Leeds University Business School

UNIVERSITY OF LEEDS

Assessed Coursework Coversheet

For use with *individual* assessed work

Student ID Number:	2	0	1	6	8	4	9	3	3
Module Code:	LUBS5990M								
Module Title:	Machine Learning in Practice								
Module Leader:	Dr Xingjie Wei								
Declared Word Count:	3499								

Please Note:

Your declared word count must be accurate, and should not mislead. Making a fraudulent statement concerning the work submitted for assessment could be considered academic malpractice and investigated as such. If the amount of work submitted is higher than that specified by the word limit or that declared on your word count, this may be reflected in the mark awarded and noted through individual feedback given to you.

It is not acceptable to present matters of substance, which should be included in the main body of the text, in the appendices ("appendix abuse"). It is not acceptable to attempt to hide words in graphs and diagrams; only text which is strictly necessary should be included in graphs and diagrams.

By submitting an assignment you confirm you have read and understood the University of Leeds **Declaration of Academic Integrity** (http://www.leeds.ac.uk/secretariat/documents/academic_integrity.pdf).

Introduction

Crowdfunding is a popular method to fund their projects or company over the internet. ICO or Initial coin offering is a new method to raise funding by issuing and selling blockchain based cryptocurrencies or digital coins. By accurately predicting the success or failure of an ICO, it becomes possible to make informed decisions regarding investment opportunities and resource allocation. The dataset provided contains several variables that can potentially influence the outcome of an ICO campaign, which we will discuss in detail in next sections.

By using various machine learning models, our aim is to create a predictive model than can efficiently analyze the variable and accurately classify whether a fundraising team or company will achieve the fundraising goal. These models can help in providing valuable insights to investors, crowdfunding platforms, and fundraising teams as well.

Overall, the use of machine learning models to predict the success of ICO campaigns can contribute to more efficient and informed decision-making processes in the crowdfunding ecosystem.

Data Understanding

We are provided with the data source as a CSV file, in python with the help of pandas library we imported the columns and it's values in pandas datarame. Table 1 tells shows the description of each variable imported with its default data type assigned when imported in python dataframe.

Table 1

Columns	Python default type assigned	Description
success	object	'Y' for 'yes' and 'N' for 'no' representing if the fundraising gold is achieved or not.
startDate	object	Start date of fund-raising campaign.
endDate	object	End date
coinNum	int64	number of blockchain coins issued.
priceUSD	float64	price of each blockchain coin in USD.
teamSize	float64	team members count in each project.
countryRegion	object	country/region of fundraising team/company.
rating	float64	score rating out of 5 (5 being best), for each fundraising project
minInvestment	int64	1 if project has set minimum investment amount (10USD) else 0.
distrubutedPercentage	float64	% of coins distributed to investors.
platform	object	name of blockchain coin
hasVideo	int64	1 if they have video, else 0.
hasGithub	int64	1 if they have github page, else 0.
hasReddit	int64	1 if they have reddit page, else 0.

In Table 1, we see that for some of the columns as per description should be categorical type but are assigned with integer or float data type, this we will fix in data processing step.
Now let's see distribution of success column to get an overview:

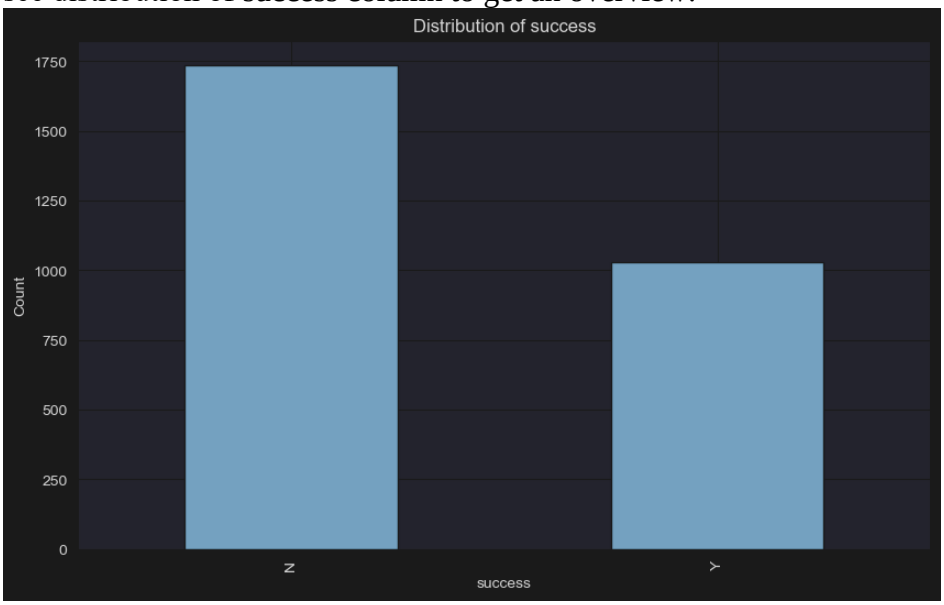

Figure 1

From figure 1, we see that number of 'N' are more than 'Y' in success column, but overall, the distribution is fairly distributed between no and yes values as one value is not too low or high.

In figure 2, we see the word cloud showing the distribution of platform column, it can be seen that Ethereum has highest number of occurrence, and we can also see Ethereum repeated twice in the word cloud, which might suggest that there are duplicate values but for some reason they are counted as separately.

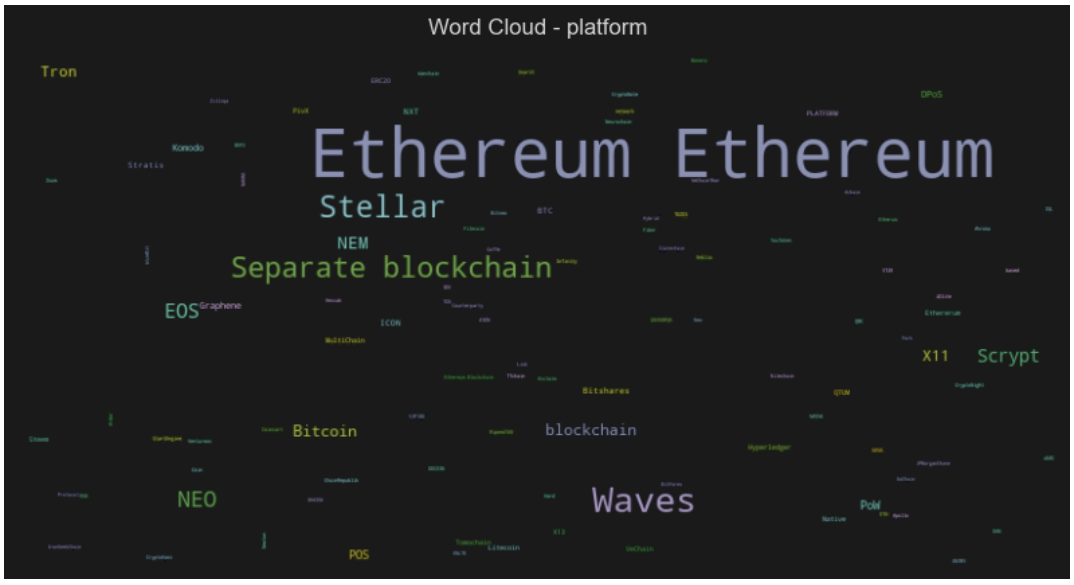

Figure 2

The gaps in the word cloud might be due to presence of empty strings, which is confirmed from figure 3, as we see that only platform has some empty strings.

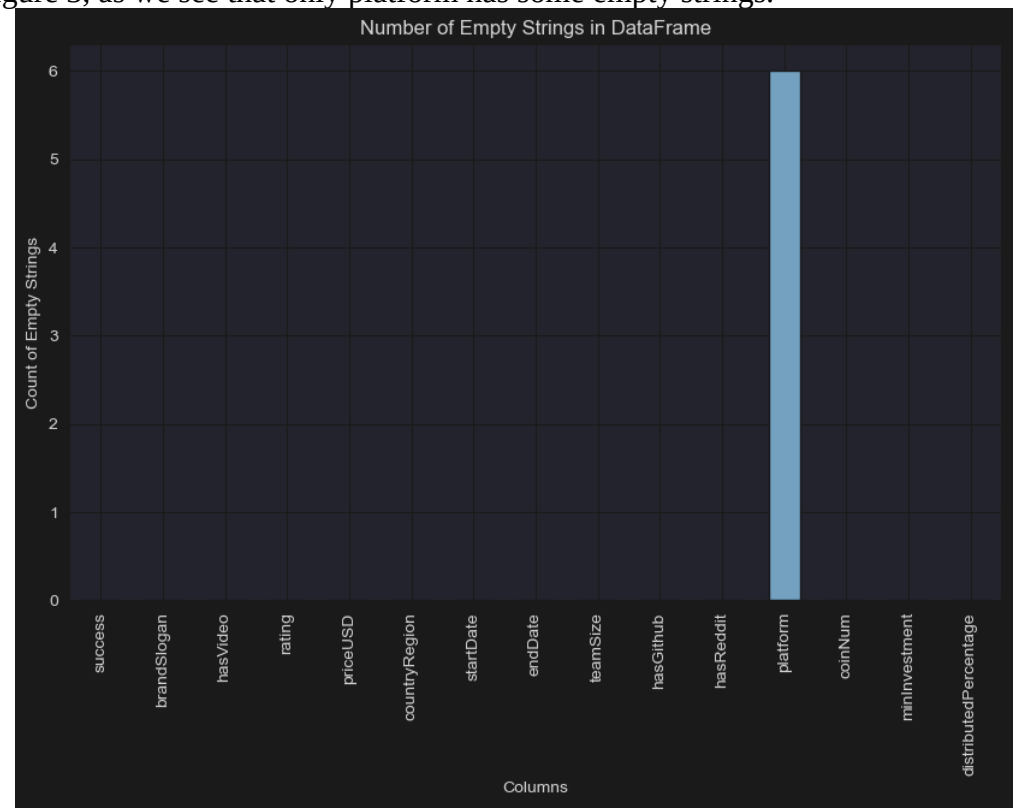

Figure 3

Figure 4 shows the wordcloud for column countryRegion, and we can see that Singapore, USA, UK, and Estonia has most presence in our dataset, along with some other countries. We cannot see any inconsistency in names, they seem fine and no repeated values with short forms can be seen.

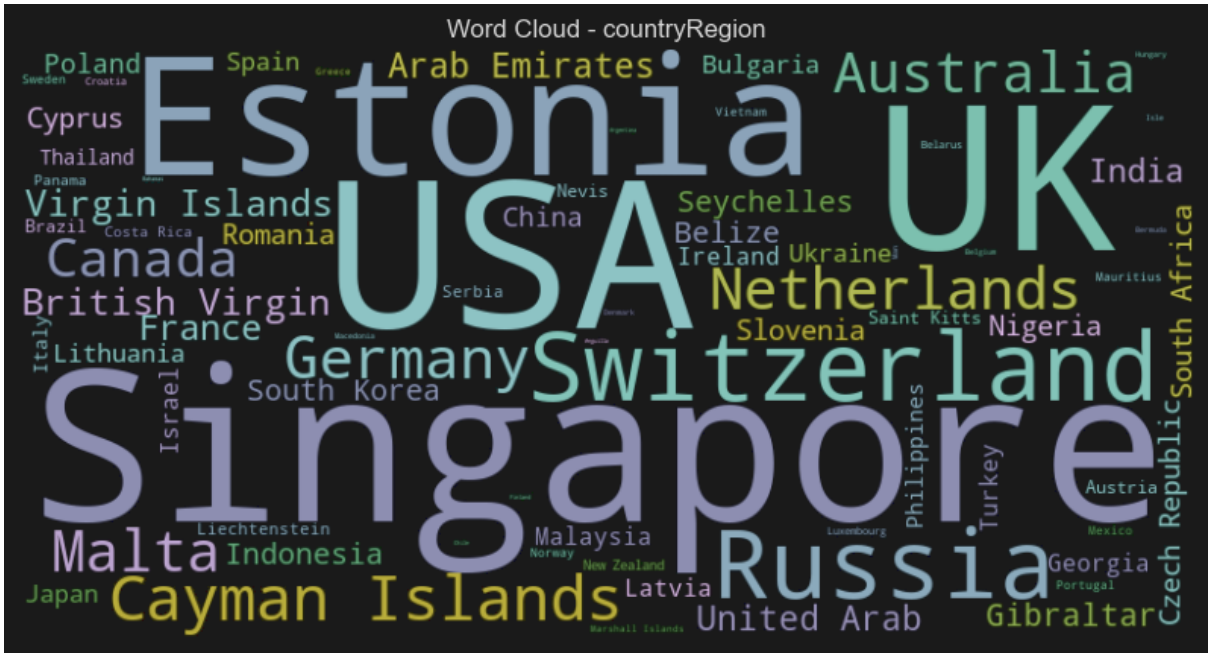

Figure 4

Figure 5 shows us the N/A values in each column, and priceUSD, countryRegion and teamSize have some N/A values which needs to be fixed.
It is important to note that N/A values are not same as null string, which was seen in figure 3, and that is the reason we don't see any value in platform as it has empty string values but not N/A values.

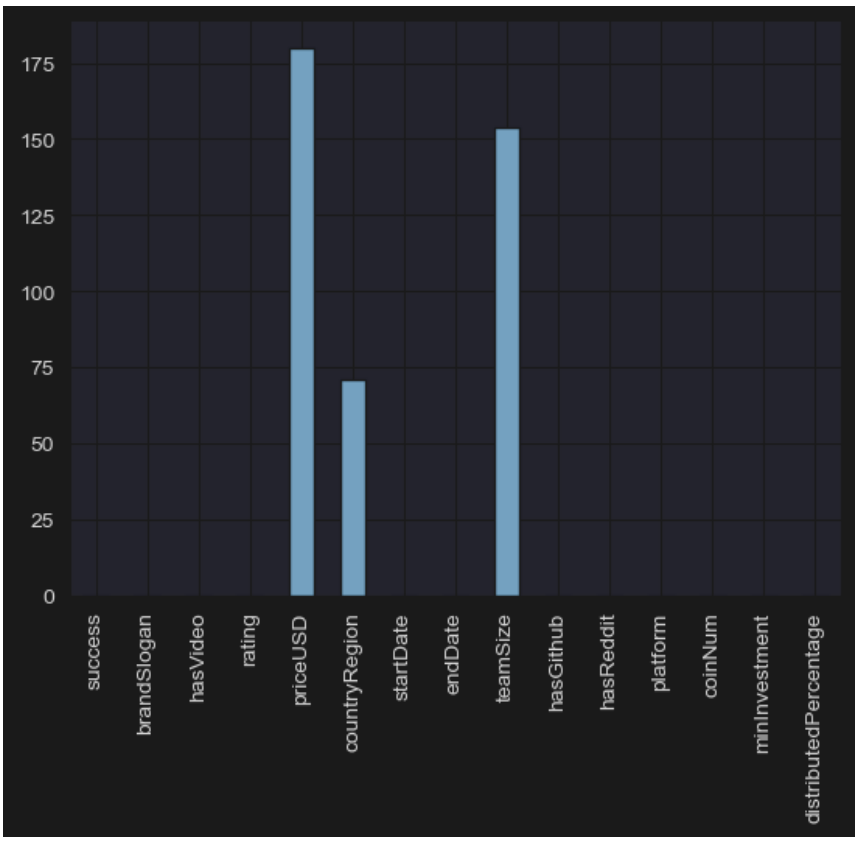

Figure 5

Data Preparation

Data type fix

As seen in Table 1, there are some columns whose data type we need to change. Following changes were made to datatypes and we can see the result in below table:

Table 2

Columns	dTypes
success	category
hasVideo	category
rating	category
countryRegion	object
startDate	datetime64 [ns]
endDate	datetime64 [ns]
hasGithub	category
hasReddit	category
minInvestment	category

Missing and incorrect values

Once we have characterized our data into correct format, next step will be to deal with null strings and N/A values present in our dataset.
To not deal with null strings separately, we converted the empty string values into N/A values, figure 6 shows the result after replacing with N/A values, we can see there are no column with empty string, but now platform also has N/A values.

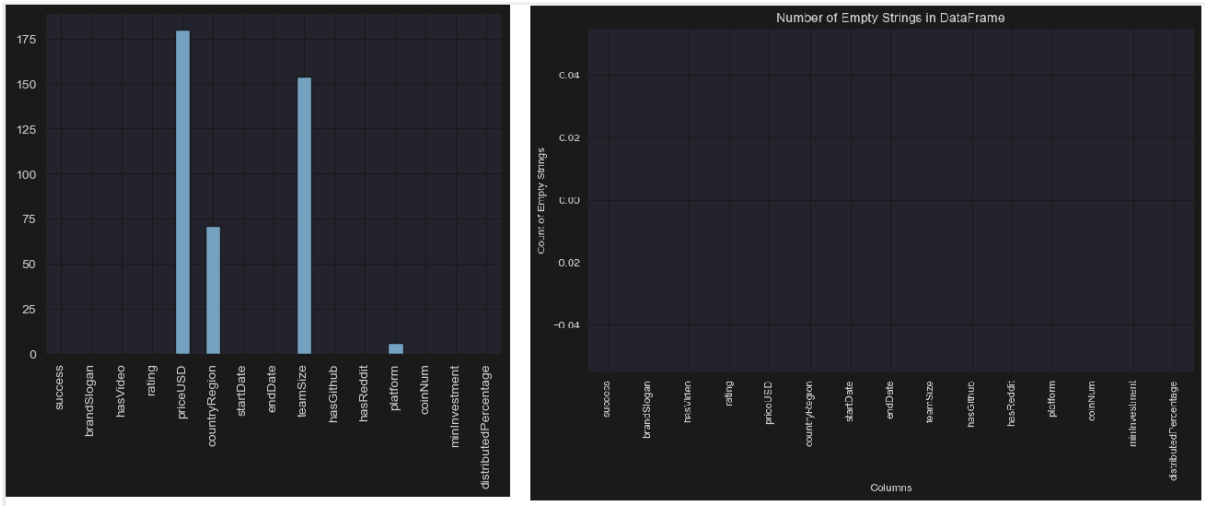

Figure 6

Now, we performed simple imputation to replace the values, table 3 shows the N/A values replaced with what.

Table 3

Columns	Values replaced with
priceUSD	Mean of column without N/A values
countryRegion	'Unknown'
teamSize	Median of column without N/A values
platform	'Unknown'

priceUSD has the highest value of N/A values and thus it is not feasible to simply remove those rows, so instead we calculated mean of data with N/A values and then replaced the values with mean. We choose mean specifically as priceUSD tells us the price of coins, which has high chance to be
countryRegion also has quite a few missing values, but the values in other column for same row might have some impact on dataset thus we did not remove the entire row, instead we labelled it with Unknow country and same approach is followed for platform as well.
teamSize also has many missing values, so removing that many rows did not seem feasible, thus replace the value with median. Mean is not considered as replacement in this case just because teamSize cannot be in decimal and it is better to take the most common teamSize i.e., median for the missing values.

After further observation, we found out that for some columns, endDate column has date before the startDate column, which as per column definition given in problem statement is not possible. This could be a data entry issue where startDate and endDate entries are swapped maybe. Thus, we swapped those specific values.
Columns startDate and endDate are in date format which may not be a great input type to machine learning models we might choose. Thus, to extract information in integer format we

created a calculated column named ‘campaign_duration’ which is calculated by taking difference in end date and start date to give number of days between end date and start date. The swapping of values which we did for start and end date column can also be verified by checking if campaign duration column is negative or not. In Figure 7 we can see there are no rows in our dataset that has negative value, proving our data is consistent for aforementioned columns.

Figure 7

We have seen in wordcloud for platform column that there are values repeating in it, like Ethereum repeating 2 times in word cloud, to resolve this issue we strip any trailing or leading whitespace as well as any non-printable special characters in that column using regex library in python, we can see the difference between unique value in figure 8 after stripping.

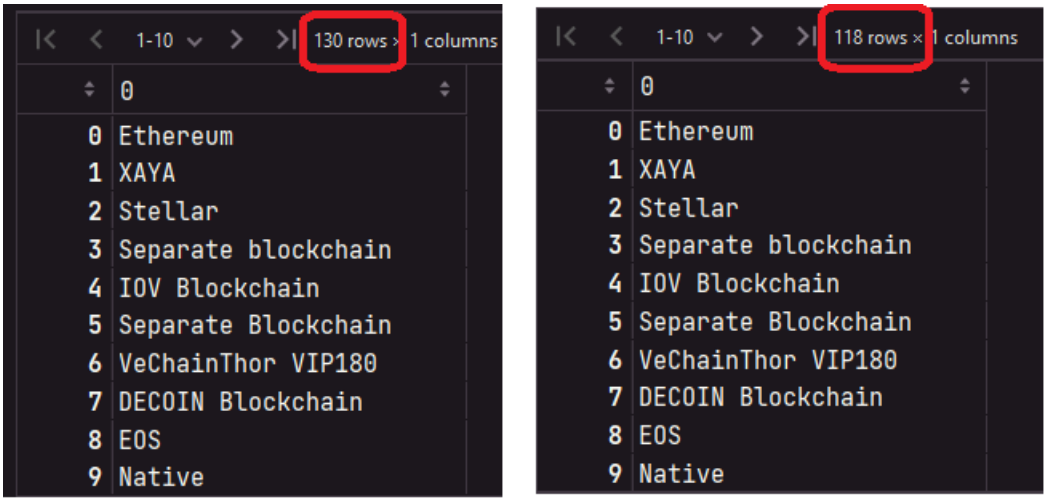

Figure 8

It was also observed that there were some columns which had spelling mistake, eg: there were columns with values ‘Ethererum’, ‘Etherum’ etc and some had short forms of same coin, e.g.: BTC for Bitocin. To handle such issue, we simply replaced the spelling mistakes and the short forms with correct actual names. Some example columns which were replaced is shown in table 4.

Table 4

Existing values	Values replaced with
BTC	Bitcoin
ETH	Ethereum
Etherum	Ethereum
Ethererum	Ethereum
Ethereum, Waves	WavesAndEthereum

After replacing the values, we can see further decrease in the unique values as shown in figure 9. We can safely say that is probably the true number of unique values.

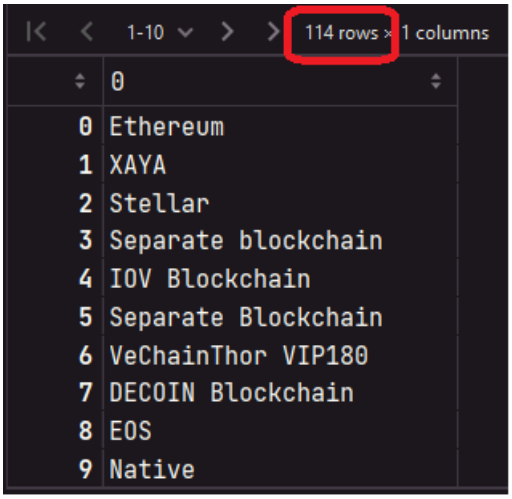

Figure 9

Now let's look at the distributedPercentage column, as per given definition we can observe that it is not a valid value if it has a value greater than 100. Even 100 seems incorrect value but we cannot say that it is not possible. We observed 2 columns which had distributedPercentage has more than 100 value. Below is the snapshot of those rows.

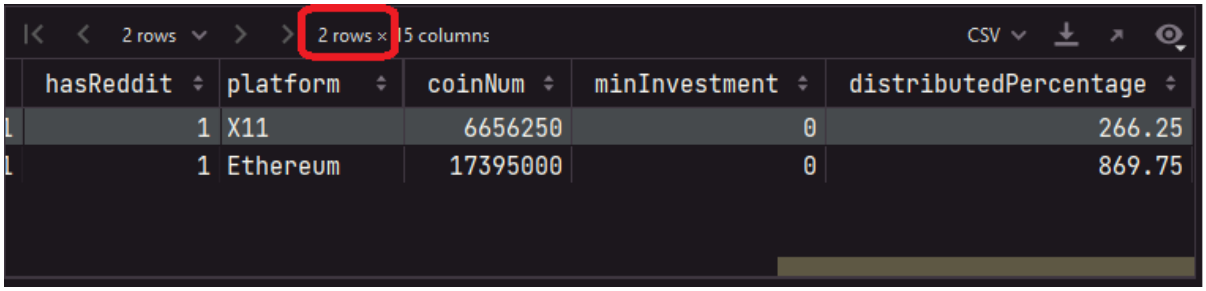

Figure 10

It can be a data entry error, the decimal positioning might have been wrong, but since it is just 2 rows, we just removed it.

Outlier Handling

From the given columns we have essentially 4 numerical columns namely - 'priceUSD', 'teamSize', 'coinNum' and 'distributedPercentage'. We can see box plot for these columns to see the outliers:

1. priceUSD: In the figure 11 we can see that there are some outliers, but 1 outlier value is way above any other outlier.

Figure 11

2. teamsSize: From figure 12, we see that this column have many outliers only in upper range, and there is no outlier in lower range.

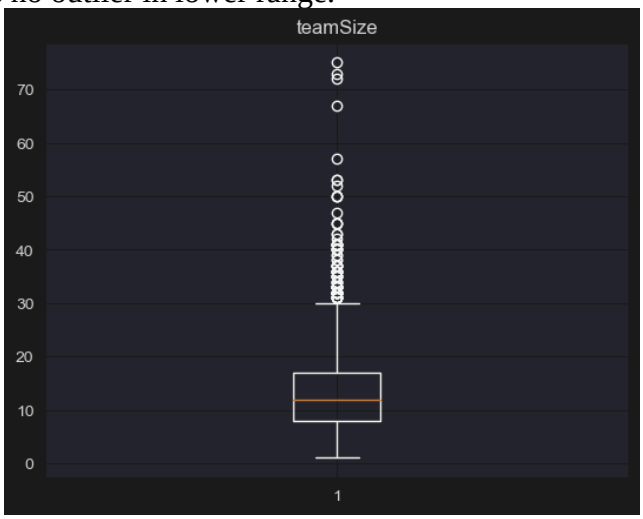

Figure 12

3. coinNum: Here we see the same case as in priceUSD, there is 1 extreme outlier value as compared other outliers.

Figure 13

4. distributedPercentage: This column has fewer outliers as compared to other columns as seen in figure 14.

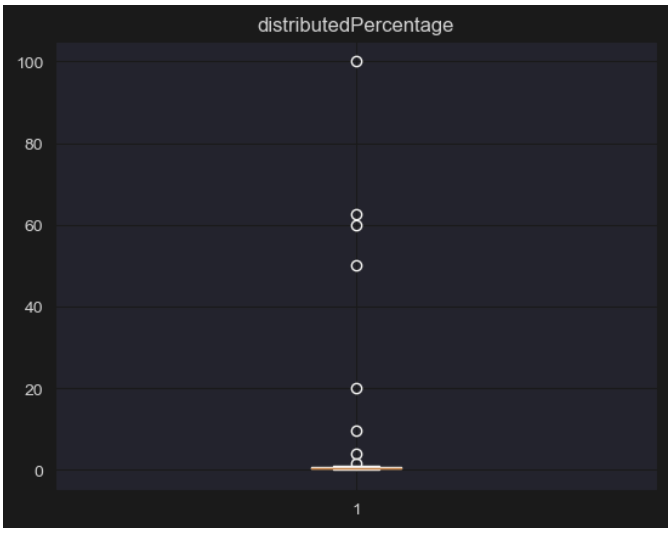

Figure 14

To handle outliers, we used Winsorizing technique, which changes the few percent of extremely high outlier values to near boundary outlier values. After performing Winsorizing we can see in figure for few columns outliers are completely gone while some columns have it. We made change to top and bottom 5% of outliers.

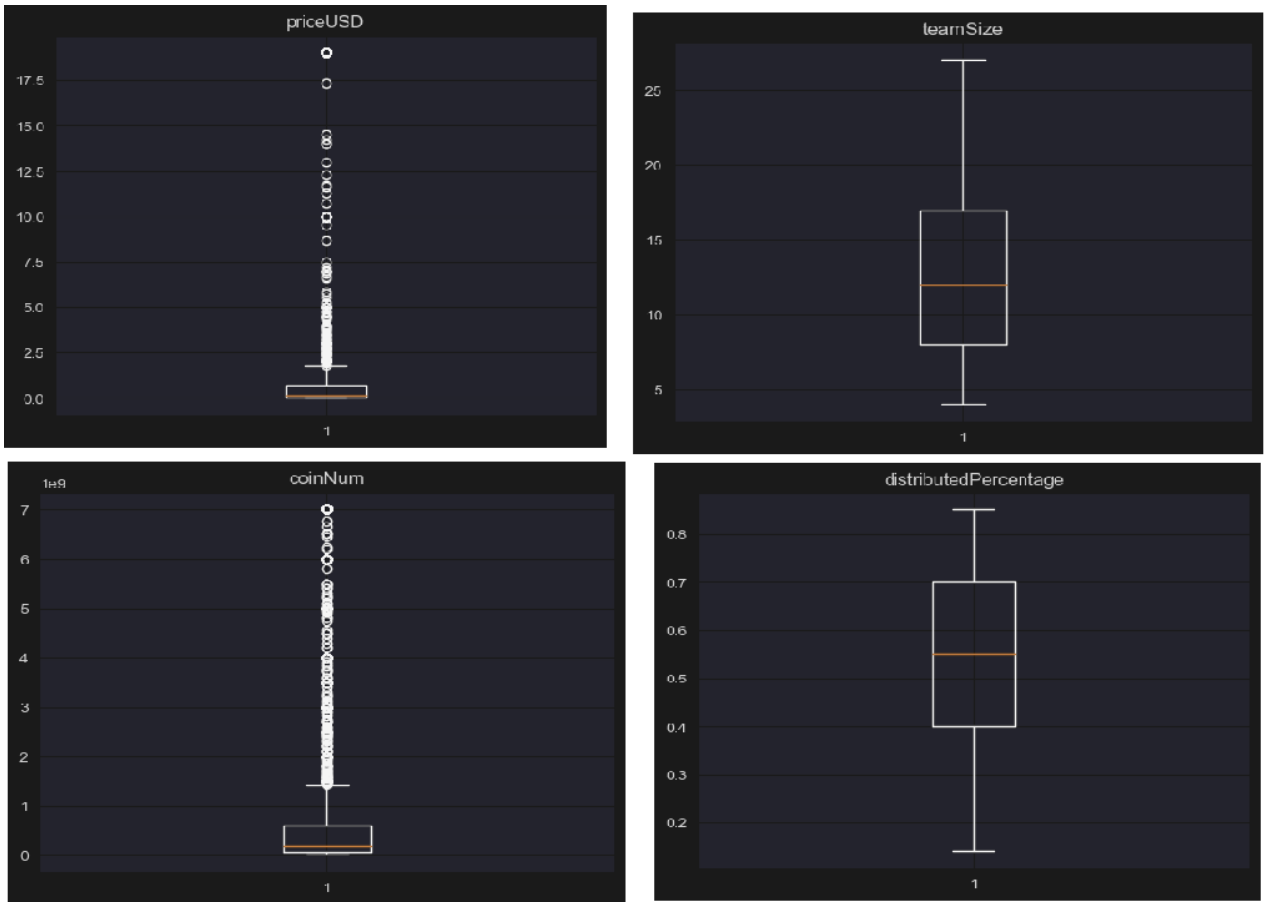

Figure 15

Normalization and One-hot encoding

For some models, the classes/columns value should be scaled appropriately, otherwise it may give inconsistent results. Thus, we need to normalize the data, for the this we used robust scaling technique as it scales the dataset without affecting outlier. The calculations are done based on Inter Quartile range and median. We used sickit-learn RobustScaler library to perform this.

One-hot encoding is a technique which is applied to categorical data, each category is columnized and given a binary value if 1 or 0 based on its presence corresponding to row. In our dataset we performed on-hot encoding for countryRegion and platform column as type of platform and country can be a deciding factor for making predictions on fundraising goal.

Modelling and Evaluation

For modelling and evaluating the models we used various sickit-learn libraries in python. Below we discuss various models applied and evaluate their performance in order to compare and come up with best possible models to help us predict the fundraising goal.

Decision Tree with Adaboosting

In this section we will discuss implementation of decision tree model with Adaboosting to predict whether team/company will reach its fundraising goal. For any classification problem usually decision tree model is the popular choice. Decision tree is machine learning technique used for classification and regression and it predicts the value based on making simple decision rules from the predictor classes/columns to make predictions. Here we are making use of AdaBoosting which is an ensemble method, which trains weak decision trees where each tree involved focuses more on misclassified instances from previous iterations.

To find the best hyperparameters for our decision tree we used grid-search. Grid search is used for hyperparameter tuning, it's a brute force technique which involves evaluating the model with every possible combination of hyperparameters from a pre-defined search space. To perform our model along with adaboosting and grid-search, we used sickit-learn library in python.
We split the data in 60-40 ratio for train and test respectively and will do that for all other models we will discuss, for grid search we provided a list of maximum depths with values [5, 10, 15, 20, 25, 30] to avoid overfitting and finding best possible depth based on model accuracy, to find the best number of estimator for adaboosting we also gave list of estimator for adaboost to grid search, the list has values [50, 100, 150, 200].
The results obtained from running the Decision Tree with Adaboost and GridSearch over 5 K-folds are as follows:

```
## Best Parameters: {'base_estimator__max_depth': 10, 'n_estimators': 200}

## Accuracy Scores: [0.63963964 0.7239819 0.67420814 0.66515837 0.65158371]
## AUC Scores: [0.67055749 0.70308642 0.70758377 0.68130511 0.64303351]
## F1 Scores: [0.44444444 0.55223881 0.56115108 0.48611111 0.34146341]
## Average Accuracy: 0.6709143532672944
## Average AUC: 0.6811132619262701
## Average F1 Score: 0.4770817710593083
##
## evaluation metrics on the test set ##
## Accuracy: 0.6491862567811935
```

```
## AUC: 0.5932770451494076
##
## F1 Score (Class 0): 0.7442882249560633
## F1 Score (Class 1): 0.4414587332053743
```

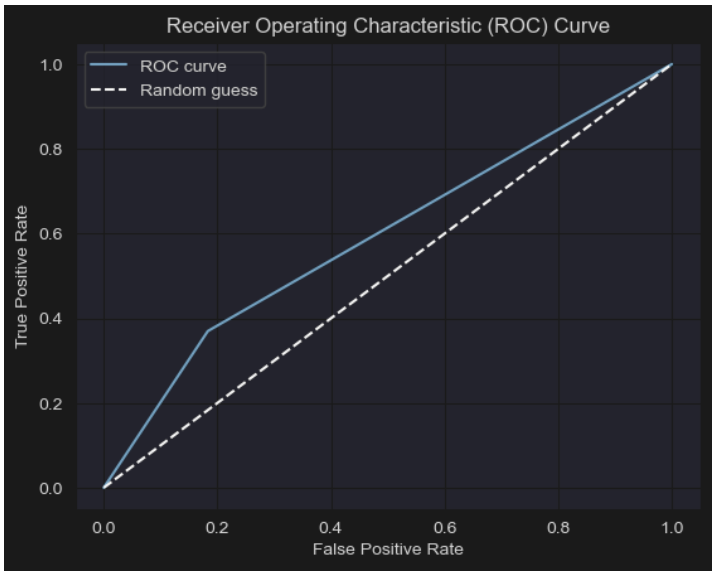

Figure 16: ROC CURVE FOR DECISION TREE

The first line tells us the best combination of parameters found by the GridSearch is a maximum depth of 15 for the base estimator (Decision Tree) and 200 estimators for Adaboost.

In line 2, the accuracy scores for each fold of the cross-validation process is mentioned which range from 0.639 to 0.724, with an average accuracy of 0.67. This indicates that the model can correctly predict the fundraising outcomes for approximately 65% of the cases.

The AUC scores on line 3 for each fold range from 0.643 to 0.707, with an average AUC of 0.68. The AUC score measures the model's ability to discriminate between positive and negative outcomes, with higher values indicating better performance.

The F1 scores for each fold range from 0.341 to 0.561. The F1 score combines precision and recall, providing an overall measure of the model's performance. The average F1 score is 0.477.

Overall, the results suggest that the Decision Tree with Adaboost, using a maximum depth of 10 and 200 estimators, achieves moderate predictive performance. This model has reasonable accuracy and AUC scores, indicating its ability to classify the fundraising outcomes to some extent. However, the F1 scores, particularly for Class 1 (positive outcomes – 'Y' values), are relatively low, suggesting the model requires more improvement.

Random Forest

In this section we will discuss application of Random Forest model for our problem statement. Random Forest is another great machine learning technique which makes use of multiple decision trees outputs and give us the best possible output out of it based on majority voting or averaging. The group of decision trees in random forest can be called as 'forest' is trained through ensemble method likes bagging or bootstrap aggregating.

We also made use of GridSearch to find best estimator/decision trees for random forest to consider and best maximum tree depth like we did in decision tree.
The results obtained from running the Random Forest with Adaboost and GridSearch over 5 K-folds are as follows:

```
## Best Parameters: {'max_depth': 15, 'n_estimators': 150}

## Accuracy Scores: [0.64864865 0.72850679 0.68778281 0.64253394
0.64705882]
## AUC Scores: [0.68632404 0.73747795 0.72037037 0.6686067 0.68236332]
## F1 Scores: [0.39393939 0.55639098 0.4962406 0.44755245 0.384]
## Average Accuracy: 0.670906200317965
## Average AUC: 0.6990284767927044
## Average F1 Score: 0.455624684087842
##
## evaluation metrics on the test set ##
## Accuracy: 0.6528028933092225
## AUC: 0.5871624491871494
## F1 Score (Class 0): 0.7536355859709153
## F1 Score (Class 1): 0.4122448979591837
```

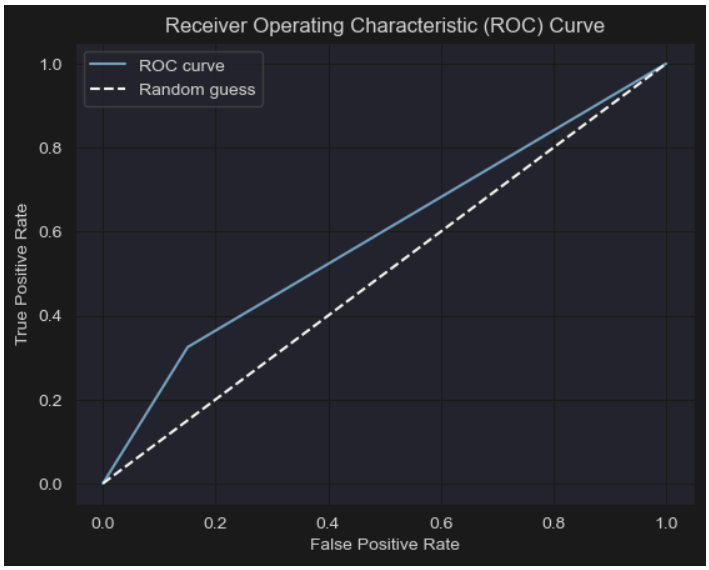

Figure 17: ROC CURVE FOR RANDOM FOREST

Best Parameters: The best parameters found by the grid search are 'max_depth': 15 and 'n_estimators': 150. These parameters define the maximum depth of each tree in the forest and the number of trees in the forest, respectively.

The Random Forest model with grid search achieved moderate accuracy, with an average accuracy of approximately 67.09%. The AUC scores indicate that the model has a reasonable ability to distinguish between successful and unsuccessful fundraising campaigns, with an average AUC of approximately 0.70. The F1 scores suggest that the model performs better in predicting unsuccessful fundraising campaigns (Class 0) compared to successful campaigns (Class 1).

Support vector machine (SVM)

In this section we will discuss application of SVM for our problem statement. SVM is a powerful technique and widely used machine learning algorithm which can handle high dimensional dataset which in our case is produced due to one-hot encoding of countryRegion and platform column.

In SVM as well we made use of GridSearch to get the best possible C and gama values, these are hyperparameters provided to SVM, C value determines the penalty for misclassifying data points in decision boundary and gama is the Kernel coefficient which tells the shape of decision boundary to make. It is important to note that we did not gave 'Kernel' coefficient, thus by default it is taken as rbf and not linear.

The results we get are as follows:

```
## Best Parameters: {'C': 1, 'gamma': 0.1}
##
## Accuracy Scores: [0.63063063 0.68325792 0.63800905 0.62895928
0.63800905]
## AUC Scores: [0.61559233 0.62389771 0.60767196 0.54880952 0.61693122]
## F1 Scores: [0.26785714 0.39655172 0.29824561 0.21153846 0.24528302]
## Average Accuracy: 0.6437731849496555
## Average AUC: 0.6025805480277024
## Average F1 Score: 0.28389519228730953
##
## evaluation metrics on the test set ##
## Accuracy: 0.6298975286317059
## AUC: 0.541495998536466
## F1 Score (Class 0): 0.751417004048583
## F1 Score (Class 1): 0.2759433962264151
```

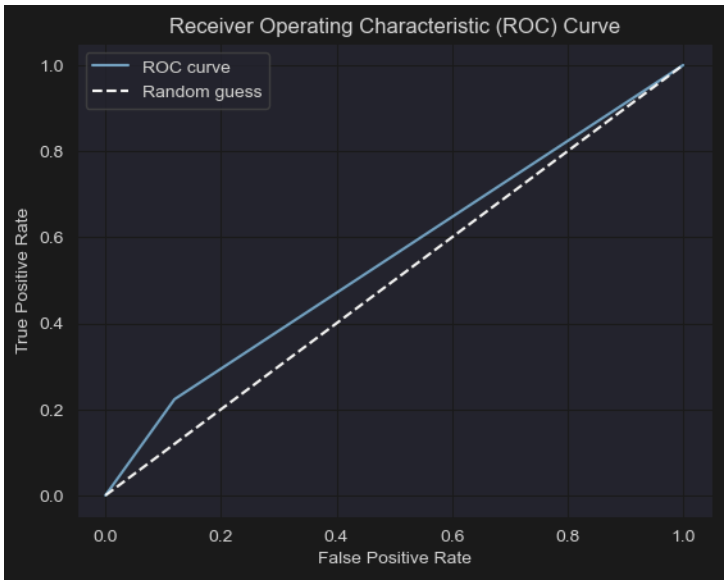

Figure 18: ROC CURVE FOR SVM

The best parameters found by the grid search are 'C': 1 and 'gamma': 0.1. These parameters define the regularization parameter C and the kernel coefficient gamma, respectively.

This model achieved relatively low performance in terms of accuracy, AUC, and F1 score. The average accuracy of the model across all folds is approximately 64.38%. The AUC

scores indicate that the model's ability to distinguish between successful and unsuccessful fundraising campaigns is limited, with an average AUC of approximately 0.60. The F1 scores are also low, suggesting that the model struggles to achieve a balance between precision and recall.

On the test set, the model achieved an accuracy of 0.6298 and an AUC of 0.5415. The F1 score for Class 0 (unsuccessful fundraising) is higher compared to Class 1 (successful fundraising), indicating that the model performs better in predicting successful campaigns.

Overall, the SVM model with grid search did not yield satisfactory results in predicting the success or failure of crowdfunding campaigns if we compare to previous models we discussed.

Artificial Neural Network (ANN)

In this section we will analyse ANN model for our problem statement. ANN is a machine learning technique derived from functioning of our human brain neural structure. It can learn complex pattern and relationship in our data set to predict the target variable.
In ANN, we made use of grid search to find better values of hyperparameters like –

- Hidden layer size – it represents number of neurons in the specific hidden layer, provided following options to choose from [(100,), (50, 50), (25, 25, 25)]
- Acivation – the type of activation, choices set were {relu, tanh}
- Alpha – It's the strength of L2 regularization (Ridge Regression) term. Provided choices - [0.0001, 0.001, 0.01]

Let's see the results below:

```
## Best Parameters: {'activation': 'relu', 'alpha': 0.001,
'hidden_layer_sizes': (50, 50)}
##
## Accuracy Scores: [0.62162162 0.66063348 0.64705882 0.59276018
0.60180995]
## AUC Scores: [0.64355401 0.65582011 0.69294533 0.67178131 0.65564374]
## F1 Scores: [0.43589744 0.5170068 0.52071006 0.40268456 0.45714286]
## Average Accuracy: 0.6247768130121072
## Average AUC: 0.6639488966318235
## Average F1 Score: 0.46668834373827367
##
## evaluation metrics on the test set ##
## Accuracy: 0.6021699819168174
## AUC: 0.5830129268512001
## F1 Score (Class 0): 0.6745562130177515
## F1 Score (Class 1): 0.48837209302325585
```

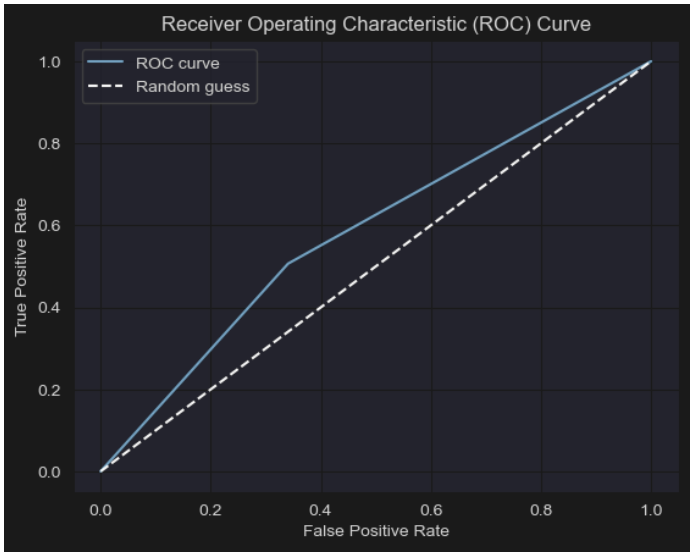

Figure 19: ROC CURVE FOR ANN

The ANN model with grid search achieved moderate performance in terms of accuracy, AUC, and F1 score. The average accuracy of the model across all folds is approximately 62.48%. The AUC scores indicate that the model has some ability to distinguish between successful and unsuccessful fundraising campaigns, with an average AUC of approximately 0.66. The F1 scores seems to be moderate, suggesting a balance between precision and recall.

On the test set, the model achieved an accuracy of 0.6 and an AUC of 0.58. The F1 score for Class 0 (unsuccessful fundraising) is relatively higher compared to Class 1 (successful fundraising), indicating that the model performs better in predicting successful campaigns.

Conclusion

Based on the results discussed in previous section we see that Random Forest has the highest average accuracy of 67%, while decision tree with Adaboosting is at near 65% and SVM at 64%, complex models like ANN achieved only 62%, suggesting decision trees-based model might be better. If we see the overall accuracy of test set as well, we again find that Random Forest and Addaboosting decision tree performed better. While ANN performed well.

F1 score and AUC score are also important factors for evaluation as we saw in evaluation of models. And it was found that SVM despite having higher accuracy, falls short when compared to even ANN, as we saw that in SVM the F1 score for successful predictions was only 0.27 which shows big imbalance in successful/unsuccessful predictions. And AUC score also was least for SVM, suggesting that overall, this model might not be good for our use case. While Random Forest and decision tree with adabost performed equally well.

The overall performance of both Random Forest and adaboost decision tree can further be increased by further hypertuning parameters, even ANN might show improvement if we manage to find specific parameters, but SVM might not be a good option for prediction.

The results do suggest that machine learning models can provide valuable insights and predictions for ICO success.

Appendix

Python Code used:

```
import pandas as pd
import numpy as np
import seaborn as sns
import missingno as msno
import matplotlib.pyplot as plt
%matplotlib inline
#Read csv file without ID column
ico_df =
pd.read_csv("..\LUBS5990M_courseworkData_202223.csv",usecols=lambda x: x
not in ["ID","brandSlogan"])
# Categorical variables of interest
categorical_vars = ['success','hasVideo', 'hasGithub', 'hasReddit']

# Plotting bar charts for each categorical variable
for var in categorical_vars:
    plt.figure(figsize=(10, 6))
    ico_df[var].value_counts().plot(kind='bar')
    plt.xlabel(var)
    plt.ylabel('Count')
    plt.title(f'Distribution of {var}')
    plt.xticks(rotation=90)
    plt.show()

from wordcloud import WordCloud
# Selecting the 'countryRegion' and 'platform' columns
country_region_data = ico_df['countryRegion'].dropna().astype(str)
platform_data = ico_df['platform'].dropna().astype(str)
# Creating a WordCloud for 'countryRegion'
plt.figure(figsize=(10, 6))
wordcloud_country_region = WordCloud(width=800, height=400,
background_color='white').generate(' '.join(country_region_data))
plt.imshow(wordcloud_country_region, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud - countryRegion')
plt.show()

# Creating a WordCloud for 'platform'
plt.figure(figsize=(10, 6))
wordcloud_platform = WordCloud(width=800, height=400,
background_color='white').generate(' '.join(platform_data))
plt.imshow(wordcloud_platform, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud - platform')
plt.show()

# Counting the number of empty strings in each column
empty_string_counts = ico_df.eq(' ').sum()

# Creating a bar plot to visualize the counts
plt.figure(figsize=(10, 6))
empty_string_counts.plot(kind='bar')
plt.xlabel('Columns')
plt.ylabel('Count of Empty Strings')
plt.title('Number of Empty Strings in DataFrame')
plt.xticks(rotation=90)
plt.show()
```

```
#Getting an overview of data
ico_df.head()

#Checking data types of each columns
ico_df.info()

#converting some columns to pandas category variable and displaying them
ico_df = ico_df.astype({'hasVideo':'category',
'hasGithub':'category','hasReddit':'category','minInvestment':'category','r
ating':'category','success':'category'})
ico_df.info()

#Converting to date/time variable
ico_df['startDate'] = pd.to_datetime(ico_df['startDate'],format='%d/%m/%Y')
ico_df['endDate'] = pd.to_datetime(ico_df['endDate'],format='%d/%m/%Y')
ico_df.info()

#Replace null strings with NA values in entire dataset
ico_df = ico_df.replace(r'^\s*$', np.nan, regex=True)

#Clean more rows after analysis.
ico_df_clean = ico_df.copy()
#priceUSD N/A values replaced with mean
ico_df_clean['priceUSD'].fillna(ico_df_clean['priceUSD'].mean(),
inplace=True)
#countryRegion N/A values replaced with Unknown country
ico_df_clean['countryRegion'].fillna(value='Unknown', inplace=True)
#teamSize N/A values replaced with median
ico_df_clean['teamSize'].fillna(ico_df_clean['teamSize'].median(),
inplace=True)
#platform N/A values replaced with Unknown platform
ico_df_clean['platform'].fillna(value='Unknown', inplace=True)

#Create new column to calculate duration
ico_df_clean['campaign_duration'] = (ico_df_clean['endDate'] -
ico_df_clean['startDate']).dt.days

# Identify rows with negative campaign duration
negative_duration_mask = ico_df_clean['campaign_duration'] < 0

# Swap start date and end date using a temporary column
ico_df_clean.loc[negative_duration_mask, 'tempDate'] =
ico_df_clean.loc[negative_duration_mask, 'startDate']
ico_df_clean.loc[negative_duration_mask, 'startDate'] =
ico_df_clean.loc[negative_duration_mask, 'endDate']
ico_df_clean.loc[negative_duration_mask, 'endDate'] =
ico_df_clean.loc[negative_duration_mask, 'tempDate']

# Remove the temporary column
ico_df_clean.drop('tempDate', axis=1, inplace=True)

# Update campaign duration for the swapped rows
ico_df_clean['campaign_duration'] = (ico_df_clean['endDate'] -
ico_df_clean['startDate']).dt.days

# Convert 'startDate' and 'endDate' columns back to datetime format
ico_df_clean['startDate'] = pd.to_datetime(ico_df_clean['startDate'])
ico_df_clean['endDate'] = pd.to_datetime(ico_df_clean['endDate'])

#check now for -ve campaign duration
ico_df_clean[ico_df_clean['campaign_duration'] < 0]
```

```
#check unique values now
ico_df_clean.platform.unique()

import re

# Convert the column to string type
ico_df_clean['platform'] = ico_df_clean['platform'].astype(str)
#Remove white spaces after the string value in platform column
ico_df_clean['platform'] = ico_df_clean['platform'].str.strip()

# Remove non-printable or special characters using regular expression
ico_df_clean['platform'] = ico_df_clean['platform'].apply(lambda x:
re.sub(r'[^\x20-\x7E]', '', x))

#check unique values now
ico_df_clean.platform.unique()

#group similar looking values
crypto_mapping = {
    'BTC':'Bitcoin',
    'Bitcoin/Bitcoin':'Bitcoin',
    'ETH': 'Ethereum',
    'Etherum': 'Ethereum',
    'Ethererum': 'Ethereum',
    'Ethereum, Waves': 'WavesAndEthereum'
}

# Create a regular expression pattern from the dictionary keys
pattern = re.compile('|'.join(map(re.escape, crypto_mapping.keys())))

# Replace the values in the DataFrame column
ico_df_clean['platform'] = ico_df_clean['platform'].str.replace(pattern,
lambda m: crypto_mapping[m.group(0)])

#check unique values now
ico_df_clean.platform.unique()

#distributedPercentage cannot be more than 100, thus remove rows with
distributedPercentage > 100
ico_df_clean = ico_df_clean[ico_df_clean['distributedPercentage'] <= 100]
#Removed 2 rows

#Plot the number of N/A values in each column
ico_df.isna().sum().plot(kind='bar')
plt.show

#CHECKING OUTLIERS
# Select the numerical columns in ico_df_clean
numerical_columns = ['priceUSD', 'teamSize', 'coinNum',
'distributedPercentage']

# Create box plots for each numerical column
for column in numerical_columns:
    plt.figure()
    plt.boxplot(ico_df_clean[column])
    plt.title(column)
    plt.show()

from scipy.stats.mstats import winsorize

ico_df_final = ico_df_clean.copy()
```

```
# Apply Winsorization to the numerical columns
for column in numerical_columns:
    ico_df_final[column] = winsorize(ico_df_final[column], limits=[0.05,
0.05])

# Create box plots for each numerical column
for column in numerical_columns:
    plt.figure()
    plt.boxplot(ico_df_final[column])
    plt.title(column)
    plt.show()

ico_df_normOut = ico_df_final.copy()

from sklearn.preprocessing import RobustScaler

# Create a MinMaxScaler object
scaler = RobustScaler()

# Fit and transform the numerical columns
ico_df_normOut[numerical_columns] =
scaler.fit_transform(ico_df_normOut[numerical_columns])

ico_df_norm = ico_df_clean.copy()

from sklearn.preprocessing import RobustScaler

# Create a MinMaxScaler object
scaler = RobustScaler()

# Fit and transform the numerical columns
ico_df_norm[numerical_columns] =
scaler.fit_transform(ico_df_norm[numerical_columns])

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,f1_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score, StratifiedKFold

# Create separate encoders for 'countryRegion' and 'platform'
country_encoder = OneHotEncoder(sparse_output=False)
platform_encoder = OneHotEncoder(sparse_output=False)

# Fit and transform the 'countryRegion' column
country_encoded =
country_encoder.fit_transform(ico_df_final[['countryRegion']])

# Fit and transform the 'platform' column
platform_encoded =
platform_encoder.fit_transform(ico_df_final[['platform']])

# Convert the encoded arrays into a DataFrame
country_encoded_df = pd.DataFrame(country_encoded,
columns=country_encoder.categories_[0])
platform_encoded_df = pd.DataFrame(platform_encoded,
columns=platform_encoder.categories_[0])
```

```
# Reset the index of the encoded columns and original dataframe
country_encoded_df.reset_index(drop=True, inplace=True)
platform_encoded_df.reset_index(drop=True, inplace=True)
ico_df_final.reset_index(drop=True, inplace=True)

# Concatenate the encoded DataFrames with the original DataFrame
ico_df_encoded = pd.concat([ico_df_final, country_encoded_df,
platform_encoded_df], axis=1)

# Remove the original 'countryRegion' and 'platform' columns
ico_df_encoded.drop(['countryRegion', 'platform','startDate','endDate'],
axis=1, inplace=True)

# Create the feature matrix (X) and the target variable (y)
X = ico_df_encoded.drop('success',axis=1)
y = ico_df_encoded['success']

# Encode the target variable 'success'
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Define the number of folds for k-fold cross-validation
n_splits = 5

# Define the train/test split ratios
train_test_splits = [0.6]  # Example split ratios, adjust as needed

# Define the max_depth values to evaluate
max_depth_values = [1, 5, 10, 15, 20, 25, 30] # Example max_depth values,
adjust as needed

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
random_state=42)

# Define the parameter grid for grid search or random search
param_grid = {
    'base_estimator__max_depth': [5, 10, 15, 20, 25, 30],  # Specify the
range of max_depth values
    'n_estimators': [50, 100, 150, 200]  # Specify the range of the number
of estimators
}

# Create a decision tree classifier as the base estimator
dt_classifier = DecisionTreeClassifier()

# Create an AdaBoost classifier
ada_boost = AdaBoostClassifier(base_estimator=dt_classifier)

# Perform grid search or random search
#Grid Search:
search = GridSearchCV(ada_boost, param_grid, scoring='accuracy', cv=5)

# Random Search:
#search = RandomizedSearchCV(ada_boost, param_grid, scoring='accuracy',
cv=5)

# Fit the search object to the training data
search.fit(X_train, y_train)

# Get the best estimator and its associated parameters
```

```
best_ada_boost = search.best_estimator_
best_params = search.best_params_

print("Best Parameters:", best_params)

# Use k-fold cross-validation to evaluate the model's performance
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
accuracy_scores = cross_val_score(best_ada_boost, X_train, y_train, cv=cv,
scoring='accuracy')
auc_scores = cross_val_score(best_ada_boost, X_train, y_train, cv=cv,
scoring='roc_auc')
f1_scores = cross_val_score(best_ada_boost, X_train, y_train, cv=cv,
scoring='f1')

# Print the evaluation scores for each fold
print("Accuracy Scores:", accuracy_scores)
print("AUC Scores:", auc_scores)
print("F1 Scores:", f1_scores)

# Calculate the average evaluation scores across all folds
avg_accuracy = np.mean(accuracy_scores)
avg_auc = np.mean(auc_scores)
avg_f1 = np.mean(f1_scores)

print("Average Accuracy:", avg_accuracy)
print("Average AUC:", avg_auc)
print("Average F1 Score:", avg_f1)

# Use the best classifier to make predictions on the test set
y_pred = best_ada_boost.predict(X_test)

# Calculate evaluation metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
f1_class0 = f1_score(y_test, y_pred, pos_label=0)
f1_class1 = f1_score(y_test, y_pred, pos_label=1)

print("Accuracy:", accuracy)
print("AUC:", auc)
print("F1 Score (Class 0):", f1_class0)
print("F1 Score (Class 1):", f1_class1)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='best')
plt.show()


from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import auc

# Create separate encoders for 'countryRegion' and 'platform'
country_encoder = OneHotEncoder(sparse_output=False)
platform_encoder = OneHotEncoder(sparse_output=False)
```

```
# Fit and transform the 'countryRegion' column
country_encoded =
country_encoder.fit_transform(ico_df_final[['countryRegion']])

# Fit and transform the 'platform' column
platform_encoded =
platform_encoder.fit_transform(ico_df_final[['platform']])

# Convert the encoded arrays into a DataFrame
country_encoded_df = pd.DataFrame(country_encoded,
columns=country_encoder.categories_[0])
platform_encoded_df = pd.DataFrame(platform_encoded,
columns=platform_encoder.categories_[0])

# Reset the index of the encoded columns and original dataframe
country_encoded_df.reset_index(drop=True, inplace=True)
platform_encoded_df.reset_index(drop=True, inplace=True)
ico_df_final.reset_index(drop=True, inplace=True)

# Concatenate the encoded DataFrames with the original DataFrame
ico_df_encoded = pd.concat([ico_df_final, country_encoded_df,
platform_encoded_df], axis=1)

# Remove the original 'countryRegion' and 'platform' columns
ico_df_encoded.drop(['countryRegion', 'platform','startDate','endDate'],
axis=1, inplace=True)

# Create the feature matrix (X) and the target variable (y)
X = ico_df_encoded.drop('success',axis=1)
y = ico_df_encoded['success']

# Encode the target variable 'success'
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
random_state=42)

# Define the parameter grid for grid search
param_grid = {
    'max_depth': [5, 10, 15, 20, 25, 30],  # Specify the range of max_depth
values
    'n_estimators': [50, 100, 150, 200]  # Specify the range of the number
of estimators
}

# Create a random forest classifier
rf_classifier = RandomForestClassifier()

# Perform grid search
search = GridSearchCV(rf_classifier, param_grid, cv=5, scoring='accuracy')

# Fit the search object to the training data
search.fit(X_train, y_train)

# Get the best estimator and its associated parameters
best_rf = search.best_estimator_
best_params = search.best_params_
```

```
print("Best Parameters:", best_params)

# Use k-fold cross-validation to evaluate the model's performance
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
accuracy_scores = cross_val_score(best_rf, X_train, y_train, cv=cv,
scoring='accuracy')
auc_scores = cross_val_score(best_rf, X_train, y_train, cv=cv,
scoring='roc_auc')
f1_scores = cross_val_score(best_rf, X_train, y_train, cv=cv, scoring='f1')

# Print the evaluation scores for each fold
print("Accuracy Scores:", accuracy_scores)
print("AUC Scores:", auc_scores)
print("F1 Scores:", f1_scores)

# Calculate the average evaluation scores across all folds
avg_accuracy = np.mean(accuracy_scores)
avg_auc = np.mean(auc_scores)
avg_f1 = np.mean(f1_scores)

print("Average Accuracy:", avg_accuracy)
print("Average AUC:", avg_auc)
print("Average F1 Score:", avg_f1)

# Use the best classifier to make predictions on the test set
y_pred = best_rf.predict(X_test)

# Calculate evaluation metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
f1_class0 = f1_score(y_test, y_pred, pos_label=0)
f1_class1 = f1_score(y_test, y_pred, pos_label=1)

print("Accuracy:", accuracy)
print("AUC:", auc)
print("F1 Score (Class 0):", f1_class0)
print("F1 Score (Class 1):", f1_class1)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='best')
plt.show()


from sklearn.svm import SVC

# Fit and transform the 'countryRegion' column
country_encoded =
country_encoder.fit_transform(ico_df_normOut[['countryRegion']])

# Fit and transform the 'platform' column
platform_encoded =
platform_encoder.fit_transform(ico_df_normOut[['platform']])
```

```
# Convert the encoded arrays into a DataFrame
country_encoded_df = pd.DataFrame(country_encoded,
columns=country_encoder.categories_[0])
platform_encoded_df = pd.DataFrame(platform_encoded,
columns=platform_encoder.categories_[0])

# Reset the index of the encoded columns and original dataframe
country_encoded_df.reset_index(drop=True, inplace=True)
platform_encoded_df.reset_index(drop=True, inplace=True)
ico_df_normOut.reset_index(drop=True, inplace=True)

# Concatenate the encoded DataFrames with the original DataFrame
ico_df_encoded = pd.concat([ico_df_normOut, country_encoded_df,
platform_encoded_df], axis=1)

# Remove the original 'countryRegion' and 'platform' columns
ico_df_encoded.drop(['countryRegion', 'platform','startDate','endDate'],
axis=1, inplace=True)

# Create the feature matrix (X) and the target variable (y)
X = ico_df_encoded.drop('success',axis=1)
y = ico_df_encoded['success']

# Encode the target variable 'success'
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
random_state=42)

# Define the parameter grid for grid search or random search
param_grid = {
    'C': [0.1, 1, 10],  # Specify the range of C values
    'gamma': [0.1, 1, 10]  # Specify the range of gamma values
}

# Create an SVM classifier
svm_classifier = SVC()

# Perform grid search
search = GridSearchCV(svm_classifier, param_grid, cv=5)

# Fit the search object to the training data
search.fit(X_train, y_train)

# Get the best estimator and its associated parameters
best_svm = search.best_estimator_
best_params = search.best_params_

print("Best Parameters:", best_params)

# Use k-fold cross-validation to evaluate the model's performance
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
accuracy_scores = cross_val_score(best_svm, X_train, y_train, cv=cv,
scoring='accuracy')
auc_scores = cross_val_score(best_svm, X_train, y_train, cv=cv,
scoring='roc_auc')
f1_scores = cross_val_score(best_svm, X_train, y_train, cv=cv,
scoring='f1')
```

```
# Print the evaluation scores for each fold
print("Accuracy Scores:", accuracy_scores)
print("AUC Scores:", auc_scores)
print("F1 Scores:", f1_scores)

# Calculate the average evaluation scores across all folds
avg_accuracy = np.mean(accuracy_scores)
avg_auc = np.mean(auc_scores)
avg_f1 = np.mean(f1_scores)

print("Average Accuracy:", avg_accuracy)
print("Average AUC:", avg_auc)
print("Average F1 Score:", avg_f1)

# Use the best classifier to make predictions on the test set
y_pred = best_svm.predict(X_test)

# Calculate evaluation metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
f1_class0 = f1_score(y_test, y_pred, pos_label=0)
f1_class1 = f1_score(y_test, y_pred, pos_label=1)

print("Accuracy:", accuracy)
print("AUC:", auc)
print("F1 Score (Class 0):", f1_class0)
print("F1 Score (Class 1):", f1_class1)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='best')
plt.show()


from sklearn.neural_network import MLPClassifier

# Fit and transform the 'countryRegion' column
country_encoded =
country_encoder.fit_transform(ico_df_normOut[['countryRegion']])

# Fit and transform the 'platform' column
platform_encoded =
platform_encoder.fit_transform(ico_df_normOut[['platform']])

# Convert the encoded arrays into a DataFrame
country_encoded_df = pd.DataFrame(country_encoded,
columns=country_encoder.categories_[0])
platform_encoded_df = pd.DataFrame(platform_encoded,
columns=platform_encoder.categories_[0])

# Reset the index of the encoded columns and original dataframe
country_encoded_df.reset_index(drop=True, inplace=True)
platform_encoded_df.reset_index(drop=True, inplace=True)
ico_df_normOut.reset_index(drop=True, inplace=True)
```

```
# Concatenate the encoded DataFrames with the original DataFrame
ico_df_encoded = pd.concat([ico_df_normOut, country_encoded_df,
platform_encoded_df], axis=1)

# Remove the original 'countryRegion' and 'platform' columns
ico_df_encoded.drop(['countryRegion', 'platform','startDate','endDate'],
axis=1, inplace=True)

# Create the feature matrix (X) and the target variable (y)
X = ico_df_encoded.drop('success',axis=1)
y = ico_df_encoded['success']

# Encode the target variable 'success'
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Define the number of folds for k-fold cross-validation
n_splits = 5

# Define the train/test split ratios
# train_test_splits = [0.6, 0.7]  # Example split ratios, adjust as needed

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
random_state=42)

# Define the parameter grid for grid search or random search
param_grid = {
    'hidden_layer_sizes': [(100,), (50, 50), (25, 25, 25)],  # Specify
different hidden layer configurations
    'activation': ['relu', 'tanh'],  # Specify different activation
functions
    'alpha': [0.0001, 0.001, 0.01]  # Specify different regularization
strengths
}

# Create an ANN classifier
ann_classifier = MLPClassifier(max_iter=1000)

# Perform grid search
search = GridSearchCV(ann_classifier, param_grid, cv=5)

# Fit the search object to the training data
search.fit(X_train, y_train)

# Get the best estimator and its associated parameters
best_ann = search.best_estimator_
best_params = search.best_params_

print("Best Parameters:", best_params)

# Use k-fold cross-validation to evaluate the model's performance
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
accuracy_scores = cross_val_score(best_ann, X_train, y_train, cv=cv,
scoring='accuracy')
auc_scores = cross_val_score(best_ann, X_train, y_train, cv=cv,
scoring='roc_auc')
f1_scores = cross_val_score(best_ann, X_train, y_train, cv=cv,
scoring='f1')
```

```
# Print the evaluation scores for each fold
print("Accuracy Scores:", accuracy_scores)
print("AUC Scores:", auc_scores)
print("F1 Scores:", f1_scores)

# Calculate the average evaluation scores across all folds
avg_accuracy = np.mean(accuracy_scores)
avg_auc = np.mean(auc_scores)
avg_f1 = np.mean(f1_scores)

print("Average Accuracy:", avg_accuracy)
print("Average AUC:", avg_auc)
print("Average F1 Score:", avg_f1)

# Use the best classifier to make predictions on the test set
y_pred = best_ann.predict(X_test)

# Calculate evaluation metrics on the test set
accuracy = accuracy_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
f1_class0 = f1_score(y_test, y_pred, pos_label=0)
f1_class1 = f1_score(y_test, y_pred, pos_label=1)

print("Accuracy:", accuracy)
print("AUC:", auc)
print("F1 Score (Class 0):", f1_class0)
print("F1 Score (Class 1):", f1_class1)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='best')
plt.show()
```