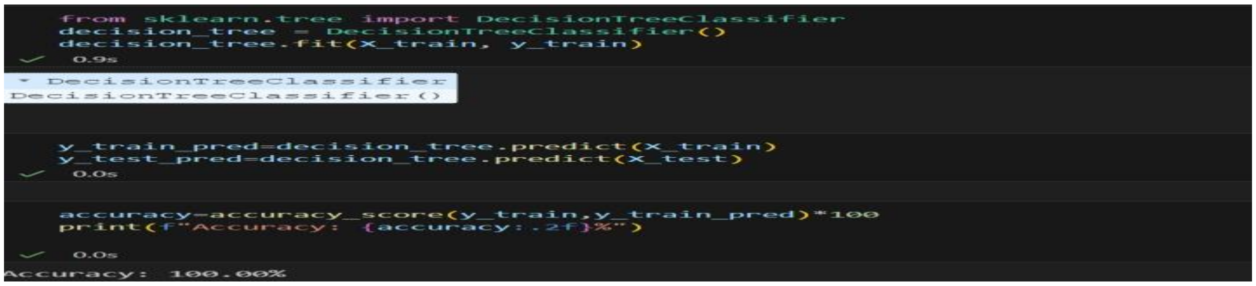


```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

y_train_pred=lr.predict(X_train)
y_test_pred=lr.predict(X_test)
✓ 0.0s
```

```
accuracy=accuracy_score(y_train,y_train_pred)*100
print(f"Accuracy: {accuracy:.2f}%")
✓ 0.0s
Accuracy: 100.00%
```

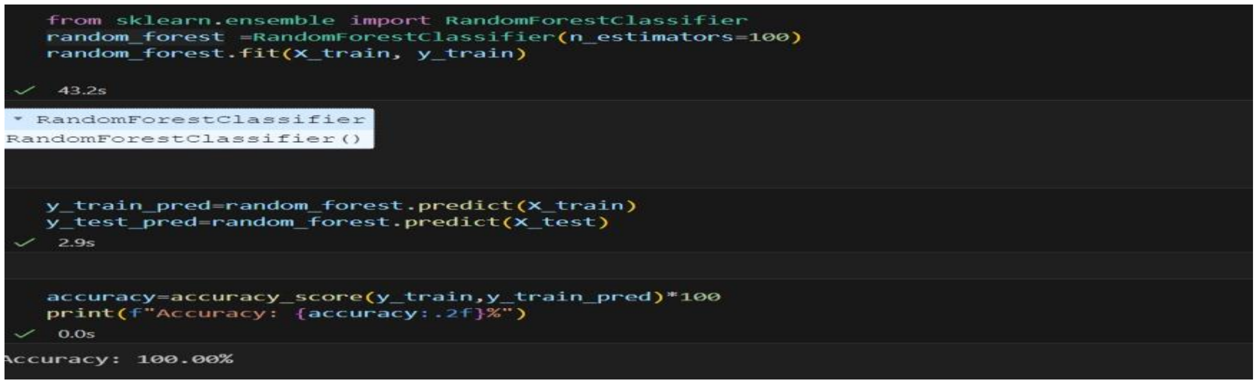
2.A decision tree algorithm is a machine learning technique used for both classification and regression tasks. It is a graphical model that represents a decision-making process, where each internal node of the tree corresponds to a decision or test on an attribute, and each leaf node represents an outcome or a class label. Decision trees are a popular choice for predictive modeling because they are easy to understand and interpret.

A decision tree is a tree-like structure used in machine learning and data mining for decision support. It recursively splits the dataset into subsets based on the most significant attribute at each step, aiming to maximize information gain, minimize impurity, or optimize another criterion, depending on the algorithm used. The ultimate goal is to create a tree that can make accurate predictions or classifications for new data points by following a path from the root node to a leaf node based on the attribute tests.



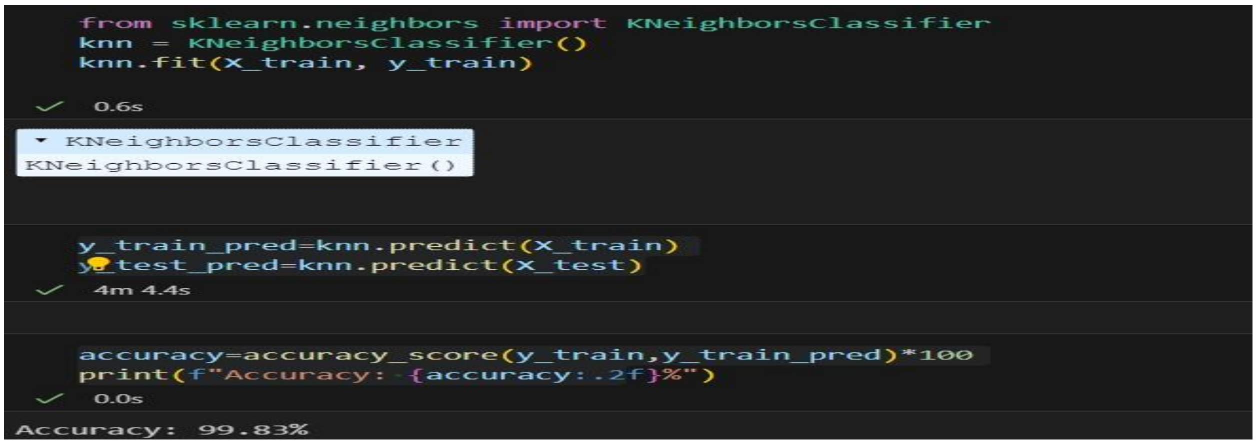
```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
 0.9s
DecisionTreeClassifier
DecisionTreeClassifier()

y_train_pred=decision_tree.predict(X_train)
y_test_pred=decision_tree.predict(X_test)
 0.0s

accuracy=accuracy_score(y_train,y_train_pred)*100
print(f"Accuracy: {accuracy:.2f}%")
 0.0s
Accuracy: 100.00%
```

3.A RandomForest Classifier is an ensemble learning method in machine learning that combines multiple decision trees to improve the accuracy and robustness of classification tasks. It is a specific implementation of the random forest algorithm for classification problems. Random forests are widely used for their effectiveness in handling a variety of datasets and producing reliable classification results.



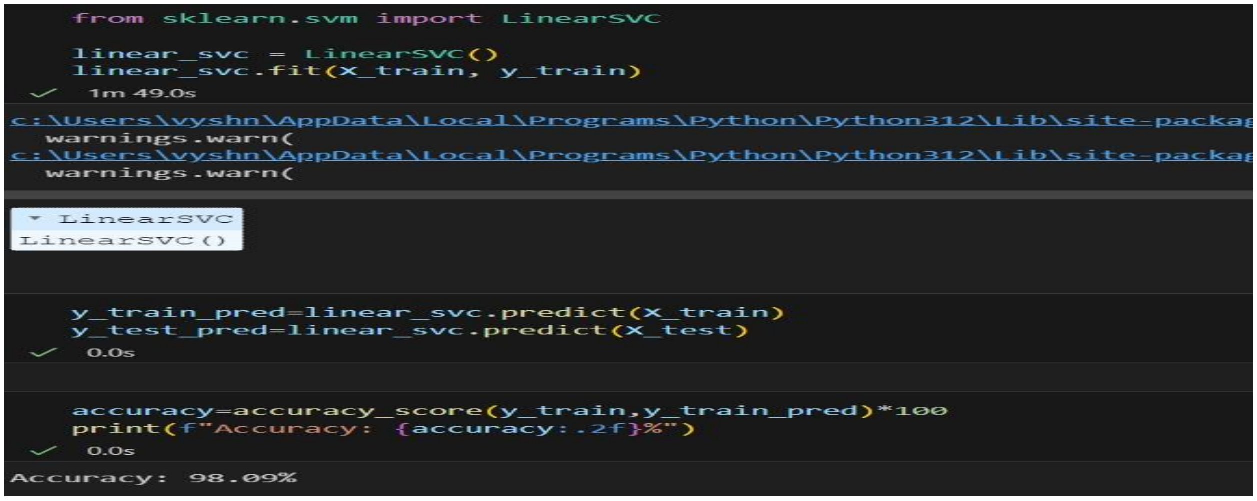
```
from sklearn.ensemble import RandomForestClassifier
random_forest =RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
 43.2s
RandomForestClassifier
RandomForestClassifier()

y_train_pred=random_forest.predict(X_train)
y_test_pred=random_forest.predict(X_test)
 2.9s

accuracy=accuracy_score(y_train,y_train_pred)*100
print(f"Accuracy: {accuracy:.2f}%")
 0.0s
Accuracy: 100.00%
```

4.The k-Nearest Neighbors (KNN) algorithm is a simple and popular supervised machine learning algorithm used for classification and regression tasks. KNN is a non-parametric and instance-based learning algorithm, meaning it doesn't make any assumptions about the underlying data distribution and instead makes predictions based on the similarity between data points.



```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
 0.6s
KNeighborsClassifier
KNeighborsClassifier()

y_train_pred=knn.predict(X_train)
y_test_pred=knn.predict(X_test)
 4m 4.4s

accuracy=accuracy_score(y_train,y_train_pred)*100
print(f"Accuracy: {accuracy:.2f}%")
 0.0s
Accuracy: 99.83%
```

5.Linear Support Vector Classifier (Linear SVC) is a machine learning algorithm used for binary classification. It aims to find a hyperplane that effectively separates data points belonging to two different classes while maximizing the margin between them. In a binary classification problem, Linear SVC assigns data points to one of two classes based on their position with respect to the separating hyperp

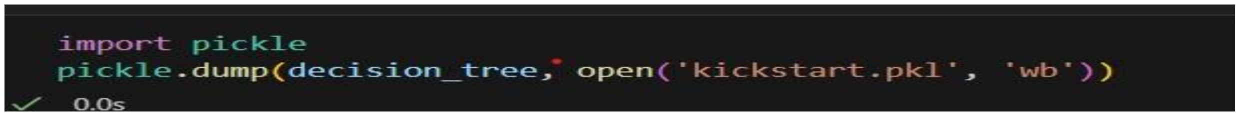


```
from sklearn.svm import LinearSVC

linear_svc = LinearSVC()
linear_svc.fit(X_train, y_train)
 1m 49.0s
c:\Users\vyshn\AppData\Local\Programs\Python\Python312\Lib\site-packag
  warnings.warn(
c:\Users\vyshn\AppData\Local\Programs\Python\Python312\Lib\site-packag
  warnings.warn(

▾ LinearSVC
LinearSVC()

y_train_pred=linear_svc.predict(X_train)
y_test_pred=linear_svc.predict(X_test)
 0.0s

accuracy=accuracy_score(y_train,y_train_pred)*100
print(f"Accuracy: {accuracy:.2f}%")
 0.0s
Accuracy: 98.09%
```

MODEL DEPLOYMENT

10.save the best model

import pickle: This line imports the pickle module, which is used for serializing (pickling) and deserializing (unpickling) Python objects, including machine learning models. pickle.dump(decision_tree, open('kickstart.pkl', 'wb')): This line pickles the decision_tree model and saves it to a file named 'kickstart.pkl' in binary write ('wb') mode. Here's what each part of this line does: pickle.dump(...): The pickle.dump() function is used to serialize and save the Python object (in this case, the decision_tree model) to a file. decision_tree: This is the trained decision tree model that you want to save. open('kickstart.pkl', 'wb'): This part opens a file named 'kickstart.pkl' in binary write mode ('wb'). You're specifying 'wb' because you are writing binary data (the serialized model) to the file.



```
import pickle
pickle.dump(decision_tree, open('kickstart.pkl', 'wb'))
 0.0s
```

INTEGRATE WITH WEB FRAMEWORK

11.integrating the web: a In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

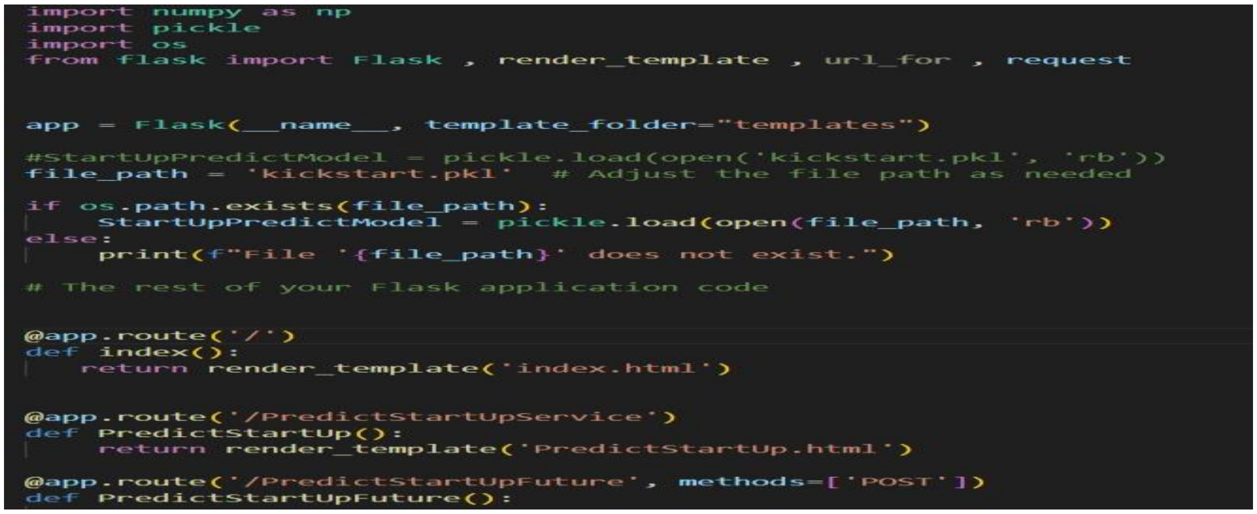
This section has the following tasks

Building HTML Pages

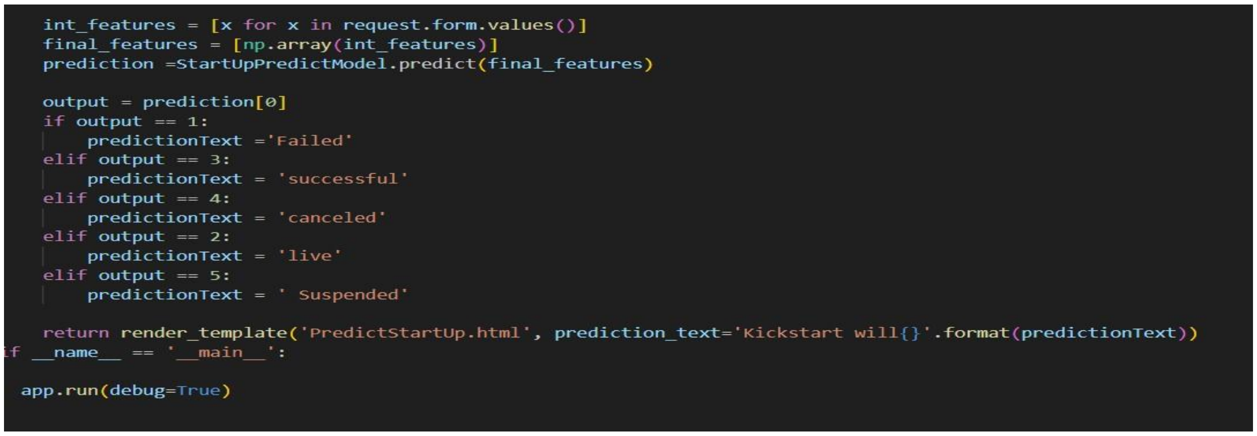
Building server-side script

Run the web application
Building Html Pages:
For this project create two HTML files and save them in the templates folder. index.html PredictStartUp.html

Build Python code:Import the libraries and load the saved model. Importing the flask module in project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (name) as argument.

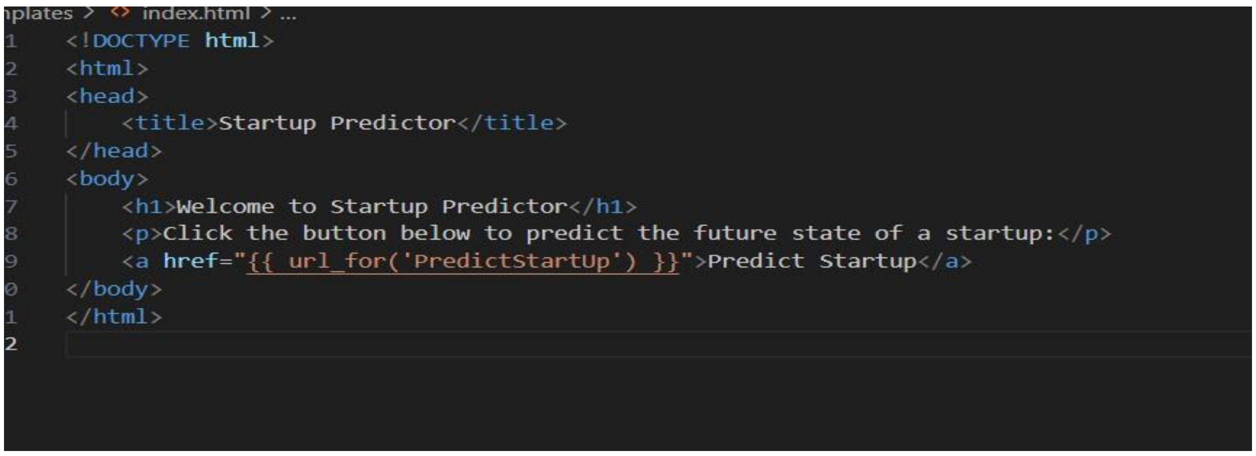


```
import numpy as np
import pickle
import os
from flask import Flask , render_template , url_for , request


app = Flask(__name__, template_folder="templates")

#StartUpPredictModel = pickle.load(open('kickstart.pkl', 'rb'))
file_path = 'kickstart.pkl'  # Adjust the file path as needed

if os.path.exists(file_path):
    StartUpPredictModel = pickle.load(open(file_path, 'rb'))
else:
    print(f"File '{file_path}' does not exist.")

# The rest of your Flask application code

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/PredictStartUpService')
def PredictStartUp():
    return render_template('PredictStartUp.html')

@app.route('/PredictStartUpFuture', methods=['POST'])
def PredictStartUpFuture():
```



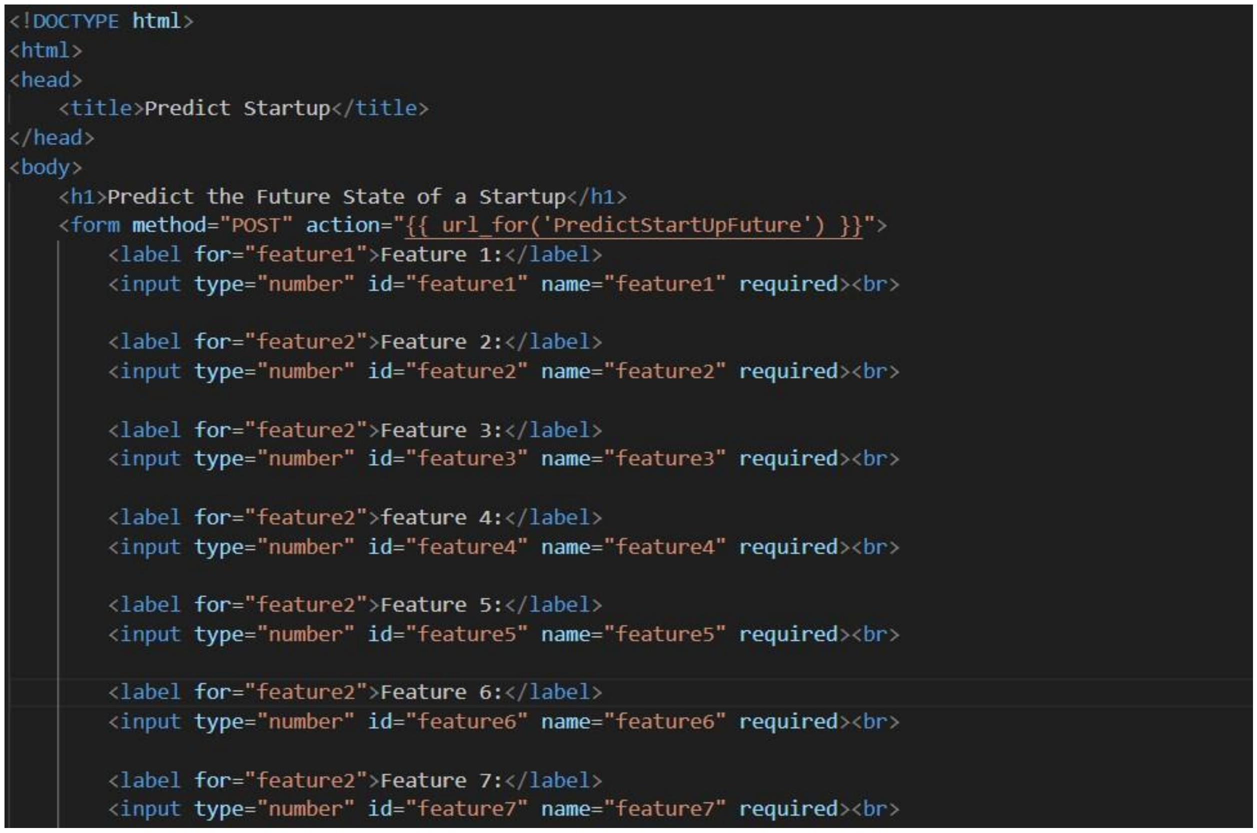
```
    int_features = [x for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction =StartUpPredictModel.predict(final_features)

    output = prediction[0]
    if output == 1:
        predictionText ='Failed'
    elif output == 3:
        predictionText = 'successful'
    elif output == 4:
        predictionText = 'canceled'
    elif output == 2:
        predictionText = 'live'
    elif output == 5:
        predictionText = ' Suspended'

    return render_template('PredictStartUp.html', prediction_text='Kickstart will{}'.format(predictionText))
if __name__ == '__main__':

  app.run(debug=True)
```

Index .html page:

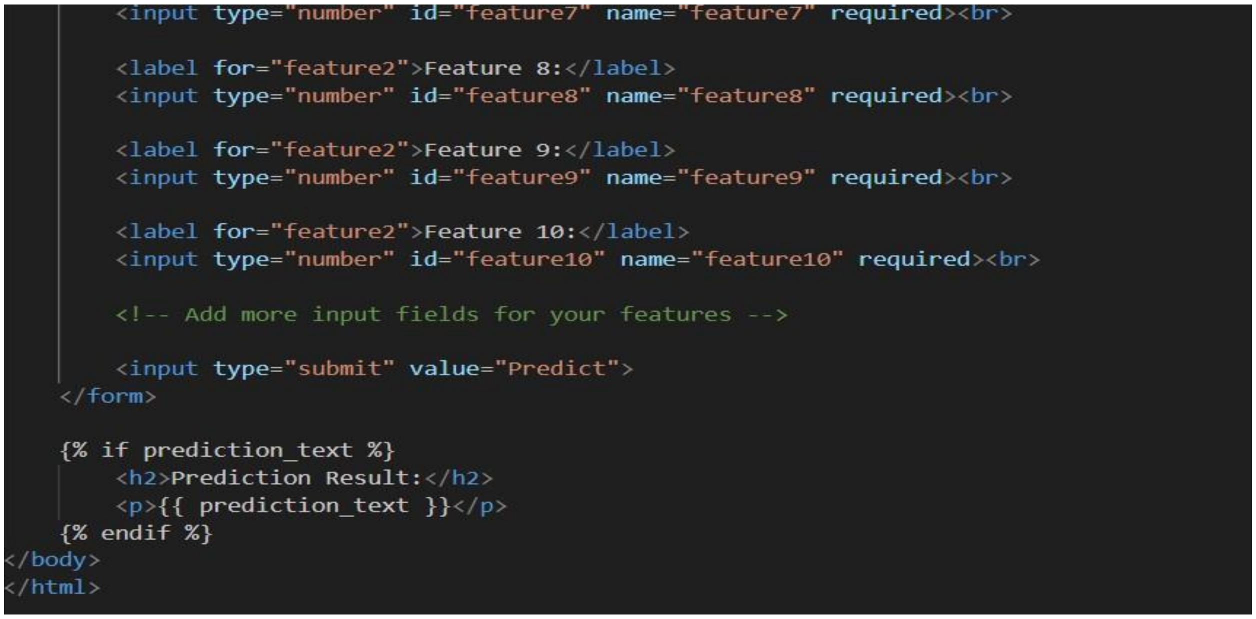


```
nplates > <> index.html > ...
<!DOCTYPE html>
<html>
<head>
    <title>Startup Predictor</title>
</head>
<body>
    <h1>Welcome to Startup Predictor</h1>
    <p>Click the button below to predict the future state of a startup:</p>
    <a href="{{ url_for('PredictStartUp') }}">Predict Startup</a>
</body>
</html>

```

pedictstartup.html page:

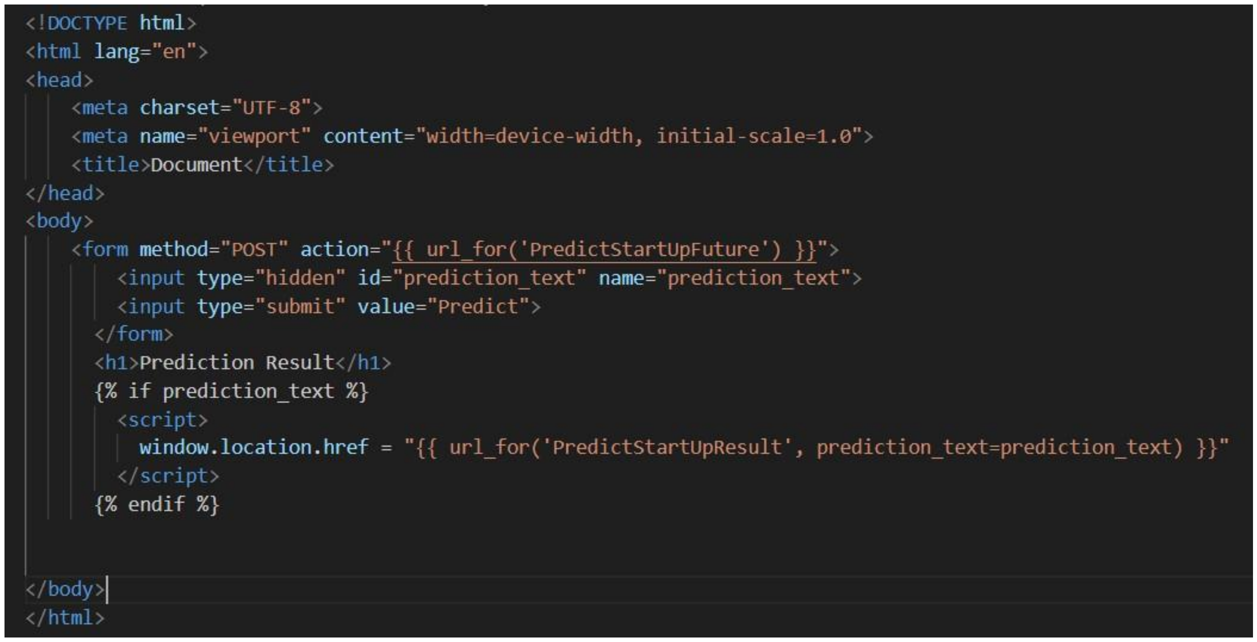


```
<!DOCTYPE html>
<html>
<head>
    <title>Predict Startup</title>
</head>
<body>
    <h1>Predict the Future State of a Startup</h1>
    <form method="POST" action="{{ url_for('PredictStartUpFuture') }}">
        <label for="feature1">Feature 1:</label>
        <input type="number" id="feature1" name="feature1" required><br>

        <label for="feature2">Feature 2:</label>
        <input type="number" id="feature2" name="feature2" required><br>

        <label for="feature2">Feature 3:</label>
        <input type="number" id="feature3" name="feature3" required><br>

        <label for="feature2">feature 4:</label>
        <input type="number" id="feature4" name="feature4" required><br>

        <label for="feature2">Feature 5:</label>
        <input type="number" id="feature5" name="feature5" required><br>

        <label for="feature2">Feature 6:</label>
        <input type="number" id="feature6" name="feature6" required><br>

        <label for="feature2">Feature 7:</label>
        <input type="number" id="feature7" name="feature7" required><br>
```



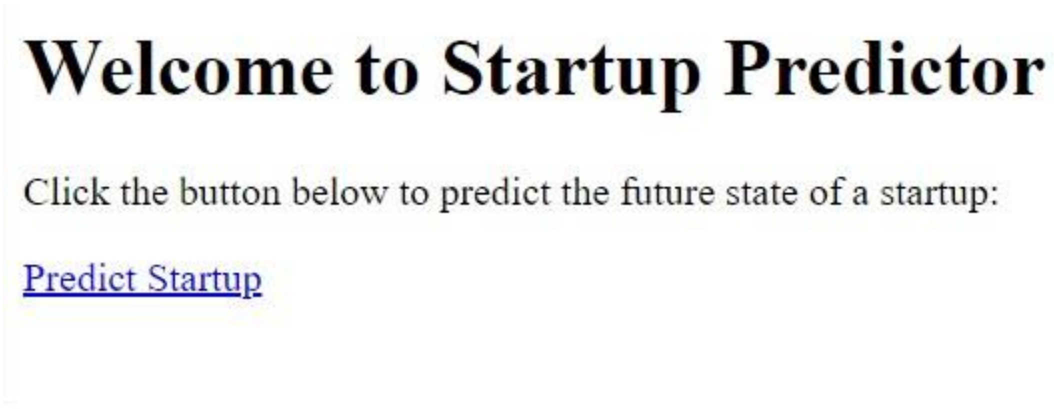
```
        <input type="number" id="feature7" name="feature7" required><br>

        <label for="feature2">Feature 8:</label>
        <input type="number" id="feature8" name="feature8" required><br>

        <label for="feature2">Feature 9:</label>
        <input type="number" id="feature9" name="feature9" required><br>

        <label for="feature2">Feature 10:</label>
        <input type="number" id="feature10" name="feature10" required><br>

        <!-- Add more input fields for your features -->

        <input type="submit" value="Predict">
    </form>

    {% if prediction_text %}
        <h2>Prediction Result:</h2>
        <p>{{ prediction_text }}</p>
    {% endif %}
</body>
</html>
```

predictionstartupresult.html:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form method="POST" action="{{ url_for('PredictStartUpFuture') }}">
        <input type="hidden" id="prediction_text" name="prediction_text">
        <input type="submit" value="Predict">
      </form>
      <h1>Prediction Result</h1>
      {% if prediction_text %}
        <script>
          window.location.href = "{{ url_for('PredictStartUpResult', prediction_text=prediction_text) }}"
        </script>
      {% endif %}


</body>
</html>
```

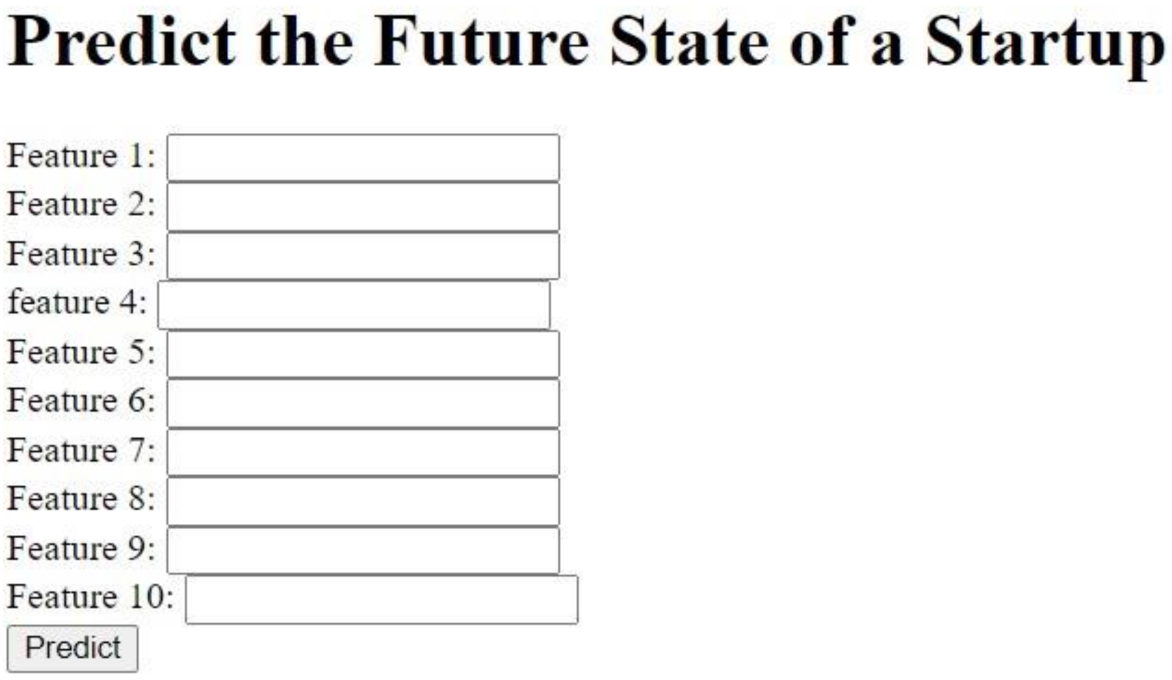
FINAL OUTPUT:



Welcome to Startup Predictor

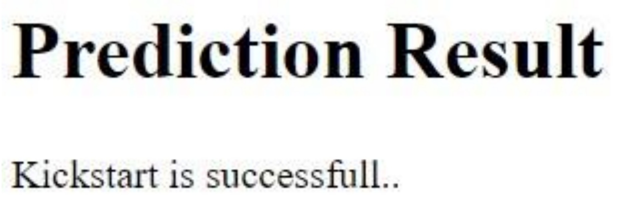
Click the button below to predict the future state of a startup:

Predict Startup



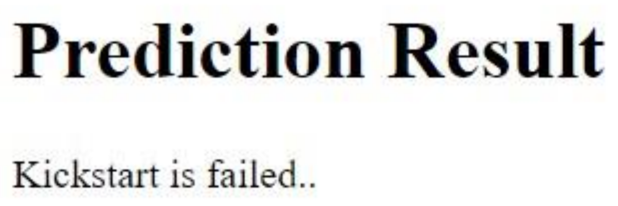
Predict the Future State of a Startup

Feature 1:
Feature 2:
Feature 3:
feature 4:
Feature 5:
Feature 6:
Feature 7:
Feature 8:
Feature 9:
Feature 10:



Prediction Result

Kickstart is successfull..



Prediction Result

Kickstart is failed..

CONCLUSION

In conclusion, an automated Kickstarter campaign can provide creators with several advantages and opportunities, as well as some notable disadvantages and challenges. Leveraging automation tools and strategies in a Kickstarter campaign can streamline certain aspects of the process, improve efficiency, and enhance the overall experience for both creators and backers.

The advantages of automating a Kickstarter campaign include efficient communication, data tracking, and email marketing, allowing creators to focus on their project's development and fulfillment. Automation can also aid in backer engagement and management, making it easier to nurture relationships with supporters and keep them informed.

However, creators should remain mindful of the potential drawbacks of automation, such as the risk of appearing impersonal or spammy. Balancing automation with personalized interactions is essential to maintain a positive relationship with backers.

Ultimately, an automated Kickstarter campaign should be part of a comprehensive strategy that combines the benefits of technology and personal touch. By carefully considering the pros and cons, creators can use automation to their advantage while preserving the authenticity and human connection that make crowdfunding campaigns successful.

BIBLIOGRAPHY

- Agrawal, A., Catalini, C., & Goldfarb, A. (2013). The Geography of Crowdfunding. NBER Working Paper No. 16820. Retrieved from https://www.nber.org/papers/w16820

- Belleflamme, P., Lambert, T., & Schwienbacher, A. (2014). Crowdfunding: Tapping the Right Crowd. Journal of Business Venturing, 29(5), 585-609. doi:10.1016/j.jbusvent.2013.07.003

- Kickstarter. (n.d.). Kickstarter Basics. Retrieved from https://www.kickstarter.com/help/basics

- Mollick, E. (2014). The Dynamics of Crowdfunding: An Exploratory Study. Journal of Business Venturing, 29(1), 1-16. doi:10.1016/j.jbusvent.2013.06.005