Leeds University Business School

UNIVERSITY OF LEEDS

Assessed Coursework Coversheet

For use with *individual* assessed work

Student ID Number:	2	0	1	6	7	9	5	0	2
Module Code:	LUBS5990M								
Module Title:	Machine Learning in Practice								
Module Leader:	Dr. Xingjie Wei								
Declared Word Count:	3452								

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).

Coursework

UNIVERSITY OF LEEDS

Machine Learning in Practice

Student Id: 201679502

Course: LUBS5990M Machine Learning in Practice

Date of Submission: 18/05/2023

1. Introduction

In this coursework, the main objective is to determine whether a company or team will successfully reach its fundraising goals through an Initial Coin Offering (ICO) using a machine learning model. Crowdfunding has become increasingly popular as a method of obtaining funds for projects or companies from individuals. ICOs, which are a variation of crowdfunding, allow companies to raise funds by issuing and selling digital coins based on blockchain technology.

This approach differs from traditional crowdfunding campaigns as it provides investors with the opportunity to acquire digital coins, commonly known as cryptocurrencies. These coins can be traded among investors and may serve practical purposes within the product or service offered by the fundraising team or company. ICOs typically follow an all-or-nothing approach, wherein the fundraising team sets a specific funding goal. The success of the ICO hinges on whether this goal is achieved within the specified timeframe.

In this coursework, machine learning models are built to predict the success of a company or team in achieving their fundraising goal through an ICO campaign. These models will evaluate various features and characteristics of the fundraising teams and projects to determine the likelihood of reaching the fundraising goal. By utilizing advanced machine learning algorithms, we aim to gain insights into the factors that contribute to a successful ICO campaign.

At the end of this coursework, we will not only provide valuable predictive models but also offer meaningful insights to fundraising teams and companies. With this knowledge, they can make informed decisions, optimize their strategies, and increase their chances of achieving their fundraising goals. Ultimately, this work aims to enhance the efficiency and effectiveness of ICO fundraising campaigns and contribute to a broader understanding of the crowdfunding landscape.

2. Data understanding

2.1 Dataset Overview

The dataset contains 2767 observations of various fundraising teams ICO campaigns, with a total of 16 attributes. There are both categories and numerical columns among these attributes. There are ten numerical columns: *ID*, *hasVideo*, *rating*, *priceUSD*, *teamSize*, *hasGithub*, *hasReddit*, *coinNum*, *minInvestment*, and *distributedPercentage*. The *ID* column is a unique identifier for each row. The columns *minInvestment*, *hasVideo*, *hasGithub*, and *hasReddit* have binary values of 0 or 1, indicating the presence or absence of a specific feature. The remaining numerical variables have separate unique values within their respective columns.

In addition to the numerical columns, the dataset contains six categorical variables: *success*, *brandSlogan*, *countryRegion*, *startDate*, *endDate*, and *platform*. The success column indicates the outcome of each fundraising work and has two values:

'Y' or 'N', indicating success or failure. The geographical region associated with each project is indicated by the *countryRegion* column. The *startDate* and *endDate* columns contain information about each fundraising campaign's start and end dates. The *platform* field indicates the fundraising platform. Finally, each fundraising team's slogan is listed in the *brandSlogan* column.

This dataset provides important insights into ICO initiatives by considering both numerical and categorical features, allowing for comprehensive analysis and exploration of various factors related to fundraising teams.

2.2 Data Quality

The dataset contains null values in three columns. The *priceUSD* column has a null value percentage of 6.51%. The *countryRegion* column has a null value percentage of 2.57%, and the *teamSize* column has a null value percentage of 5.57%. Overall, the percentage of null values in the dataset is 13.55%. Furthermore, there are some observations in the *priceUSD* column with a value of 0, which is considered a data error. The percentage of such occurrences in the *priceUSD* column is 5.49%. There are no duplicates present in the dataset.

In the categorical column *countryRegion*, there are strings that have the same country name but differ in case sensitivity. Additionally, some country names contain special characters, such as *ç*. In the *platform* column, the strings have extra spaces, and some platforms are misspelled or represented by shortcuts. Moreover, there are instances where the *platform* column contains an empty string.

2.3 Statistical Summaries

The statistical summaries of a few numeric columns were calculated. The mean of the *rating* variable is 3.12. The first, second, and third quartiles of the *rating* are 2.6, 3.1, and 3.7, respectively. This indicates that the mean is approximately equal to the median, suggesting that the data is evenly distributed.

The mean of the *priceUSD* variable is 19.01, and the first, second, and third quartiles are 0.04, 0.12, and 0.5, respectively. The mean is greater than the third quartile, indicating the presence of outliers in the data.

The mean of the *teamSize* variable is 13.10, and the first, second, and third quartiles are 7, 12, and 17, respectively. The mean is close to the second quartile, suggesting that the data is evenly distributed around the median.

The mean of the *distributedPercentage* variable is 1.06, and the first, second, and third quartiles are 0.4, 0.55, and 0.7, respectively. The mean is greater than the second quartile, indicating the presence of outliers in the data.

The mean of the *coinNum* variable is 8.177888e+12, and the first, second, and third quantile are 5.000000e+07, 1.800000e+08 and 6.000000e+08, respectivelty. The mean is greater than the third quartile, indicating the presence of outliers in the data.

2.4 Data Visualisation

The histograms were plotted for all the numerical variables in the dataset. The histogram of the *rating* variable suggests that it follows a normal distribution, with a majority of records concentrated around a rating of three. There are very few records with low ratings and high ratings as well. The boxplot of the *rating* variable confirms that there are no outliers present in the data as shown in figure 1

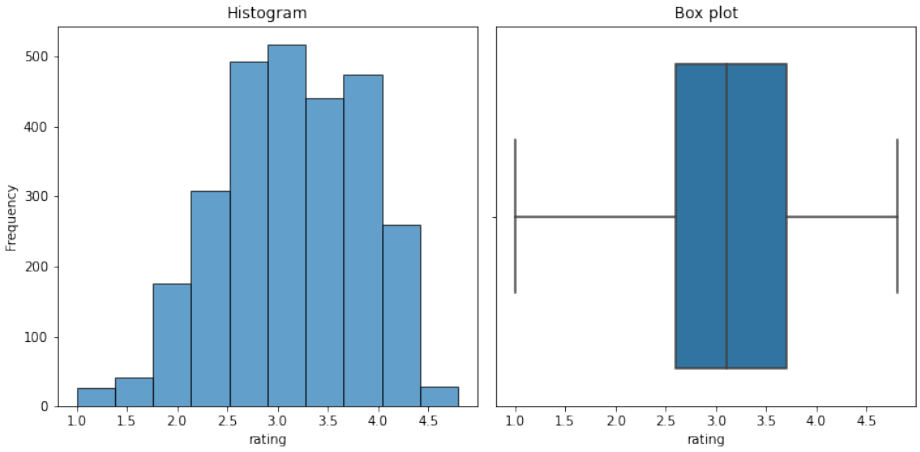

Figure 1: Histogram and Boxplot of rating variable.

The frequency histogram of the *priceUSD* variable exhibits a long tail on the left side, indicating the presence of outliers in the data as shown in figure 2. The boxplot of *priceUSD* also reveals the presence of extreme outliers. However, due to the presence of these outliers, the box plot may not be the most appropriate visualization for this variable.

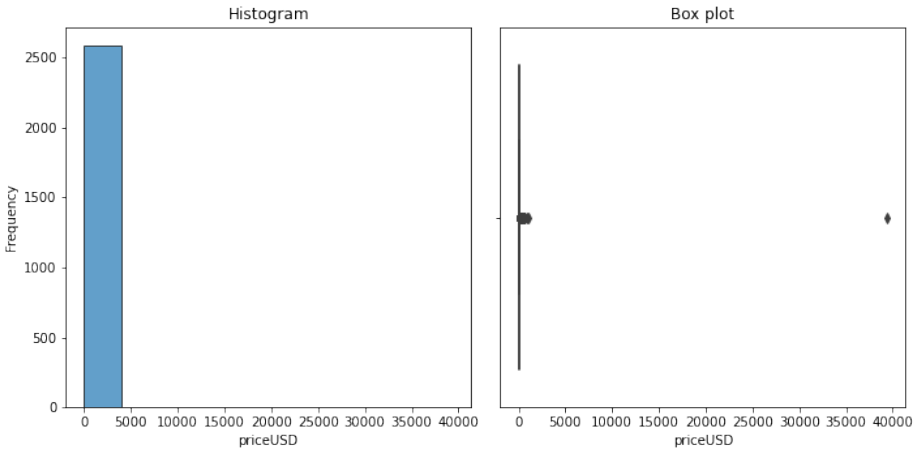

Figure 2: Histogram and Boxplot of priceUSD variable.

The histogram of the *teamSize* variable indicates that the majority of records are concentrated towards the lower values, with only a few records towards the higher values. The histogram is left-skewed, with a long tail on the left side. The boxplot of *teamSize* as shown in figure 3 suggests the presence of outliers in the upper quartiles.

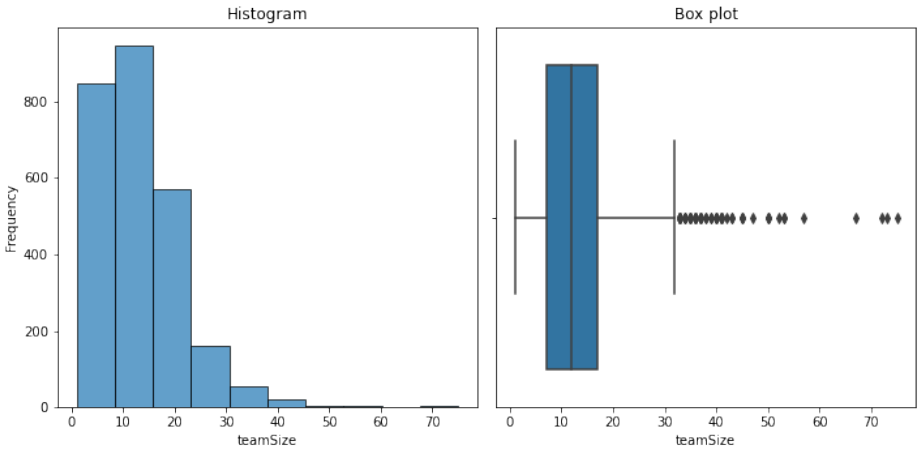

Figure 3: Histogram and Boxplot of teamSize variable.

The histogram of the *coinNum* variable shows a high concentration of records around zero as shown in figure 4, indicating that a large number of records have a value of zero for *coinNum*. There is only one tail in the histogram, and the boxplot is also appropriate. However, the presence of outliers is evident from the boxplot, including one extreme outlier.

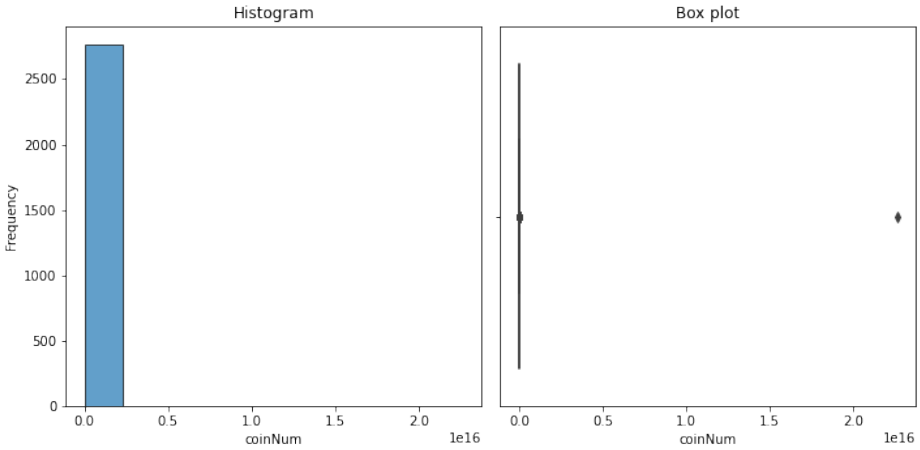

Figure 4: Histogram and Boxplot of coinNum variable.

The histogram Figure 5 of the *distributedPercentage* variable exhibits a long tail on the left side, similar to the *priceUSD* variable. This is also a result of the presence of outliers. The boxplot for *distributedPercentage* confirms the presence of outliers in this variable as well.

The percentage of outliers in the *coinNum* column is 14.28%, which is quite significant. The next highest percentage of outliers is found in the *priceUSD* column, with a value of 8.67%. The *teamSize* and *distributedPercentage* columns have lower percentages of outliers, with values of 2.39% and 0.36% respectively. It is important to address these outliers as they can have a notable impact on the data analysis and modeling process.

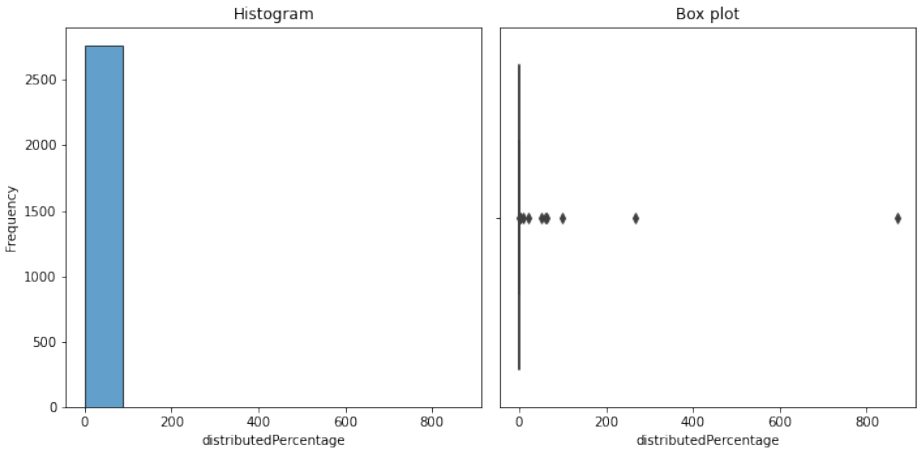

Figure 5: Box plot and Histogram of distributedPercentage variable.

The correlation matrix of the variables was calculated and visualized using a heatmap, as shown in Figure 6. The heatmap indicates that there is no strong correlation between most of the variables, except for the correlation between the *ID* and *rating* columns, as well as the correlation between the *ID* and *hasReddit* columns. Considering this high correlation with other variables, the *ID* column is dropped before building the model. This is done to avoid multicollinearity issues and to ensure that the model is not influenced by redundant information.

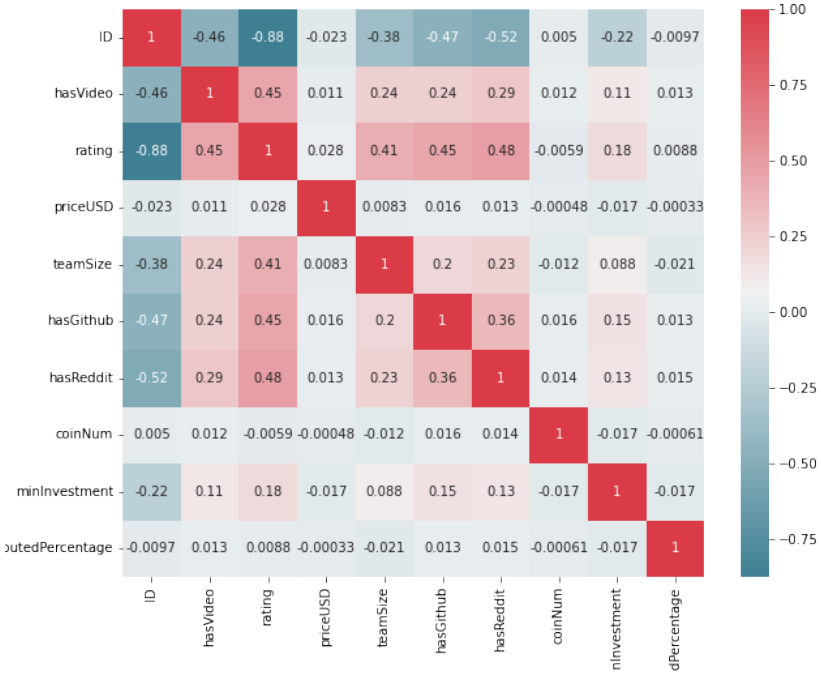

Figure 6: Correlation Matrix.

3. Data Preparation

3.1 Handling Missing Data

Since the *countryRegion* column is categorical, the null values can be imputed with the mode value, which represents the most frequent country region in the dataset. On the other hand, the *priceUSD* and *teamSize* columns are numerical, and to handle the null values, they can be replaced with either the mean or median value. However, since both columns contain outliers, it is more robust to use the median value for imputation.

Additionally, it is mentioned that the *priceUSD* column has a data error where some values are 0. These zero values can also be replaced with the median value to address the issue and ensure consistency in the dataset.

3.2 Handling Outliers

Outliers play a vital role in model development. Although there are a significant number of outliers in the dataset, dropping all of them may result in the loss of valuable information for the model. However, it is observed from the boxplots that there are a few extreme data points in the columns *priceUSD*, *teamSize*, and *distributedPercentage*. To address this, the extreme data points above the 95^{th} percentile and below the 5^{th} percentile were trimmed. After trimming these extreme data points, the dataset consists of 2521 observations. Approximately 5% of the data has been dropped.

3.3 Feature Engineering

In the dataset, six categorical columns need to be transformed into numerical features for model building. This is achieved through feature engineering techniques. Firstly, label encoding is applied to the *success* column, which has only two unique values. 'N' is replaced with zero and 'Y' is replaced with one.

A new numerical column called *duration* is created, which represents the number of days required for fundraising based on the start date and end date. This provides a quantitative measure of the fundraising duration.

The *brandSlogan_score* column is created based on the *brandSlogan* column. Sentiment analysis is applied to the brand slogans, and the resulting sentiment scores are stored in the new numerical column *brandSlogan_score*. This allows for capturing the sentiment or emotional tone conveyed by the brand slogans.

In the *platform* column, it is observed that 87.10% of the data corresponds to the Ethereum platform, which has variations due to data errors, extra spaces, shortcuts, and spelling mistakes. To address this, a new column called *platform_ethereum* is created. It assigns a value of one for rows where the platform is Ethereum and zero for all other platforms.

For the *countryRegion* column, data errors are resolved by converting all characters to lowercase and replacing special characters with their corresponding alphabetic characters. Additionally, one-hot encoding is applied to the top 10 countries, which

account for approximately 60% of the data. This allows representing the countries as binary columns, where a value of 1 indicates the presence of a specific country and 0 indicates its absence. It is worth noting that the *countryRegion* column initially had 120 unique values, with the remaining countries accounting for 40% of the data.

3.4 Feature Selection

After performing feature engineering, the input feature for the model is created using all the numerical variables except the target variable, and the feature engineering columns are included. The output/class label column is also created for the model. The total number of input features is 22.

To select the most relevant features for the model, we used the ANOVA (Analysis of Variance) test. ANOVA is a statistical test used to determine if the means of two or more groups are significantly different from each other. By applying the ANOVA test, we have identified the top 5 features for building the model. These features include *hasvideo*, *rating*, *teamSize*, *hasGithub*, and *hasReddit*. These selected features will be used as input variables in the model to make predictions or classifications.

3.5 Splitting the data

The dataset contains 2767 observations. Since it is a binary classification model, the output variable should have only 0 or 1 values. The output variable has 1739 observations as zeros and 1028 observations as ones, indicating an imbalanced dataset.

The data is split into training data and test data in an 80:20 ratio. Since it is an imbalanced dataset, it is important to ensure that the number of imbalanced observations is maintained in the same proportion in both the training and test datasets to avoid biased model performance. Please note that the term **imbalance** refers to the unequal distribution of the target variable classes.

3.6 Feature Scaling

Scaling is a preprocessing step for machine learning algorithms. It transforms numerical features to a common scale, ensuring that they have similar ranges and distributions. One commonly used scaling technique is Standardization, which performs a z-score transformation. It subtracts the mean from each data point and divides it by the standard deviation, resulting in a distribution with a mean of zero and a standard deviation of one.

Scaling is particularly important for algorithms that rely on distance calculations, such as k-nearest neighbors. It helps to prevent features with larger scales from dominating the learning process and ensures that all features contribute equally. Additionally, scaling can improve the convergence and efficiency of gradient-based optimization algorithms.

While scaling is generally beneficial for most machine learning algorithms, it may not have a significant impact on tree-based algorithms like decision trees or random forests. These algorithms are not sensitive to the scale of the input features because they make decisions based on relative feature thresholds.

4. Modelling

In this section, a machine learning model is built to achieve the objective. As the output feature consists of classes or labels, classification models are employed for this task. A classification model is a supervised machine learning model that aims to predict categorical or discrete class labels for data instances based on their input features. Given that the output feature in this dataset comprises two labels, the task is considered a binary classification problem. To accomplish our objective, we have explored various classification techniques to predict the success of fundraising campaigns through ICOs. Each model possesses its strengths and makes certain assumptions, and we evaluated their performance using appropriate metrics to identify the most suitable models for this specific task.

4.1 K nearest neighbor classifier

This algorithm classifies new data points based on their closest neighbors. It does not make any assumptions about the underlying data distribution and can handle both binary and multi-class classification problems. However, it may not be suitable for large datasets as it can be computationally expensive.

4.2 Naive bayes classifier

Naïve Bayes is a probabilistic classifier based on Bayes' theorem. It assumes independence between features and calculates the probability of each class given the feature values. The class with the highest probability is selected as the predicted label. Naïve Bayes performs well with high-dimensional datasets and is computationally efficient. However, it may encounter difficulties when dealing with correlated features.

4.3 Support vector machines

The SVM algorithm finds an optimal hyperplane to separate data into different classes. It can handle both linearly separable and non-linearly separable data by utilizing various kernel functions. SVM aims to maximize the margin between classes, which enhances its ability to generalize well to unseen data. However, SVM's performance is sensitive to the choice of hyperparameters, and selecting appropriate hyperparameters is crucial for achieving optimal results.

4.4 Decision Tree classifier

Decision trees are effective models that create a tree-like structure to make decisions based on feature values. They are easy to interpret and can handle both categorical and numerical features. However, decision trees tend to overfit the training data, meaning they may capture the noise and specific patterns of the training set too well, resulting in poor generalization to unseen data.

4.5 Random Forest classifier

Random Forest is an ensemble technique that combines multiple decision trees to make predictions. By aggregating the predictions of individual trees, it can reduce the overfitting issue commonly associated with decision trees. Random Forest is a robust model that can handle high-dimensional data and provide feature importance measures, which can be useful for understanding the impact of different features on the predictions. However, it can be computationally expensive for large datasets due to the training and inference processes involving multiple trees.

4.6 Adaboost classifier

Adaboost is an ensemble method that combines weak classifiers to create a strong classifier. It assigns higher weights to misclassified samples, allowing subsequent weak learners to focus on those samples and improve their overall performance. Adaboost is known for its ability to handle noisy data and can effectively deal with misclassifications. However, it is sensitive to outliers, which can impact its performance.

4.7 XGBoost classifier

XGBoost is an optimized implementation of the gradient boosting algorithm that sequentially adds weak learners to improve the overall performance. It handles missing data, supports regularization techniques, and provides efficient tree-based parallelization. XGBoost is known for its high performance and has been successful in many machine-learning competitions.

The models were built using the aforementioned classification techniques to identify the best model for accurately and robustly predicting the success of ICO fundraising campaigns. To evaluate the performance of each model, we will utilize various evaluation metrics such as accuracy, F1-score, recall, precision, ROC, and AUC. These metrics will allow us to assess the effectiveness of each model and determine which one performs the best for this particular task.

5. Evaluation

The metrics used for evaluating the model are Accuracy, Precision, Recall, F1-score, ROC, and AUC curve. Before evaluating the model using these metrics, let's understand what each metric means:

- **Accuracy:** It indicates the overall correctness of the model's predictions.
- **Precision:** It is the proportion of true positive predictions out of all positive predictions made by the model.
- **Recall:** It is the proportion of true positive predictions out of all actual positive instances in the dataset.
- **F1-score:** It combines both precision and recall into a single metric that balances both measures, providing a harmonic mean between them.

- **ROC:** It is a curve that shows the performance of all classification models at various classification thresholds.
- **Area under the Curve (AUC):** It measures the model's ability to distinguish between positive and negative instances by calculating the area under the receiver operating characteristic (ROC) curve.

Based on the table mentioned, we can assess the performance of each model using various performance measures. The AdaBoost model demonstrates the highest accuracy, achieving a value of 0.684. The SVM model exhibits the highest precision, with a value of 0.667. In terms of recall, the SVM model performs the best, attaining a value of 0.243. The Adaboost model displays the highest F1-score, reaching a value of 0.493. Lastly, the SVM model achieves the highest AUC, with a value of 0.669.

Model	Accuracy	Precision	Recall	F1-score	AUC
K-Nearest Neighbors	0.652	0.569	0.315	0.405	0.653
Naive Bayes	0.633	0.512	0.528	0.520	0.682
SVM	0.669	0.667	0.244	0.357	0.670
RandomForest	0.629	0.509	0.411	0.455	0.605
DecisionTree	0.623	0.500	0.396	0.442	0.670
XGB	0.623	0.500	0.376	0.429	0.618
AdaBoost	0.686	0.630	0.406	0.494	0.680

Table 1: Model Performance Metrics

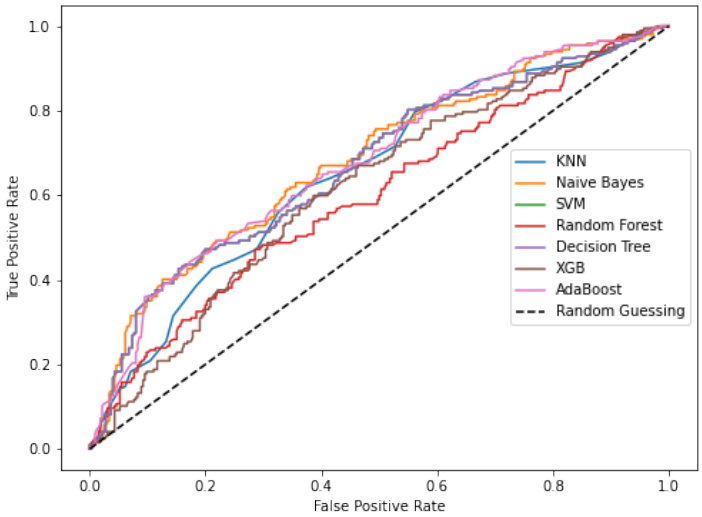

Figure 7: ROC Curve of All Models

The ROC curve of all classification models is plotted in Figure 7. From the figure, it can be observed that Naive Bayes has the highest AUC of 0.682, indicating a better ability to distinguish between positive and negative instances compared to other models. The second-highest AUC of 0.680 is achieved by the AdaBoost model.

Based on all the evaluation metrics, we can see that the AdaBoost model achieved the highest accuracy among the models. Additionally, its AUC is the second highest, which is close to the highest AUC value. The AdaBoost model also exhibits relatively better recall and F1-score compared to other models. Therefore, we can conclude that the AdaBoost model performed better than the others and can be considered the best model for predicting whether a team/company will reach its fundraising goal successfully through the ICO.

6. Conclusion

In conclusion, our analysis focused on predicting the success of fundraising goals in ICOs using classification models. Among these models, the AdaBoost model demonstrated the highest performance, indicating its potential for accurately predicting ICO success. However, it is important to acknowledge that predicting ICO outcomes is a complex task influenced by various factors such as market dynamics and project scope. Therefore, while the models provide valuable insights, decision-makers should consider additional factors when assessing the potential success of ICO fundraising campaigns. Our analysis contributes to the field of ICO analysis and empowers stakeholders to make more informed decisions, optimizing resource allocation for increased chances of achieving fundraising goals in the dynamic and competitive ICO ecosystem.

A. Appendices

A.1 Data Understanding

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from textblob import TextBlob
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from category_encoders import BinaryEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import warnings

df = pd.read_csv("LUBS5990M_courseworkData_202223.csv")
print(df.head())

# Dataset Overview
print(df.shape)
numerical_vars = df.select_dtypes(include=['float64', '
   int64']).columns.tolist()
print("Numerical␣Variables:␣", numerical_vars)
```

```
categorical_vars = df.select_dtypes(include=['object']).
   columns.tolist()
print("Categorical Variables: ", categorical_vars)

for col in numerical_vars:
    unique_count = df[col].nunique()
    print(f"Unique value count for column '{col}': {
       unique_count}")

original_numerical_values = ['rating', 'priceUSD', '
   teamSize', 'coinNum', 'distributedPercentage']
for col in categorical_vars:
    unique_count = df[col].nunique()
    print(f"Unique value count for column '{col}': {
       unique_count}")

# Data Quality
missing_perc = df.isna().mean() * 100
columns_with_missing = missing_perc[missing_perc > 0]
table = pd.DataFrame(columns=["Column Name", "Missing
   Percentage"])
table["Column Name"] = columns_with_missing.index
table["Missing Percentage"] = columns_with_missing.values.
   round(2)
print(table)

num_missing = df.isnull().any(axis=1).sum()
pct_missing = (num_missing / len(df)) * 100
print("Percentage of records with missing values:",
   pct_missing.round(2))

percentage = (df[df['priceUSD'] == 0]['priceUSD'].count() /
    len(df)) * 100
print(f"Percentage of priceUSD values that are 0 : {
   percentage:.2f}%")

duplicate_percentage = (len(df[df.duplicated()]) / len(df))
    * 100
print(f"Percentage of duplicates: {duplicate_percentage}%")

for col in categorical_vars:
    empty_count = df[df[col] == ' '].shape[0]
    total_count = df.shape[0]
    percentage = np.round(np.divide(empty_count,
       total_count) * 100.0, 2)
```

```
    print(f"Percentage␣of␣empty␣strings␣in␣column␣{col}:␣{
        percentage}%")

unique_chars = df['countryRegion'].unique()
print((sorted([char for char in unique_chars if pd.notnull(
    char)])))

unique_chars = df['platform'].unique()
print((sorted([char for char in unique_chars if pd.notnull(
    char)])))

# Statistical Summaries
summary = df[original_numerical_values].describe()
print(summary)

# Data Visualisation
def plot_histogram_boxplot(df, col):
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 5)
        )

    ax1.hist(df[col], bins=10, edgecolor='black', alpha
        =0.7)
    ax1.set_xlabel(col)
    ax1.set_ylabel('Frequency')
    ax1.set_title('Histogram')

    sns.boxplot(df[col])
    ax2.set_title('Box␣plot')
    ax2.set_xlabel(col)

    # plt.suptitle('Box Plots and Histogram of '+ col +'
        variable', fontsize=14)
    plt.tight_layout()

    plt.savefig(col + '.png', format='png')
    plt.show()

for i, col in enumerate(original_numerical_values):
    plot_histogram_boxplot(df, col)

warnings.filterwarnings('ignore')

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
outliers = ((df < lower_bound) | (df > upper_bound))
outliers_percentage = (outliers.sum() / len(df)) * 100
columns_with_outliers = outliers_percentage[
   outliers_percentage > 0]

print("\nColumns␣with␣Outliers:")
table = pd.DataFrame(columns=["Column␣Name", "Outliers␣
   Percentage"])
table["Column␣Name"] = columns_with_outliers.index
table["Outliers␣Percentage"] = columns_with_outliers.values
   .round(2)
print(table)

correlation_matrix = df.corr()
fig, ax = plt.subplots(figsize=(10, 8))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(correlation_matrix, annot=True, cmap=cmap, ax=
   ax)
ax.set_title('Correlation␣Matrix')
plt.savefig('correlation_matrix.png', format='png')
plt.show()
```

Code extraction 1: Python code for data understanding section

A.2 Data Preparation

```
# Handling Missing Data
df['priceUSD'].fillna(df['priceUSD'].median(), inplace=True
   )
df['teamSize'].fillna(df['teamSize'].median(), inplace=True
   )
df['countryRegion'].fillna(df['countryRegion'].mode()[0],
   inplace=True)

df['priceUSD'] = np.where(df['priceUSD'] == 0, df['priceUSD
   '].median(), df['priceUSD'])

# Handling Outliers
outlier_columns = ['priceUSD', 'teamSize', '
   distributedPercentage', 'coinNum']
for column in outlier_columns:
    q1 = df[column].quantile(0.05)
    q3 = df[column].quantile(0.95)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
```

```
    df = df[(df[column] >= lower_bound) & (df[column] <=
       upper_bound)]
print(df.shape)

# Feature Engineering
label_encoder = LabelEncoder()
df['success_encoded'] = label_encoder.fit_transform(df['
   success'])
value_counts = df['countryRegion'].str.lower().replace(['
   curacao', 'cura ao'], 'curacao').value_counts()
percentage = value_counts / len(df) * 100
total_percentage = percentage.head(10).sum()
print("Total percentage for top 10 unique values: %.2f%%" %
    total_percentage)
df['countryRegion'] = df['countryRegion'].str.lower().
   replace(['curacao', 'cura ao'], 'curacao')
top_10_countries = df['countryRegion'].value_counts().head
   (10).index.tolist()

filtered_df = df[df['countryRegion'].isin(top_10_countries)
   ]
df = pd.concat([df, pd.get_dummies(filtered_df['
   countryRegion'])], axis=1)

countries_to_fill = ['cayman islands', 'estonia','germany',
    'malta', 'netherlands', 'russia', 'singapore', '
   switzerland','uk', 'usa',]
df[countries_to_fill] = df[countries_to_fill].fillna(0)

df['duration'] = np.abs((pd.to_datetime(df['endDate']) - pd
   .to_datetime(df['startDate'])).dt.days)

values_to_check = ['ETH', 'Ethererum', 'Ethereum', '
   Ethereum, Waves', 'Etherum']
value_counts = df['platform'].str.strip().value_counts()
filtered_counts = value_counts[value_counts.index.isin(
   values_to_check)]
percentages = (filtered_counts / value_counts.sum()) * 100
print("Total percentage for Ethererum platform : %.2f%%" %
   percentages.sum())

values_to_check = ['ETH', 'Ethererum', 'Ethereum', '
   Ethereum, Waves', 'Etherum']
df['platform_Ethereum'] = np.where(df['platform'].isin(
   values_to_check), 1, 0)

df['brandSlogan_score'] = df['brandSlogan'].apply(lambda x:
```

```
    TextBlob(str(x)).sentiment.polarity)

# Feature Selection
X = df.drop(['success','success_encoded','brandSlogan','
   platform','startDate','endDate','ID','countryRegion'],
   axis=1)
y = df['success_encoded']
print(X.shape)
from sklearn.feature_selection import SelectKBest,
   f_classif

selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X, y)
feature_indices = selector.get_support(indices=True)
selected_feature_names = X.columns[feature_indices] if
   isinstance(X, pd.DataFrame) else None

print(selected_feature_names.values)

#  Test Train Split
platform_counts = df['success'].value_counts()
print("Frequency␣of␣success:")
print(platform_counts)
X_train, X_test, y_train, y_test = train_test_split(
   X_selected, y, test_size=0.2, stratify=y, random_state
   =42)

# Scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Code extraction 2: Python code for data preparation section

A.3 Modelling

```
# KNN classifier Model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,
   classification_report, precision_score, recall_score,
   f1_score, roc_curve, auc
knn = KNeighborsClassifier(n_neighbors=47)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
probas = knn.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])
roc_auc = auc(fpr, tpr)
metrics_df = pd.DataFrame({
    'Model': ['K-Nearest Neighbors'],
    'Accuracy': [accuracy],
    'Precision': [precision],
    'Recall': [recall],
    'F1-score': [f1],
    'AUC' : [roc_auc],
})

# Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
y_pred = naive_bayes.predict(X_test)
probas = naive_bayes.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])

metrics_df.loc[1, 'Model'] = 'Naive Bayes'
metrics_df.loc[1, 'Accuracy'] = accuracy_score(y_test,
   y_pred)
metrics_df.loc[1, 'Precision'] = precision_score(y_test,
   y_pred)
metrics_df.loc[1, 'Recall'] = recall_score(y_test, y_pred)
metrics_df.loc[1, 'F1-score'] = f1_score(y_test, y_pred)
metrics_df.loc[1, 'AUC'] = auc(fpr, tpr)

# SVM Model
from sklearn.svm import SVC

svm = SVC(kernel='linear', probability=True)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
probas = svm.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])

metrics_df.loc[2, 'Model'] = 'SVM'
metrics_df.loc[2, 'Accuracy'] = accuracy_score(y_test,
   y_pred)
metrics_df.loc[2, 'Precision'] = precision_score(y_test,
   y_pred)
metrics_df.loc[2, 'Recall'] = recall_score(y_test, y_pred)
```

```
metrics_df.loc[2, 'F1-score'] = f1_score(y_test, y_pred)
metrics_df.loc[2, 'AUC'] = auc(fpr, tpr)

# Random Forest Model
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
y_pred = random_forest.predict(X_test)
probas = random_forest.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])

metrics_df.loc[3, 'Model'] = 'RandomForest'
metrics_df.loc[3, 'Accuracy'] = accuracy_score(y_test,
   y_pred)
metrics_df.loc[3, 'Precision'] = precision_score(y_test,
   y_pred)
metrics_df.loc[3, 'Recall'] = recall_score(y_test, y_pred)
metrics_df.loc[3, 'F1-score'] = f1_score(y_test, y_pred)
metrics_df.loc[3, 'AUC'] = auc(fpr, tpr)

# Decision Tree Model
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
probas = svm.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])

metrics_df.loc[4, 'Model'] = 'DecisionTree'
metrics_df.loc[4, 'Accuracy'] = accuracy_score(y_test,
   y_pred)
metrics_df.loc[4, 'Precision'] = precision_score(y_test,
   y_pred)
metrics_df.loc[4, 'Recall'] = recall_score(y_test, y_pred)
metrics_df.loc[4, 'F1-score'] = f1_score(y_test, y_pred)
metrics_df.loc[4, 'AUC'] = auc(fpr, tpr)

# XGBoost Model
import xgboost as xgb

xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)
probas = xgb_model.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])
```

```
metrics_df.loc[5, 'Model'] = 'XGB'
metrics_df.loc[5, 'Accuracy'] = accuracy_score(y_test,
   y_pred)
metrics_df.loc[5, 'Precision'] = precision_score(y_test,
   y_pred)
metrics_df.loc[5, 'Recall'] = recall_score(y_test, y_pred)
metrics_df.loc[5, 'F1-score'] = f1_score(y_test, y_pred)
metrics_df.loc[5, 'AUC'] = auc(fpr, tpr)

# AdaBoost Model
from sklearn.ensemble import AdaBoostClassifier

ada_model = AdaBoostClassifier()
ada_model.fit(X_train, y_train)
y_pred = ada_model.predict(X_test)
probas = xgb_model.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])

metrics_df.loc[6, 'Model'] = 'AdaBoost'
metrics_df.loc[6, 'Accuracy'] = accuracy_score(y_test,
   y_pred)
metrics_df.loc[6, 'Precision'] = precision_score(y_test,
   y_pred)
metrics_df.loc[6, 'Recall'] = recall_score(y_test, y_pred)
metrics_df.loc[6, 'F1-score'] = f1_score(y_test, y_pred)
metrics_df.loc[6, 'AUC'] = auc(fpr, tpr)
```

Code extraction 3: Python code for building the model

A.4 Evaluation

```
# prints the metrics for all models
print(metrics_df)

# plots ROC curve for all models
from sklearn.metrics import roc_curve, auc
models = ['KNN', 'Naive␣Bayes', 'SVM', 'Random␣Forest', '
   Decision␣Tree', 'XGB', 'AdaBoost']
probas = [probas0, probas1, probas2, probas3, probas4,
   probas5, probas6]

plt.figure(figsize=(8, 6))
for i in range(len(models)):
    fpr, tpr, thresholds = roc_curve(y_test, probas[i][:,
       1])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=models[i])
```

```
plt.plot([0, 1], [0, 1], 'k--', label='Random␣Guessing')
plt.xlabel('False␣Positive␣Rate')
plt.ylabel('True␣Positive␣Rate')
plt.title('Receiver␣Operating␣Characteristic␣(ROC)␣Curve')
plt.legend(loc='center␣right')
plt.savefig('ROC.png', format='png')
plt.show()
```

Code extraction 4: Python code for model evaluation