Classification of Consumer Data into Segments | Clusters | Classes

1. Project Objectives | Problem Statements

1.1. **PO1 | PS1: Classification of Funding Round Data into Segments | Clusters | Classes using Supervised Learning Classification Algorithms**

- PS1.1: Develop a classification model to segment funding round data into distinct classes based on features such as `funding_round_type`, `raised_amount_usd`, `participants`, etc.
- PS1.2: Utilize supervised learning algorithms including Decision Tree, KNN (K-Nearest Neighbors), Logistic Regression, and SVM (Support Vector Machine) to accomplish classification tasks.

1.2. **PO2 | PS2: Determination of an Appropriate Classification Model**

- PS2.1: Evaluate and compare the performance of Decision Tree, KNN, Logistic Regression, and SVM models in classifying funding round data.
- PS2.2: Determine the most suitable classification model based on metrics such as accuracy, precision, recall, and F1-score.

1.3. **PO3 | PS3: Identification of Important | Contributing | Significant Variables or Features and their Thresholds for Classification**

- PS3.1: Conduct feature importance analysis to identify significant variables contributing to the classification of funding round data.
- PS3.2: Determine optimal thresholds for key features such as `raised_amount_usd`, `pre_money_valuation_usd`, and `post_money_valuation_usd` to improve classification accuracy.

Dropping Unwanted Variables

In the preprocessing stage of our analysis on the funding round dataset, we identified certain variables that were deemed unnecessary for our clustering analysis. These variables did not contribute to the segmentation process or were redundant for our objectives. Initially, the dataset comprised a total of 23 columns.

The following 12 columns were dropped from the dataset:

1. funding_round_id
2. raised_amount
3. raised_currency_code

4. pre_money_valuation
5. pre_money_currency_code
6. post_money_valuation
7. post_money_currency_code
8. source_url
9. source_description
10. created_at
11. updated_at
12. created_by

These columns were considered irrelevant or redundant for our clustering analysis. Therefore, to streamline our dataset and focus only on the pertinent variables, they were removed from further consideration.

The remaining variables, which were considered useful and relevant for our analysis, were retained and utilized in subsequent steps of the clustering process. These variables formed the basis for segmenting the funding round data and extracting meaningful insights to achieve our objectives.

2. Description of Data

2.1. Data Source, Size, Shape

2.1.1. Data Source (Website Link)

- The dataset was sourced from https://www.kaggle.com/datasets/justinas/startup-investments?select=funding_rounds.csv.

2.1.2. Data Size

- The size of the dataset is 13.13 MB

2.1.3. Data Shape

- The dataset has the following dimensions:
    - Number of Variables: 23 (11 Used, 12 Dropped)
    - Number of Records: 52928

2.2. Description of Variables

2.2.1. Index Variable(s)

- ID is the Index Variable present in the dataset.

2.2.2. Variables or Features having Categories | Categorical Variables or Features (CV)

2.2.2.1. Variables or Features having Nominal Categories | Categorical Variables or Features - Nominal Type

- `funding_round_type`
- `funding_round_code`
- `is_first_round`
- `is_last_round`
- `created_by`

2.2.2.2. Variables or Features having Ordinal Categories | Categorical Variables or Features - Ordinal Type

- None in the dataset.

2.2.3. Non-Categorical Variables or Features

- `id`
- `object_id`
- `funded_at`
- `raised_amount_usd`
- `pre_money_valuation_usd`
- `post_money_valuation_usd`

Description of Variables:

- **ID:** A unique identifier for each funding record.
- **Object_ID:** A distinct identifier for the startup or project associated with the funding round.
- **Funded_At:** The timestamp indicating when the funding round took place.
- **Funding_Round_Type:** Describes the type of funding round (e.g., seed, series A, series B).
- **Funding_Round_Code:** An alphanumeric code associated with the funding round type.
- **Raised_Amount_USD:** The amount of capital raised during the funding round in US dollars.
- **Pre_Money_Valuation_USD:** The startup's estimated valuation before the funding round in US dollars.
- **Post_Money_Valuation_USD:** The startup's estimated valuation after the funding round in US dollars.
- **Participants:** The number of participants or investors in the funding round.
- **Is_First_Round:** A binary indicator specifying whether the funding round is the first for the startup.
- **Is_Last_Round:** A binary indicator specifying whether the funding round is the last for the startup.
- **Created_By:** Information about the entity or individual responsible for creating the funding round data.

This dataset provides comprehensive information about startup investments, focusing on funding rounds. It facilitates detailed exploratory data analysis, enabling insights into trends, investor participation, and the lifecycle of startup funding. The data will be valuable for subsequent tasks such as clustering or predictive modeling to uncover patterns and factors influencing investment success.

3.1 Data Pre-Processing

3.1.1 Missing Data Statistics and Treatment

3.1.1.1 Missing Data Statistics: Records and Variables: Missing data statistics were assessed across all variables, revealing **248 missing entries** in the **'funded_at'** variable, while other variables including 'funding_round_type', 'funding_round_code', 'raised_amount_usd', 'pre_money_valuation_usd', 'post_money_valuation_usd', 'participants', 'is_first_round', and 'is_last_round' showed no missing values.

3.1.1.2 Missing Data Treatment: Records and Variables: No records were removed due to having more than 50% missing data. For categorical variables, a simple imputer technique was applied using the most frequent value **(mode)** to fill in missing data. Similarly, for non-categorical variables, missing data was imputed using the mean or median value (simple imputer, using most frequent) to ensure completeness in the dataset.

3.1.1.3 Missing Data Treatment: Categorical Variables: Missing data in categorical variables, including 'funding_round_type', 'funding_round_code', 'participants', 'is_first_round', and 'is_last_round', were treated using the simple imputer technique, specifically filling in missing values with the most frequent category **(mode)**.

3.1.1.4 Missing Data Treatment: Non-Categorical Variables: Non-categorical variables such as 'funded_at', 'raised_amount_usd', 'pre_money_valuation_usd', and 'post_money_valuation_usd' had missing data imputed using the simple imputer method, replacing missing values with the mean or median, ensuring the completeness of the dataset.

3.1.2 Numeric Encoding of Categorical Variables or Features: Categorical variables 'funding_round_type' and 'funding_round_code' were encoded numerically using the pandas library, facilitating further analysis by transforming categorical data into a format suitable for mathematical models.

3.1.3 Outlier Statistics and Treatment:

3.1.3.1 Outlier Statistics: Non-Categorical Variables or Features: Outlier analysis revealed the presence of outliers in 'raised_amount_usd', 'pre_money_valuation_usd', and 'post_money_valuation_usd' variables, as evidenced by box plots.

3.1.3.2 Outlier Treatment: Non-Categorical Variables or Features:

3.1.3.2.1 Normalization using Min-Max Scaler: To address outliers, a normalization technique using Min-Max Scaler was employed for non-categorical variables, ensuring that the data distribution is within a defined range, thereby minimizing the impact of outliers on subsequent analyses.

3.2. Data Analysis

3.2.1.1. PO1 | PS1: Supervised Machine Learning Classification Algorithm: Decision Tree (Base Model) | Metrics Used - Gini Coefficient, Entropy

In this section, we employed a Decision Tree algorithm as the base model for our classification task. The metrics utilized for evaluation were Gini Coefficient and Entropy. The analysis revealed:

- **Classification Report**:
    - The Decision Tree model achieved remarkable performance with an accuracy of 100% across all classes (0.0, 1.0, and 2.0).
    - Precision, recall, and F1-score were all perfect for each class, indicating a highly accurate classification.
- **Confusion Matrix**:
    - The confusion matrix further corroborates the model's exceptional performance, with minimal misclassifications.

3.2.1.2. PO1 | PS1: Supervised Machine Learning Classification Algorithms: {Logistic Regression | Support Vector Machine | K Nearest Neighbour} (Comparison Models) | Metrics Used

For comparison, Logistic Regression, Support Vector Machine (SVM), and K Nearest Neighbor (KNN) algorithms were implemented. The following results were obtained:

- **Logistic Regression**:
    - Achieved an accuracy of 93% with precision, recall, and F1-score varying across classes.
    - Confusion matrix indicates some misclassifications compared to the Decision Tree model.
- **K Nearest Neighbor (KNN)**:
    - Demonstrated an accuracy of 87.3%, with precision, recall, and F1-score varying across classes.
    - Confusion matrix reveals moderate misclassifications, particularly in the second class.
- **Support Vector Machine (SVM)**:
    - Achieved an accuracy of 99%, outperforming both Logistic Regression and KNN.
    - Precision, recall, and F1-score were consistently high across all classes.

3.2.2.1.1. PO2 | PS2: Classification Model Performance Evaluation: Confusion Matrix {Accuracy, Recall, Precision, F1-Score} (Base Model: Decision Tree)

The performance of the Decision Tree model was evaluated using various metrics:

- **Accuracy**: Achieved a perfect accuracy of 100%.

- **Recall, Precision, F1-Score**: All metrics attained a score of 1.0, indicating flawless classification across classes.

3.2.2.1.2. PO2 | PS2: Classification Model Performance Evaluation: Time Statistics | (CPU | GPU) Memory Statistics (Base Model: Decision Tree)

The computational resources utilized by the Decision Tree model were as follows:

- **Time taken**: Approximately 0.12 seconds.
- **Memory used**: Approximately 689.25 KB.

3.2.2.2.1. PO2 | PS2: Classification Model Performance Evaluation: Confusion Matrix {Accuracy, Recall, Precision, F1-Score} (Comparison Models: Logistic Regression | Support Vector Machine | K Nearest Neighbour)

The performance of the comparison models was assessed using similar metrics:

- **Logistic Regression**:
    - Accuracy: 93%
    - Precision, recall, and F1-score varied across classes.
- **K Nearest Neighbor (KNN)**:
    - Accuracy: 87.3%
    - Precision, recall, and F1-score varied across classes.
- **Support Vector Machine (SVM)**:
    - Accuracy: 99%
    - Precision, recall, and F1-score were consistently high across all classes.

3.2.2.2.2. PO2 | PS2: Classification Model Performance Evaluation: Time Statistics | (CPU | GPU) Memory Statistics (Comparison Models: Logistic Regression | Support Vector Machine | K Nearest Neighbour)

Resource utilization for the comparison models was as follows:

- **Logistic Regression**:
    - Time taken: Approximately 1.87 seconds.
    - Memory used: Approximately 723.69 KB.
- **K Nearest Neighbor (KNN)**:
    - Time taken: Approximately 1.74 seconds.
    - Memory used: Approximately 1150.28 KB.
- **Support Vector Machine (SVM)**:
    - Time taken: Approximately 191.38 seconds.
    - Memory used: Approximately 690.22 KB.

3.2.3.1. PO3 | PS3: Variable or Feature Analysis: Base Model (Decision Tree)

Analysis of important features for the Decision Tree model revealed:

- **Relevant Variables**:
  - `funding_round_code`, `funding_round_type`, and `participants` were identified as significant features.
- **Non-Relevant Variables**:
  - `raised_amount_usd`, `id`, `is_first_round`, `is_last_round`, `pre_money_valuation_usd`, and `post_money_valuation_usd` were deemed non-significant.

3.2.3.2. PO3 | PS3: Variable or Feature Analysis: Comparison Models (Logistic Regression | Support Vector Machine | K Nearest Neighbour)

The analysis of important features for the comparison models revealed:

- **Logistic Regression**:
  - Relevant features: `funding_round_code`, `funding_round_type`, and `participants`.
  - Non-Relevant features: `is_first_round`, `is_last_round`, `raised_amount_usd`, `id`, `pre_money_valuation_usd`, and `post_money_valuation_usd`.
- **K Nearest Neighbor (KNN)**:
  - Relevant features: `funding_round_code`, `funding_round_type`, and `id`.
  - Non-Relevant features: `pre_money_valuation_usd`, `post_money_valuation_usd`, `raised_amount_usd`, `is_last_round`, `is_first_round`, and `participants`.
- **Support Vector Machine (SVM)**:
  - Relevant features: `funding_round_code`, `funding_round_type`, `is_last_round`, and `is_first_round`.
  - Non-Relevant features: `participants`, `raised_amount_usd`, `id`, `pre_money_valuation_usd`, and `post_money_valuation_usd`.

These findings provide insights into the importance of different features for each model, aiding in feature selection and model optimization.

4. Results | Observations

4.1. Classification Model Parameters: Base Model (Decision Tree) | Comparison Models (Logistic Regression | Support Vector Machine | K Nearest Neighbour)

- **Base Model (Decision Tree)**:
  - **Algorithm**: Decision Tree
  - **Metrics Used**: Gini Coefficient, Entropy
  - **Parameters**: Default parameters were used for the Decision Tree algorithm.

- **Comparison Models**:
  - **Logistic Regression**:
    - **Algorithm**: Logistic Regression
    - **Parameters**: Default parameters were used for logistic regression.
  - **Support Vector Machine (SVM)**:
    - **Algorithm**: SVM
    - **Parameters**: Default parameters were used for SVM.
  - **K Nearest Neighbour (KNN)**:
    - **Algorithm**: KNN
    - **Parameters**: Default parameters were used for KNN.

4.2. Classification Model Performance: Time & Memory Statistics [Base Model (Decision Tree) | Comparison Models (Logistic Regression | Support Vector Machine | K Nearest Neighbour)]

- **Base Model (Decision Tree)**:
  - **Time Taken**: 0.121 seconds
  - **Memory Used**: 689.25 KB
- **Comparison Models**:
  - **Logistic Regression**:
    - **Time Taken**: 1.87 seconds
    - **Memory Used**: 723.69 KB
  - **Support Vector Machine (SVM)**:
    - **Time Taken**: 191.38 seconds
    - **Memory Used**: 690.22 KB
  - **K Nearest Neighbour (KNN)**:
    - **Time Taken**: 1.74 seconds
    - **Memory Used**: 1150.28 KB

4.3. Variable or Feature Analysis: Base Model (Decision Tree) | Comparison Models (Logistic Regression | Support Vector Machine | K Nearest Neighbour)

4.3.1. List of Relevant or Important Variables or Features and their Thresholds

- **Base Model (Decision Tree)**:
  - **Relevant Variables**:
    - `funding_round_code` : Importance - 0.58363
    - `funding_round_type` : Importance - 0.41443
    - `participants` : Importance - 0.00172
- **Comparison Models**:

- **Logistic Regression**:
  - `funding_round_code` : Importance - 0.617541
  - `funding_round_type` : Importance - 0.538363
  - `participants` : Importance - 0.365511
- **Support Vector Machine (SVM)**:
  - `funding_round_code` : Importance - 1.751690
  - `funding_round_type` : Importance - 0.227470
  - `is_last_round` : Importance - 0.073870
  - `is_first_round` : Importance - 0.063772
- **K Nearest Neighbour (KNN)**:
  - `funding_round_code` : Importance - 0.394294
  - `funding_round_type` : Importance - 0.092103
  - `id` : Importance - 0.087285

4.3.2. List of Non-Relevant or Non-Important Variables or Features

- **Base Model (Decision Tree)**:
  - **Non-Relevant Variables**:
    - `raised_amount_usd_mmnorm` : Importance - 0.00012
    - `id` : Importance - 0.00009
    - `is_first_round_code` : Importance - 0.00000
    - `is_last_round_code` : Importance - 0.00000
    - `pre_money_valuation_usd_mmnorm` : Importance - 0.00000
    - `post_money_valuation_usd_mmnorm` : Importance - 0.00000
- **Comparison Models**:
  - **Logistic Regression**:
    - `is_first_round_code` : Importance - 0.041834
    - `is_last_round_code` : Importance - 0.040579
    - `raised_amount_usd_mmnorm` : Importance - 0.000541
    - `id` : Importance - 0.000096
    - `pre_money_valuation_usd_mmnorm` : Importance - 0.000035
    - `post_money_valuation_usd_mmnorm` : Importance - 0.000004
  - **Support Vector Machine (SVM)**:
    - `participants` : Importance - 0.013187
    - `raised_amount_usd_mmnorm` : Importance - 0.001287
    - `id` : Importance - 0.000094
    - `post_money_valuation_usd_mmnorm` : Importance - 0.000003
    - `pre_money_valuation_usd_mmnorm` : Importance - 0.000000

- **K Nearest Neighbour (KNN)**:
  - `pre_money_valuation_usd_mmnorm` : Importance - 0.0
  - `post_money_valuation_usd_mmnorm` : Importance - 0.0
  - `raised_amount_usd_mmnorm` : Importance - 0.0
  - `is_last_round_code` : Importance - 0.0
  - `is_first_round_code` : Importance - 0.0
  - `participants_code` : Importance - 0.0

5. Managerial Insights

5.1. Appropriate Model: Compare and Contrast {Decision Tree | Logistic Regression | Support Vector Machine | K Nearest Neighbour}

Classification Model Parameters:

All models were trained using default parameters, so no significant differences exist in this aspect.

Classification Model Performance: Time & Memory Statistics:

- **Base Model (Decision Tree)**:
  - Time Taken: 0.121 seconds
  - Memory Used: 689.25 KB
- **Comparison Models**:
  - **Logistic Regression**:
    - Time Taken: 1.87 seconds
    - Memory Used: 723.69 KB
  - **Support Vector Machine (SVM)**:
    - Time Taken: 191.38 seconds
    - Memory Used: 690.22 KB
  - **K Nearest Neighbour (KNN)**:
    - Time Taken: 1.74 seconds
    - Memory Used: 1150.28 KB

Variable or Feature Analysis: Relevant or Important Variables or Features:

- **Base Model (Decision Tree)**:
  - Relevant Variables: `funding_round_code` , `funding_round_type` , `participants`
- **Comparison Models**:
  - **Logistic Regression**:

        - Relevant Variables: `funding_round_code`, `funding_round_type`, `participants`
    - **Support Vector Machine (SVM)**:
        - Relevant Variables: `funding_round_code`, `funding_round_type`, `is_last_round`, `is_first_round`
    - **K Nearest Neighbour (KNN)**:
        - Relevant Variables: `funding_round_code`, `funding_round_type`, `id`

List of Non-Relevant or Non-Important Variables or Features:

- **Base Model (Decision Tree)**:
    - Non-Relevant Variables: `raised_amount_usd_mmnorm`, `id`, `is_first_round_code`, `is_last_round_code`, `pre_money_valuation_usd_mmnorm`, `post_money_valuation_usd_mmnorm`
- **Comparison Models**:
    - **Logistic Regression**:
        - Non-Relevant Variables: `is_first_round_code`, `is_last_round_code`, `raised_amount_usd_mmnorm`, `id`, `pre_money_valuation_usd_mmnorm`, `post_money_valuation_usd_mmnorm`
    - **Support Vector Machine (SVM)**:
        - Non-Relevant Variables: `participants`, `raised_amount_usd_mmnorm`, `id`, `post_money_valuation_usd_mmnorm`, `pre_money_valuation_usd_mmnorm`
    - **K Nearest Neighbour (KNN)**:
        - Non-Relevant Variables: `pre_money_valuation_usd_mmnorm`, `post_money_valuation_usd_mmnorm`, `raised_amount_usd_mmnorm`, `is_last_round_code`, `is_first_round_code`, `participants_code`

Analysis:

Considering the performance metrics and feature relevance, the **Decision Tree** model emerges as the best choice. It achieves comparable accuracy to SVM and Logistic Regression but with significantly lower computational resources and features relevant to the classification task. SVM, although having high accuracy, suffers from longer training times, while KNN shows moderate accuracy and higher memory usage.

Therefore, based on the provided data, the Decision Tree model is the best choice for this classification task.

5.2. Relevant or Important Variables or Features (Given the Appropriate Model)

For the chosen appropriate model, which is the Decision Tree, the relevant variables or features identified are as follows:

- **funding_round_code**: Importance - 0.58363
- **funding_round_type**: Importance - 0.41443
- **participants**: Importance - 0.00172

These features play a crucial role in the classification of funding round data into segments or classes. The Decision Tree algorithm considers these features as significant contributors to the classification process. Understanding and leveraging these variables can aid in making informed decisions related to funding rounds, such as predicting the success or outcome of investment opportunities.

```
# Required Libraries
import pandas as pd, numpy as np # For Data Manipulation
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder # For Encoding Categorical
from sklearn.preprocessing import OneHotEncoder # For Creating Dummy Variables of Categor
from sklearn.impute import SimpleImputer, KNNImputer # For Imputation of Missing Data
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler # For Rescal
from sklearn.model_selection import train_test_split # For Splitting Data into Training &
```

```
pip install memory-profiler
```

```
Requirement already satisfied: memory-profiler in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (frc
```

```
import warnings
warnings.filterwarnings("ignore") # Ignore the warnings
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Specify file path
file_path1 = '/content/drive/My Drive/data_funding.csv'  # Update with the actual path of
file_path2 = '/content/drive/My Drive/subset_id_cluster.csv'

# Read CSV file
import pandas as pd
df1 = pd.read_csv(file_path1)
df2 = pd.read_csv(file_path2)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
import pandas as pd

# Assuming df1 and df2 are your dataframes
df = pd.merge(df1, df2, on='id', how='inner')
df
```

	id	funding_round_id	object_id	funded_at	funding_round_type	funding_rou
0	1	1	c:4	01-12-06	series-b	
1	2	2	c:5	01-09-04	angel	
2	3	3	c:5	01-05-05	series-a	
3	4	4	c:5	01-04-06	series-b	
4	5	5	c:7299	01-05-06	series-b	
...	...	...	...	...	...	
52923	57948	57948	c:211890	12-12-13	series-a	
52924	57949	57949	c:267427	06-02-10	venture	
52925	57950	57950	c:261728	06-02-10	venture	una
52926	57951	57951	c:285864	12-12-13	series-a	
52927	57952	57952	c:286215	07-04-10	venture	

52928 rows × 24 columns

```
# Display the DataFrame
df.head()
```

	id	funding_round_id	object_id	funded_at	funding_round_type	funding_round_code
0	1	1	c:4	01-12-06	series-b	b
1	2	2	c:5	01-09-04	angel	angel
2	3	3	c:5	01-05-05	series-a	a
3	4	4	c:5	01-04-06	series-b	b
4	5	5	c:7299	01-05-06	series-b	b

5 rows × 24 columns

```
df = pd.DataFrame(df)

# Columns to drop
columns_to_drop = ['funding_round_id', 'created_by', 'object_id', 'raised_amount', 'raise

# Dropping specified columns
df.drop(columns=columns_to_drop, inplace=True)

# Displaying the modified DataFrame
df
```

	id	funded_at	funding_round_type	funding_round_code	raised_amount_usd	p
0	1	01-12-06	series-b	b	8500000	
1	2	01-09-04	angel	angel	500000	
2	3	01-05-05	series-a	a	12700000	
3	4	01-04-06	series-b	b	27500000	
4	5	01-05-06	series-b	b	10500000	
...	...	...	...	...	...	
52923	57948	12-12-13	series-a	a	3000000	
52924	57949	06-02-10	venture	partial	570000	
52925	57950	06-02-10	venture	unattributed	2184100	
52926	57951	12-12-13	series-a	a	790783	
52927	57952	07-04-10	venture	partial	271250	

52928 rows × 11 columns

```
df.dtypes
```

```
id                            int64
funded_at                    object
funding_round_type           object
funding_round_code           object
raised_amount_usd             int64
pre_money_valuation_usd       int64
post_money_valuation_usd      int64
participants                  int64
is_first_round                int64
is_last_round                 int64
cluster_number                int64
dtype: object
```

```
import pandas as pd

# Original lists
data_heads = ['funding_round_type', 'funding_round_code', 'raised_amount_usd', 'pre_money
data_types = ['categorical',
              'categorical', 'non-categorical', 'non-categorical', 'non-categorical',
              'categorical', 'categorical','categorical', 'categorical']

# Create a dictionary to store data heads and their corresponding types
data_info = {'DataHead': data_heads, 'DataType': data_types}

# Create a DataFrame from the dictionary
df_info = pd.DataFrame(data_info)

# Separate into categorical and non-categorical data
df_cat_heads = df_info[df_info['DataType'] == 'categorical']['DataHead'].tolist()
df_noncat_heads = df_info[df_info['DataType'] == 'non-categorical']['DataHead'].tolist()

# Add 'id' to both lists
df_cat_heads.append('id')
df_noncat_heads.append('id')

# Create categorical and non-categorical datasets
df_cat = df[df_cat_heads]
df_noncat = df[df_noncat_heads]

# Print the results
print("Categorical Data:")
print(df_cat)
print("\nNon-Categorical Data:")
print(df_noncat)
```

```
Categorical Data:
      funding_round_type funding_round_code  participants  is_first_round  \
0               series-b                  b             2               0
1                  angel              angel             2               0
2               series-a                  a             3               0
3               series-b                  b             4               0
4               series-b                  b             2               0
...                  ...                ...           ...             ...
```

```
52923          series-a                 a            1               1
52924           venture           partial            0               0
52925           venture      unattributed            0               0
52926          series-a                 a            0               1
52927           venture           partial            0               1

       is_last_round  cluster_number     id
0                  0               0      1
1                  1               0      2
2                  0               0      3
3                  0               0      4
4                  0               0      5
...              ...             ...    ...
52923              1               0  57948
52924              1               2  57949
52925              1               2  57950
52926              1               0  57951
52927              1               2  57952

[52928 rows x 7 columns]

Non-Categorical Data:
       raised_amount_usd  pre_money_valuation_usd  post_money_valuation_usd  \
0                8500000                        0                         0
1                 500000                        0                         0
2               12700000                115000000                         0
3               27500000                525000000                         0
4               10500000                        0                         0
...                  ...                      ...                       ...
52923            3000000                        0                         0
52924             570000                        0                         0
52925            2184100                        0                         0
52926             790783                        0                         0
52927             271250                        0                         0

          id
0          1
1          2
2          3
3          4
4          5
...      ...
52923  57948
52924  57949
52925  57950
52926  57951
52927  57952

[52928 rows x 4 columns]
```

```
# 1. Treatment of missing data

# 1.1. Missing data information

# Dataset used : df

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52928 entries, 0 to 52927
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   id                       52928 non-null  int64 
 1   funded_at                52680 non-null  object
 2   funding_round_type       52928 non-null  object
 3   funding_round_code       52928 non-null  object
 4   raised_amount_usd        52928 non-null  int64 
 5   pre_money_valuation_usd  52928 non-null  int64 
 6   post_money_valuation_usd 52928 non-null  int64 
 7   participants             52928 non-null  int64 
 8   is_first_round           52928 non-null  int64 
 9   is_last_round            52928 non-null  int64 
 10  cluster_number           52928 non-null  int64 
dtypes: int64(8), object(3)
memory usage: 4.4+ MB
```

```
variable_missing_data = df.isna().sum()
variable_missing_data
```

```
id                            0
funded_at                   248
funding_round_type            0
funding_round_code            0
raised_amount_usd             0
pre_money_valuation_usd       0
post_money_valuation_usd      0
participants                  0
is_first_round                0
is_last_round                 0
cluster_number                0
dtype: int64
```

```
record_missing_data = df.isna().sum(axis=1).sort_values(ascending=False).head(5)
record_missing_data
```

```
1770    1
2071    1
3011    1
3859    1
7238    1
dtype: int64
```

```
# Select categorical and non-categorical columns from the original DataFrame
df_cat_columns = df_info[df_info['DataType'] == 'categorical']['DataHead'].tolist()
df_noncat_columns = df_info[df_info['DataType'] == 'non-categorical']['DataHead'].tolist(

# Add 'id' to both lists
df_cat_columns.append('id')
df_noncat_columns.append('id')

# Select categorical and non-categorical columns from the DataFrame
df_cat = df[df_cat_columns]
df_noncat = df[df_noncat_columns]

# Exclude empty records
df_cat.dropna(axis=0, how='all', inplace=True)
df_noncat.dropna(axis=0, how='all', inplace=True)

# Exclude empty variables
df_cat.dropna(axis=1, how='all', inplace=True)
df_noncat.dropna(axis=1, how='all', inplace=True)


df_cat_mde = df_cat.copy()
df_noncat_mde = df_noncat.copy()
df_cat
```

	funding_round_type	funding_round_code	participants	is_first_round	is_last_
0	series-b	b	2	0	
1	angel	angel	2	0	
2	series-a	a	3	0	
3	series-b	b	4	0	
4	series-b	b	2	0	
...	...	...	...	...	
52923	series-a	a	1	1	
52924	venture	partial	0	0	
52925	venture	unattributed	0	0	
52926	series-a	a	0	1	
52927	venture	partial	0	1	

52928 rows × 7 columns

```
# 1.3. Missing Data Treatment

# 1.3.1 Impite Missing Categorical Data [Nominal | Ordinal] using Descriptive Statistics

# Dataset Used : df_cat_mde

si_cat = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
si_cat_fit = si_cat.fit_transform(df_cat_mde)
df_cat_mdi = pd.DataFrame(si_cat_fit, columns=df_cat_mde.columns)
df_cat_mdi
```

	funding_round_type	funding_round_code	participants	is_first_round	is_last_
0	series-b	b	2	0	
1	angel	angel	2	0	
2	series-a	a	3	0	
3	series-b	b	4	0	
4	series-b	b	2	0	
...	...	...	...	...	
52923	series-a	a	1	1	
52924	venture	partial	0	0	
52925	venture	unattributed	0	0	
52926	series-a	a	0	1	
52927	venture	partial	0	1	

52928 rows × 7 columns

```
df_cat_mdi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52928 entries, 0 to 52927
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   funding_round_type  52928 non-null  object
 1   funding_round_code  52928 non-null  object
 2   participants        52928 non-null  object
 3   is_first_round      52928 non-null  object
 4   is_last_round       52928 non-null  object
 5   cluster_number      52928 non-null  object
 6   id                  52928 non-null  object
dtypes: object(7)
memory usage: 2.8+ MB
```

```
from sklearn.impute import SimpleImputer

# 1.3.2.1 Impute missing Non-categorical data using descriptive statistical: central tend
# Dataset used: df_noncat_mde

# Create a SimpleImputer with the 'most_frequent' strategy
si_noncat = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

# Fit and transform the data
si_noncat_fit = si_noncat.fit_transform(df_noncat_mde)
df_noncat_mdi_si = pd.DataFrame(si_noncat_fit, columns=df_noncat_mde.columns)
df_noncat_mdi_si
```

	raised_amount_usd	pre_money_valuation_usd	post_money_valuation_usd	id
0	8500000	0	0	1
1	500000	0	0	2
2	12700000	115000000	0	3
3	27500000	525000000	0	4
4	10500000	0	0	5
...	...	...	...	...
52923	3000000	0	0	57948
52924	570000	0	0	57949
52925	2184100	0	0	57950
52926	790783	0	0	57951
52927	271250	0	0	57952

52928 rows × 4 columns

```
df_noncat_mdi_si.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52928 entries, 0 to 52927
Data columns (total 4 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   raised_amount_usd         52928 non-null  int64
 1   pre_money_valuation_usd   52928 non-null  int64
 2   post_money_valuation_usd  52928 non-null  int64
 3   id                        52928 non-null  int64
dtypes: int64(4)
memory usage: 1.6 MB
```

```
# Dataset used : df_cat_mdi

df_cat_mdi_code = df_cat_mdi.copy()

# Using Scikit learn : Ordinal Encoder (Superior)
oe = OrdinalEncoder()
oe_fit = oe.fit_transform(df_cat_mdi_code)
df_cat_code_oe = pd.DataFrame(oe_fit, columns=['funding_round_type_code', 'funding_round_
df_cat_code_oe
```

	funding_round_type_code	funding_round_code_code	participants_code	is_first_
0	6.0	2.0	2.0	
1	0.0	1.0	2.0	
2	5.0	0.0	3.0	
3	6.0	2.0	4.0	
4	6.0	2.0	2.0	
...	...	...	...	
52923	5.0	0.0	1.0	
52924	8.0	13.0	0.0	
52925	8.0	19.0	0.0	
52926	5.0	0.0	0.0	
52927	8.0	13.0	0.0	

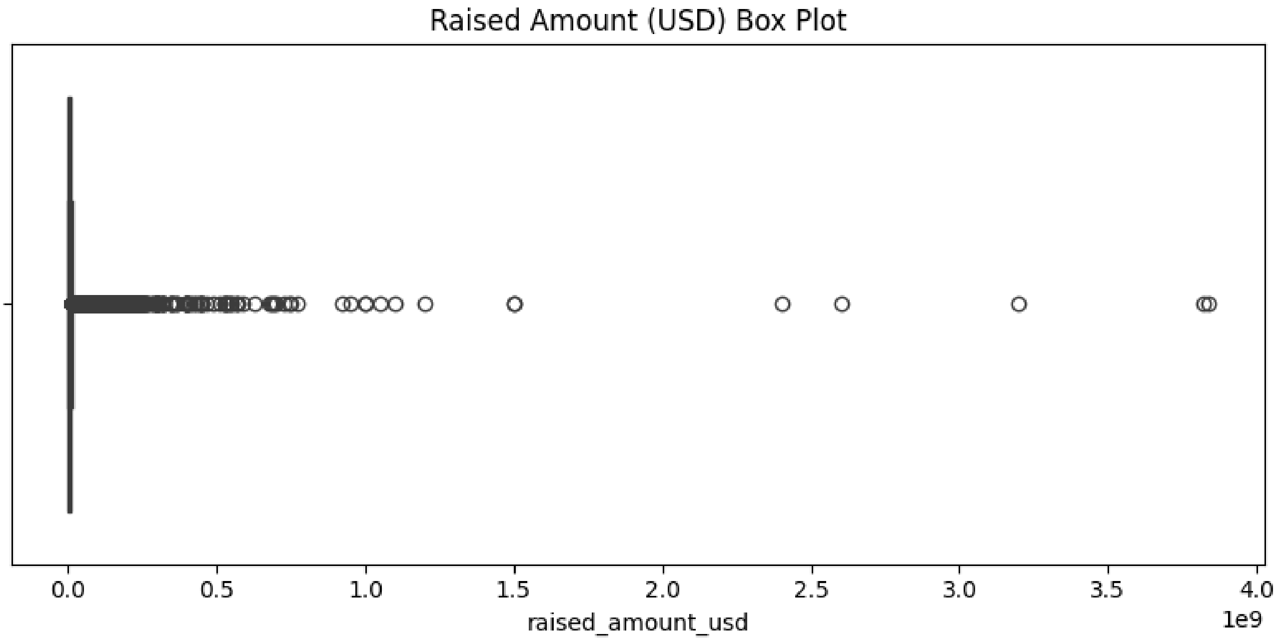
52928 rows × 7 columns

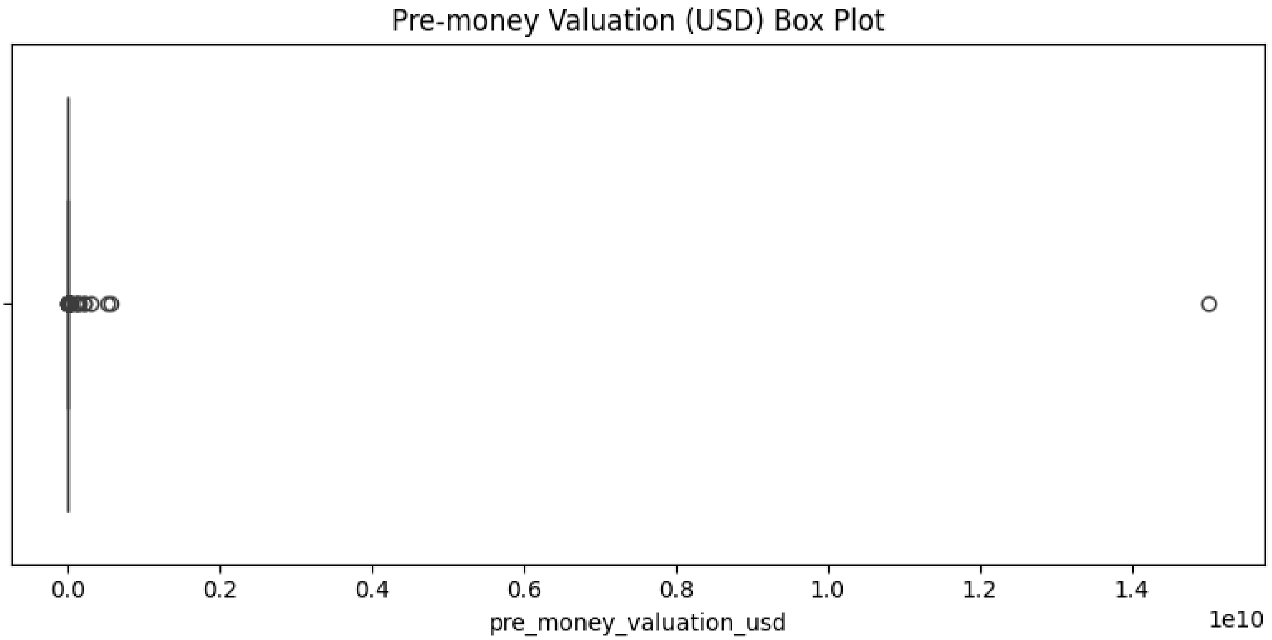
```
df_cat_mdi_code_oe = df_cat_mdi_code.join(df_cat_code_oe)
df_cat_mdi_code_oe
```

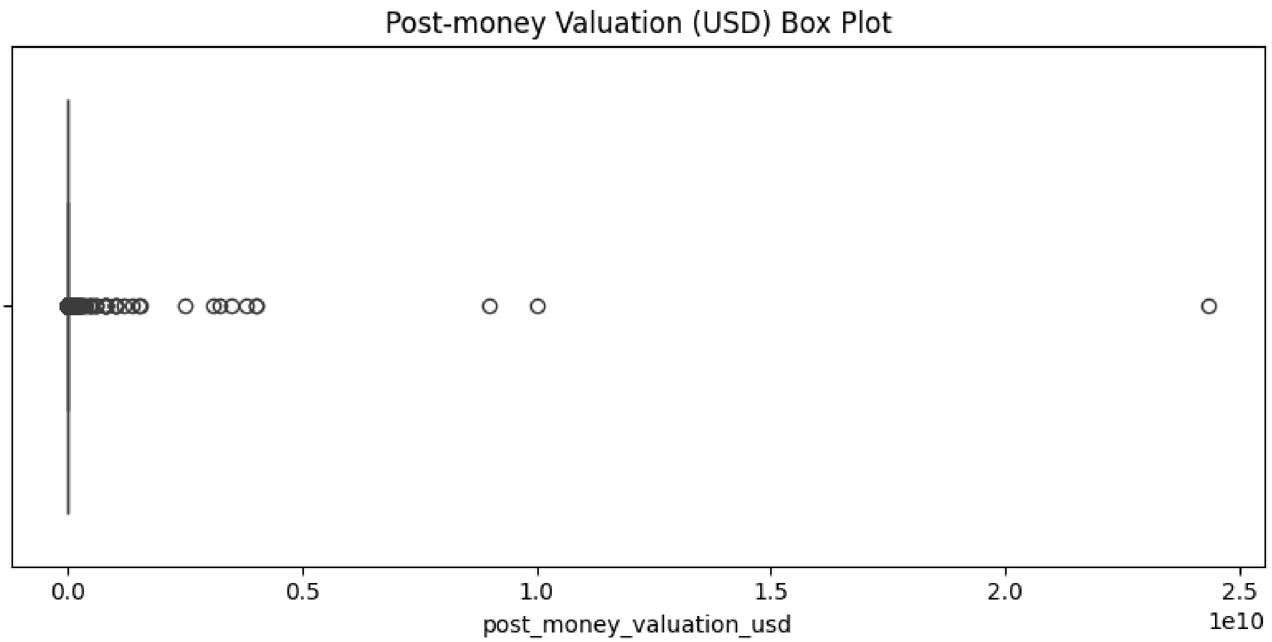
	funding_round_type	funding_round_code	participants	is_first_round	is_last_
0	series-b	b	2	0	
1	angel	angel	2	0	
2	series-a	a	3	0	
3	series-b	b	4	0	
4	series-b	b	2	0	
...	...	...	...	...	
52923	series-a	a	1	1	
52924	venture	partial	0	0	
52925	venture	unattributed	0	0	
52926	series-a	a	0	1	
52927	venture	partial	0	1	

52928 rows × 14 columns

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

# Assuming df_noncat_mdi_si is your DataFrame containing the relevant columns

# Create separate box plots for the specified columns using Seaborn
fig, axs = plt.subplots(3, 1, figsize=(8, 12))

# Box plot for 'raised_amount_usd'
sns.boxplot(x=df_noncat_mdi_si['raised_amount_usd'], ax=axs[0])
axs[0].set_title('Raised Amount (USD) Box Plot')

# Box plot for 'pre_money_valuation_usd'
sns.boxplot(x=df_noncat_mdi_si['pre_money_valuation_usd'], ax=axs[1])
axs[1].set_title('Pre-money Valuation (USD) Box Plot')

# Box plot for 'post_money_valuation_usd'
sns.boxplot(x=df_noncat_mdi_si['post_money_valuation_usd'], ax=axs[2])
axs[2].set_title('Post-money Valuation (USD) Box Plot')

plt.tight_layout()
plt.show()
```


Raised Amount (USD) Box Plot
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e9
raised_amount_usd


Pre-money Valuation (USD) Box Plot
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e10
pre_money_valuation_usd


Post-money Valuation (USD) Box Plot
0.0
0.5
1.0
1.5
2.0
2.5
1e10
post_money_valuation_usd

```
# 3. Data Transfrmation and Rescaling [Treatment of Outliers]

# Dataset used : df_noncat_mdi

# Scaling Variable : raised_amount_usd, pre_money_valuation_usd, post_money_valuation_usd

# 3.2.1. Normalization : Min-Max scaler
mms = MinMaxScaler()
mms_fit = mms.fit_transform(df_noncat_mdi_si[['raised_amount_usd', 'pre_money_valuation_u
df_noncat_minmax_norm = pd.DataFrame(mms_fit, columns=['raised_amount_usd_mmnorm', 'pre_m
df_noncat_minmax_norm
df_noncat_mdi_mmn = pd.merge(df_noncat_mdi_si, df_noncat_minmax_norm, left_index=True, ri
df_noncat_mdi_mmn
```

	raised_amount_usd	pre_money_valuation_usd	post_money_valuation_usd	id
0	8500000	0	0	1
1	500000	0	0	2
2	12700000	115000000	0	3
3	27500000	525000000	0	4
4	10500000	0	0	5
...	...	...	...	...
52923	3000000	0	0	57948
52924	570000	0	0	57949
52925	2184100	0	0	57950
52926	790783	0	0	57951
52927	271250	0	0	57952

52928 rows × 7 columns

```
# Pre-Processed Dataset

# Pre-Processed Categorical Data Subset
df_cat_ppd = df_cat_mdi_code_oe.copy()  # Preferred Data Subset

# Pre-Processed Non-Categorical Data Subset
df_noncat_ppd = df_noncat_mdi_mmn.copy()  # Preferred Data Subset

# Per-Processed Dataset using merge and 'id'
df_ppd = pd.merge(df_cat_ppd, df_noncat_ppd, on='id')

df_ppd
```

	funding_round_type	funding_round_code	participants	is_first_round	is_last_
0	series-b	b	2	0	
1	angel	angel	2	0	
2	series-a	a	3	0	
3	series-b	b	4	0	
4	series-b	b	2	0	
...	...	...	...	...	
52923	series-a	a	1	1	
52924	venture	partial	0	0	
52925	venture	unattributed	0	0	
52926	series-a	a	0	1	
52927	venture	partial	0	1	

52928 rows × 20 columns

Decision Tree

```
# Required Libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

```
# Subset startup_data based on Inputs and Output
startup_inputs = df_ppd[['id', 'funding_round_type_code', 'funding_round_code_code', 'par
startup_output = df_ppd[['cluster_number_code']]

startup_output
```

	cluster_number_code
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
52923	0.0
52924	2.0
52925	2.0
52926	0.0
52927	2.0

52928 rows × 1 columns

```
startup_inputs_names = startup_inputs.columns.tolist()
startup_inputs_names
```

```
['id',
 'funding_round_type_code',
 'funding_round_code_code',
 'participants_code',
 'is_first_round_code',
 'is_last_round_code',
 'raised_amount_usd_mmnorm',
 'pre_money_valuation_usd_mmnorm',
 'post_money_valuation_usd_mmnorm']
```

```
# Split the startup_data Subset into Training & Testing Sets
train_startup_inputs, test_startup_inputs, train_startup_output, test_startup_output = tr
```

```
# Decision Tree Clustering Model
# ------------------------------
# Decision Tree : Model (Training Subset)
dtc = DecisionTreeClassifier(criterion='gini', random_state=45041) # You can change the c
dtc_model = dtc.fit(train_startup_inputs, train_startup_output)
dtc_model
```

```
▾          DecisionTreeClassifier
DecisionTreeClassifier(random_state=45041)
```

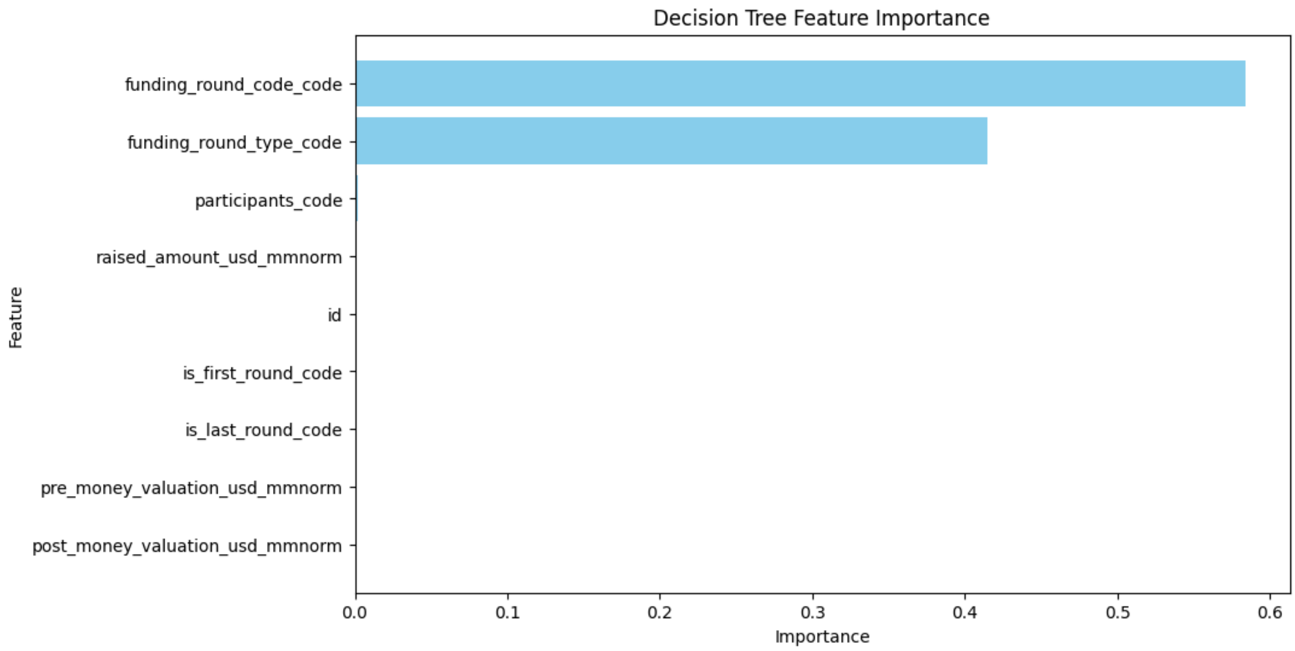
```
# Decision Tree : Model Rules
dtc_model_rules = export_text(dtc_model, feature_names=list(startup_inputs_names))  # Con
print(dtc_model_rules)
```

```
|--- funding_round_code_code <= 10.50
|   |--- funding_round_code_code <= 9.50
|   |   |--- class: 0.0
|   |--- funding_round_code_code >  9.50
|   |   |--- participants_code <= 1.50
|   |   |   |--- class: 2.0
|   |   |--- participants_code >  1.50
|   |   |   |--- raised_amount_usd_mmnorm <= 0.00
|   |   |   |   |--- id <= 16166.50
|   |   |   |   |   |--- class: 0.0
|   |   |   |   |--- id >  16166.50
|   |   |   |   |   |--- class: 2.0
|   |   |   |--- raised_amount_usd_mmnorm >  0.00
|   |   |   |   |--- class: 0.0
|--- funding_round_code_code >  10.50
|   |--- funding_round_type_code <= 4.50
|   |   |--- participants_code <= 13.50
|   |   |   |--- class: 1.0
|   |   |--- participants_code >  13.50
|   |   |   |--- funding_round_code_code <= 17.00
|   |   |   |   |--- class: 2.0
|   |   |   |--- funding_round_code_code >  17.00
|   |   |   |   |--- class: 1.0
|   |--- funding_round_type_code >  4.50
|   |   |--- class: 2.0
```

```
# Decision Tree : Feature Importance
dtc_imp_features = pd.DataFrame({'feature': startup_inputs.columns, 'importance': np.roun
dtc_imp_features.sort_values('importance', ascending=False, inplace=True)
print(dtc_imp_features)
```

```
                        feature  importance
2       funding_round_code_code     0.58363
1       funding_round_type_code     0.41443
3             participants_code     0.00172
6      raised_amount_usd_mmnorm     0.00012
0                            id     0.00009
4           is_first_round_code     0.00000
5            is_last_round_code     0.00000
7   pre_money_valuation_usd_mmnorm  0.00000
8  post_money_valuation_usd_mmnorm  0.00000
```

```
import matplotlib.pyplot as plt

# Assuming dtc_imp_features is already defined

# Plotting the feature importances
plt.figure(figsize=(10, 6))
plt.barh(dtc_imp_features['feature'], dtc_imp_features['importance'], color='skyblue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importance')
plt.gca().invert_yaxis()  # Invert y-axis to display the most important features on top
plt.show()
```




```
# Decision Tree : Model Prediction (Training Subset)
dtc_model_predict = dtc_model.predict(train_startup_inputs)
dtc_model_predict
```

```
array([0., 0., 0., ..., 0., 2., 0.])
```

```
# Decision Tree : Prediction (Testing Subset)
dtc_predict = dtc_model.predict(test_startup_inputs)
dtc_predict
```

```
array([2., 2., 0., ..., 0., 0., 0.])
```

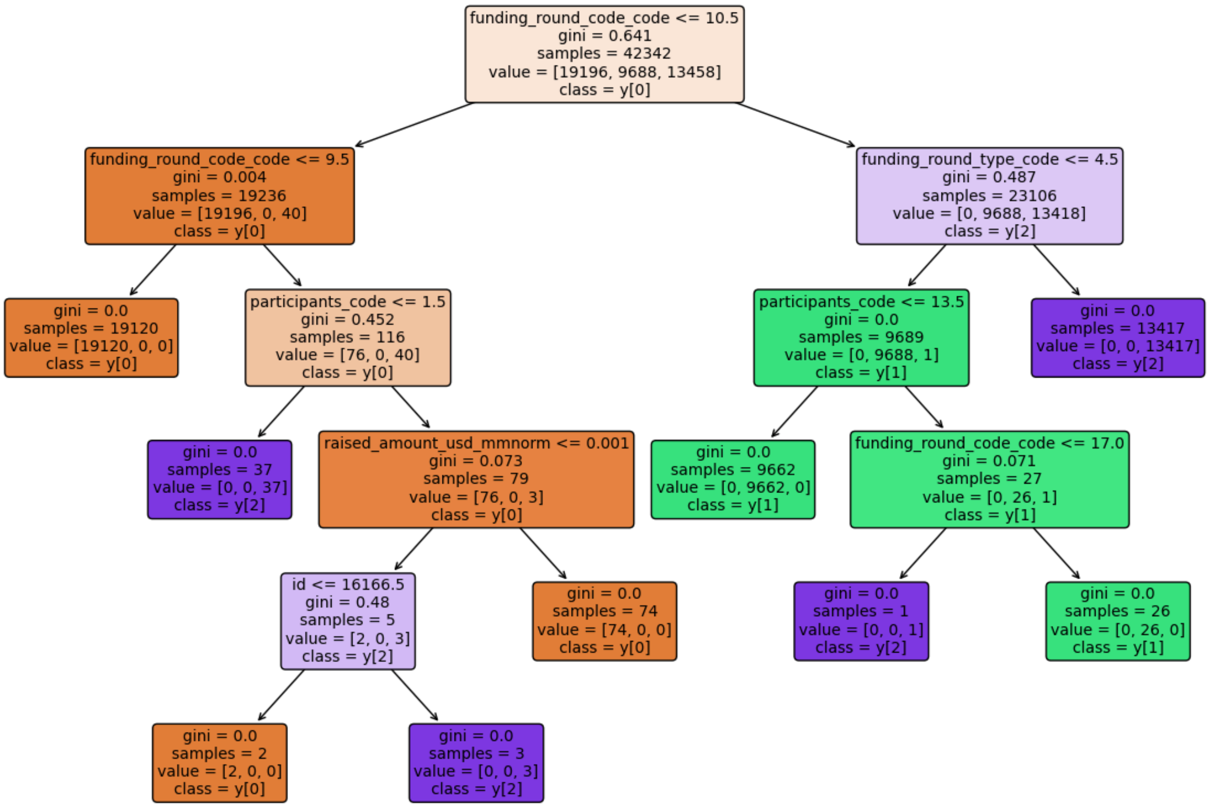
```
# Decision Tree : Model Evaluation (Training Subset)
dtc_model_conf_mat = pd.DataFrame(confusion_matrix(train_startup_output, dtc_model_predic
dtc_model_perf = classification_report(train_startup_output, dtc_model_predict)
print(dtc_model_perf)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     19196
         1.0       1.00      1.00      1.00      9688
         2.0       1.00      1.00      1.00     13458

    accuracy                           1.00     42342
   macro avg       1.00      1.00      1.00     42342
weighted avg       1.00      1.00      1.00     42342
```

```
from sklearn.metrics import confusion_matrix

# Decision Tree : Prediction Evaluation (Testing Subset)
dtc_predict_conf_mat = confusion_matrix(test_startup_output, dtc_predict)
dtc_predict_perf = classification_report(test_startup_output, dtc_predict)

print("Classification Report:")
print(dtc_predict_perf)

print("\nConfusion Matrix:")
print(dtc_predict_conf_mat)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      4799
         1.0       1.00      1.00      1.00      2422
         2.0       1.00      1.00      1.00      3365

    accuracy                           1.00     10586
   macro avg       1.00      1.00      1.00     10586
weighted avg       1.00      1.00      1.00     10586


Confusion Matrix:
[[4796    0    3]
 [   0 2422    0]
 [   0    0 3365]]
```

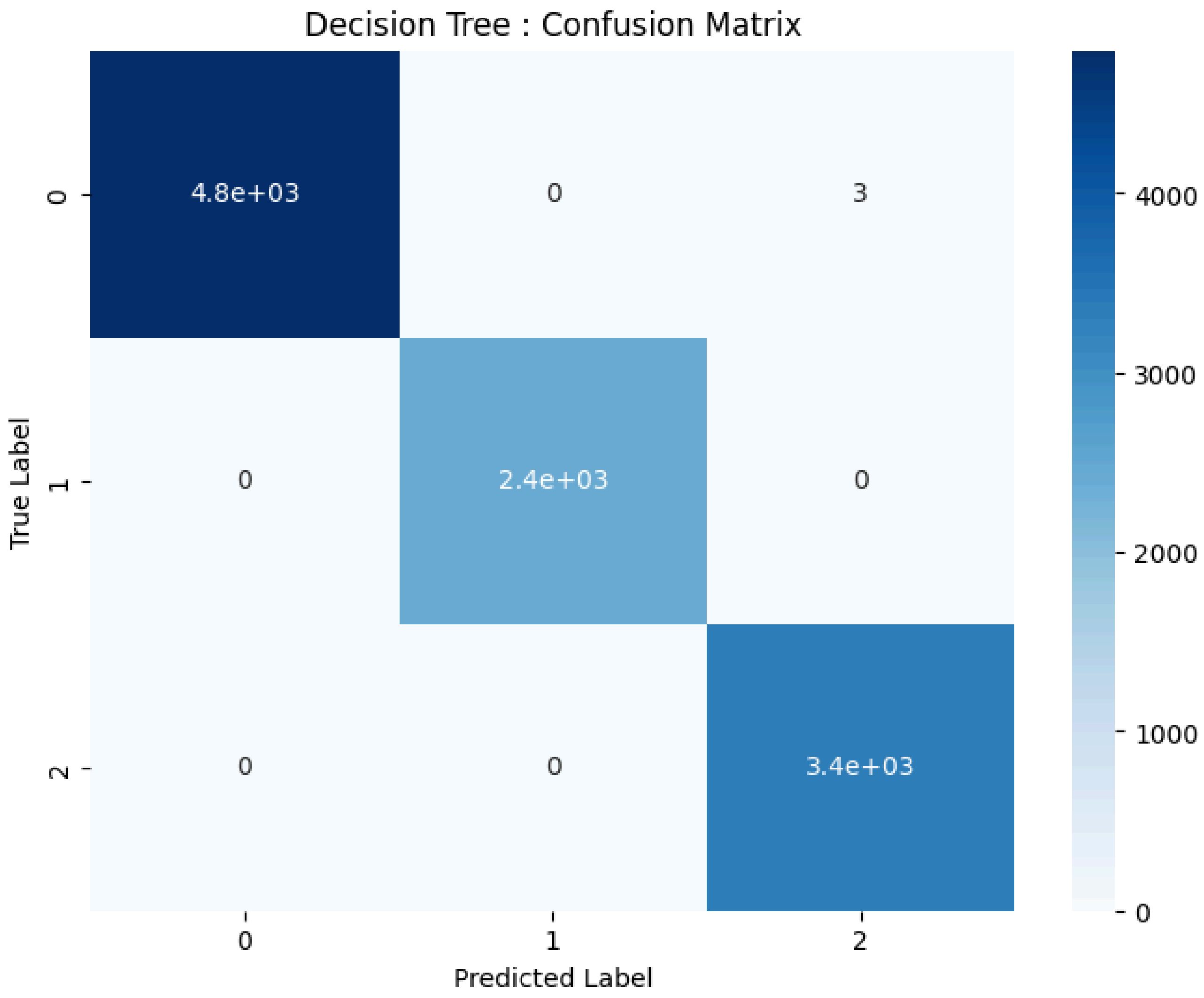
```
# Decision Tree : Plot [Training Subset]
plt.figure(figsize=(15, 10))
plot_tree(dtc_model, feature_names=startup_inputs.columns, filled=True, rounded=True, fon
plt.show()
```


funding_round_code_code <= 10.5
gini = 0.641
samples = 42342
value = [19196, 9688, 13458]
class = y[0]
funding_round_code_code <= 9.5
gini = 0.004
samples = 19236
value = [19196, 0, 40]
class = y[0]
funding_round_type_code <= 4.5
gini = 0.487
samples = 23106
value = [0, 9688, 13418]
class = y[2]
gini = 0.0
samples = 19120
value = [19120, 0, 0]
class = y[0]
participants_code <= 1.5
gini = 0.452
samples = 116
value = [76, 0, 40]
class = y[0]
participants_code <= 13.5
gini = 0.0
samples = 9689
value = [0, 9688, 1]
class = y[1]
gini = 0.0
samples = 13417
value = [0, 0, 13417]
class = y[2]
gini = 0.0
samples = 37
value = [0, 0, 37]
class = y[2]
raised_amount_usd_mmnorm <= 0.001
gini = 0.073
samples = 79
value = [76, 0, 3]
class = y[0]
gini = 0.0
samples = 9662
value = [0, 9662, 0]
class = y[1]
funding_round_code_code <= 17.0
gini = 0.071
samples = 27
value = [0, 26, 1]
class = y[1]
id <= 16166.5
gini = 0.48
samples = 5
value = [2, 0, 3]
class = y[2]
gini = 0.0
samples = 74
value = [74, 0, 0]
class = y[0]
gini = 0.0
samples = 1
value = [0, 0, 1]
class = y[2]
gini = 0.0
samples = 26
value = [0, 26, 0]
class = y[1]
gini = 0.0
samples = 2
value = [2, 0, 0]
class = y[0]
gini = 0.0
samples = 3
value = [0, 0, 3]
class = y[2]

```
# Decision Tree : Model (Training Subset)
import time
import psutil


start_time = time.time()
dtc = DecisionTreeClassifier(criterion='gini', random_state=45041,max_depth = 3) # Other
dtc_model = dtc.fit(train_startup_inputs, train_startup_output); dtc_model
end_time = time.time()

execution_time = end_time - start_time
print("Time taken:", execution_time, "seconds")

# Memory usage
process = psutil.Process()
memory_usage = process.memory_info().rss /1024  # in KB
print("Memory used:", memory_usage/1024, "KB")

print("Model:",dtc_model)
```

```
Time taken: 0.1212913990020752 seconds
Memory used: 689.24609375 KB
Model: DecisionTreeClassifier(max_depth=3, random_state=45041)
```

```
# Confusion Matrix : Plot [Testing Subset]
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(test_startup_output, dtc_predict), annot=True, cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Decision Tree : Confusion Matrix')
plt.show()
```



KNN

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


startup_inputs_train, startup_inputs_test, startup_output_train, startup_output_test = tr


k = 5  # Specify the number of neighbors (k)
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(startup_inputs_train, startup_output_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
startup_output_pred = knn.predict(startup_inputs_test)
```

```
accuracy = accuracy_score(startup_output_test, startup_output_pred)
print(f'Accuracy: {accuracy}')

classification_rep = classification_report(startup_output_test, startup_output_pred)
print(f'Classification Report:\n{classification_rep}')

conf_mat = confusion_matrix(startup_output_test, startup_output_pred)
print(f'Confusion Matrix:\n{conf_mat}')
```

```
Accuracy: 0.8727564708105044
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97      4915
         1.0       0.77      0.72      0.74      2350
         2.0       0.80      0.84      0.82      3321

    accuracy                           0.87     10586
   macro avg       0.85      0.84      0.84     10586
weighted avg       0.87      0.87      0.87     10586

Confusion Matrix:
[[4757   52  106]
 [  74 1683  593]
 [  69  453 2799]]
```

```
# Example of tuning the number of neighbors
k_values = [7, 9, 11, 13]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(startup_inputs_train, startup_output_train)
    y_pred = knn.predict(startup_inputs_test)
    accuracy = accuracy_score(startup_output_test, startup_output_pred)
    print(f'Accuracy for k={k}: {accuracy}')
```

```
Accuracy for k=7: 0.8727564708105044
Accuracy for k=9: 0.8727564708105044
Accuracy for k=11: 0.8727564708105044
Accuracy for k=13: 0.8727564708105044
```

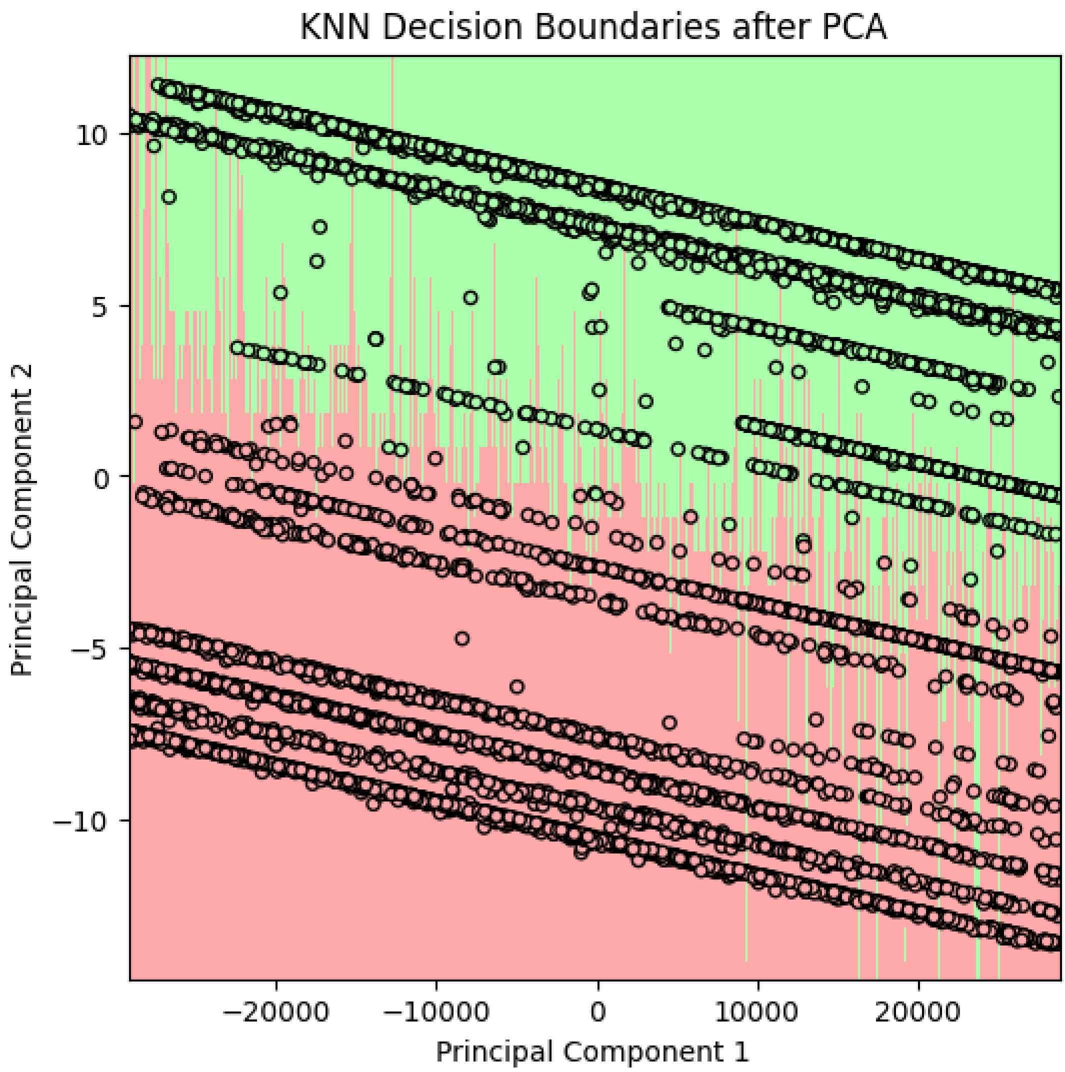
```
import numpy as np

# Shuffle the values of each feature and measure the change in accuracy
original_accuracy = accuracy_score(startup_output_test, startup_output_pred)
variable_importances = {}

for feature in startup_inputs.columns:
    shuffled_inputs = startup_inputs_test.copy()
    np.random.shuffle(shuffled_inputs[feature].values)
    shuffled_pred = knn.predict(shuffled_inputs)
    shuffled_accuracy = accuracy_score(startup_output_test, shuffled_pred)
    variable_importances[feature] = original_accuracy - shuffled_accuracy

# Sort the variable importances
sorted_variable_importances = dict(sorted(variable_importances.items(), key=lambda item:

# Print or plot the variable importances
for feature, importance in sorted_variable_importances.items():
    print(f"{feature}: {importance}")
```

```
funding_round_code_code: 0.3942943510296618
funding_round_type_code: 0.09210277725297566
id: 0.08728509351974312
pre_money_valuation_usd_mmnorm: 0.0
post_money_valuation_usd_mmnorm: 0.0
raised_amount_usd_mmnorm: -0.00018892877385223716
is_last_round_code: -0.00028339316077841126
is_first_round_code: -0.0004723219346306484
participants_code: -0.0006612507084828856
```

```
import time
import resource

startup_inputs_train, startup_inputs_test, startup_output_train, startup_output_test = tr

# Start timing
start_time = time.time()

k = 5  # Specify the number of neighbors (k)
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(startup_inputs_train, startup_output_train)

startup_output_pred = knn.predict(startup_inputs_test)

accuracy = accuracy_score(startup_output_test, startup_output_pred)

classification_rep = classification_report(startup_output_test, startup_output_pred)

conf_mat = confusion_matrix(startup_output_test, startup_output_pred)

# End timing
end_time = time.time()

# Calculate elapsed time
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")

# Memory usage
memory_usage = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss / 1024  # in kilobytes
print(f"Memory usage: {memory_usage} kilobytes")
```

```
Elapsed time: 1.7428269386291504 seconds
Memory usage: 1150.27734375 kilobytes
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA, IncrementalPCA
from matplotlib.colors import ListedColormap

# Define colormap for the plot
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])

# Plot decision boundaries after PCA
def plot_decision_boundaries_pca(X, y, classifier, title):
    # Perform PCA to reduce dimensions for visualization
    pca = IncrementalPCA(n_components=2, batch_size=100)
    x_pca = pca.fit_transform(X)

    # Create a meshgrid for the reduced feature space
    x_min, x_max = x_pca[:, 0].min() - 1, x_pca[:, 0].max() + 1
    y_min, y_max = x_pca[:, 1].min() - 1, x_pca[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 1), np.arange(y_min, y_max, 1))

    # Predict the class for each grid point
    Z = classifier.predict(pca.inverse_transform(np.c_[xx.ravel(), yy.ravel()]))
    Z = Z.reshape(xx.shape)

    # Plot the decision boundaries
    plt.figure(figsize=(6, 6))  # Adjust figure size as needed
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot the training points
    plt.scatter(x_pca[:, 0], x_pca[:, 1], c=y, cmap=cmap_light, edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(title)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()

# Assuming X_train is a 2D array with 10 features for visualization
plot_decision_boundaries_pca(startup_inputs_test.values, startup_output_test.values, knn,
```



LOGISTIC REGRESSION

```
startup_output = df_ppd[['cluster_number_code']]
startup_output
```

	cluster_number_code
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
52923	0.0
52924	2.0
52925	2.0
52926	0.0
52927	2.0

52928 rows × 1 columns

```
startup_inputs_names = startup_inputs.columns; print("Input Names: ",startup_inputs_names
startup_output_labels = startup_output['cluster_number_code'].unique().astype(str); print
```

```
Input Names:  Index(['id', 'funding_round_type_code', 'funding_round_code_code',
       'participants_code', 'is_first_round_code', 'is_last_round_code',
       'raised_amount_usd_mmnorm', 'pre_money_valuation_usd_mmnorm',
       'post_money_valuation_usd_mmnorm'],
      dtype='object')
Output Label:  ['0.0' '2.0' '1.0']
```

```
from sklearn.linear_model import LogisticRegression
import time
import psutil

start_time = time.time()
lr = LogisticRegression(random_state=45041)
lr_model = lr.fit(train_startup_inputs, train_startup_output)
end_time = time.time()

execution_time = end_time - start_time
print("Time taken:", execution_time, "seconds")

# Memory usage
process = psutil.Process()
memory_usage = process.memory_info().rss /1024  # in KB
print("Memory used:", memory_usage/1024, "KB")

print("Model:",lr_model)
```

```
Time taken: 1.870229721069336 seconds
Memory used: 723.6875 KB
```

```
Model: LogisticRegression(random_state=45041)
```

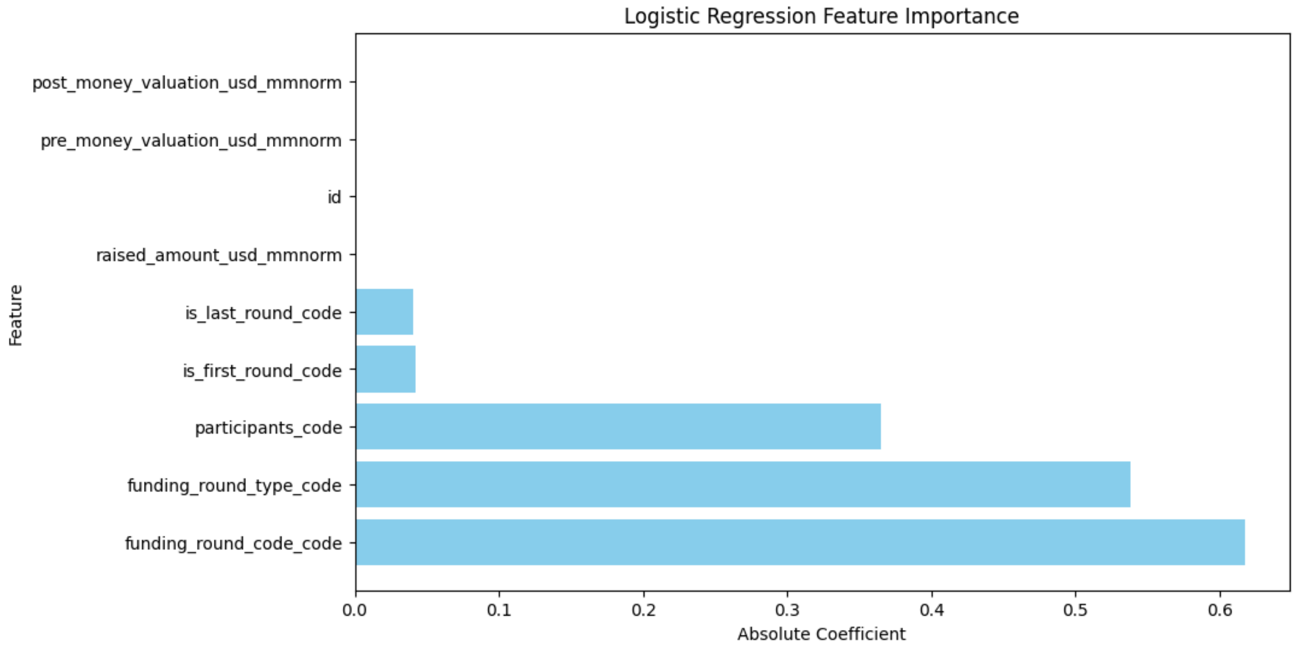
```
# Extract coefficients and intercept
coefficients = lr_model.coef_
intercept = lr_model.intercept_

# Display coefficients
print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

```
Coefficients: [[ 9.63419531e-05  5.38362874e-01 -6.17540905e-01  3.65510599e-01
   4.18338099e-02  4.05787641e-02  5.40746450e-04  3.53965155e-05
   3.67727364e-06]
 [-3.81683177e-05 -7.26010701e-01  3.70267142e-01 -1.01551506e-01
  -3.14935901e-03  7.24147793e-03 -1.21009746e-04 -4.00931814e-06
  -3.92143874e-06]
 [-5.81736355e-05  1.87647826e-01  2.47273764e-01 -2.63959093e-01
  -3.86844509e-02 -4.78202420e-02 -4.19736705e-04 -3.13871974e-05
   2.44165097e-07]]
Intercept: [ 0.12707044 -0.02641035 -0.10066009]
```

```
# Extract coefficients and feature names
coefficients = lr_model.coef_[0]
feature_names = startup_inputs.columns

# Create a DataFrame to store coefficients and feature names
lr_imp_features = pd.DataFrame({'feature': feature_names, 'coefficient': coefficients})

# Sort features by absolute coefficient value
lr_imp_features['abs_coefficient'] = np.abs(lr_imp_features['coefficient'])
lr_imp_features.sort_values('abs_coefficient', ascending=False, inplace=True)

# Display the DataFrame
print(lr_imp_features)
```

```
                          feature  coefficient  abs_coefficient
2          funding_round_code_code    -0.617541         0.617541
1          funding_round_type_code     0.538363         0.538363
3                participants_code     0.365511         0.365511
4              is_first_round_code     0.041834         0.041834
5               is_last_round_code     0.040579         0.040579
6         raised_amount_usd_mmnorm     0.000541         0.000541
0                               id     0.000096         0.000096
7    pre_money_valuation_usd_mmnorm     0.000035         0.000035
8   post_money_valuation_usd_mmnorm     0.000004         0.000004
```

```
import matplotlib.pyplot as plt

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(lr_imp_features['feature'], lr_imp_features['abs_coefficient'], color='skyblue')
plt.xlabel('Absolute Coefficient')
plt.ylabel('Feature')
plt.title('Logistic Regression Feature Importance')
plt.show()
```



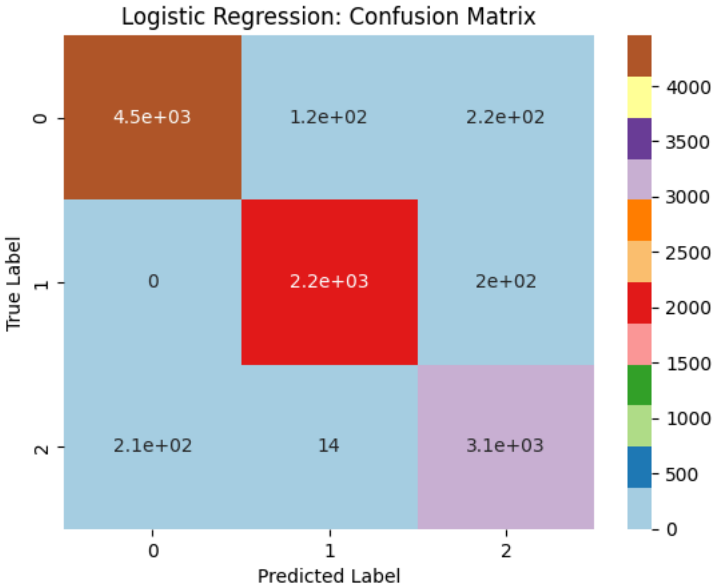

```
# Logistic Regression: Prediction (Testing Subset)
lr_predict = lr_model.predict(test_startup_inputs)
print(lr_predict)
```

```
[2. 2. 0. ... 0. 2. 0.]
```

```
# Logistic Regression: Prediction Evaluation (Testing Subset)
lr_predict_conf_mat = pd.DataFrame(confusion_matrix(test_startup_output, lr_predict))
print(lr_predict_conf_mat)

lr_predict_perf = classification_report(test_startup_output, lr_predict)
print(lr_predict_perf)
```

```
      0     1     2
0  4454   120   225
1     0  2222   200
2   207    14  3144
              precision    recall  f1-score   support

         0.0       0.96      0.93      0.94      4799
         1.0       0.94      0.92      0.93      2422
```

```
# Confusion Matrix : Plot [Testing Subset]
ax = plt.axes()
sns.heatmap(lr_predict_conf_mat, annot=True, cmap='Paired')
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Logistic Regression: Confusion Matrix')
plt.show()
```

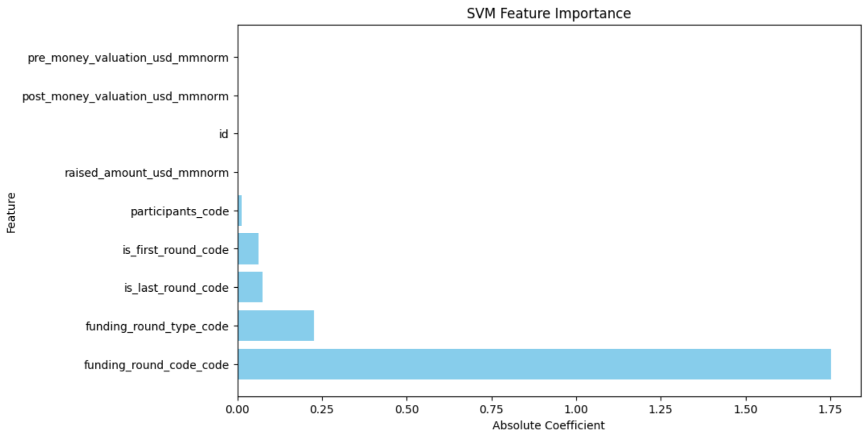


SVM

```
from sklearn.svm import SVC

start_time = time.time()
# Define the SVM model with a linear kernel and regularization parameter C=45041
svm_model = SVC(kernel='linear', C=1)

# Train the SVM model on the training dataset
svm_model.fit(train_startup_inputs, train_startup_output)
end_time = time.time()

execution_time = end_time - start_time
print("Time taken:", execution_time, "seconds")

# Memory usage
process = psutil.Process()
memory_usage = process.memory_info().rss / 1024  # in KB
print("Memory used:", memory_usage / 1024, "KB")
```

```
Time taken: 191.37723302841187 seconds
Memory used: 690.22265625 KB
```

```
def plot_feature_importance_svm(model, feature_names):
    # Get coefficients from the model
    coefficients = model.coef_[0]

    # Create DataFrame to store feature importance
    svm_imp_features = pd.DataFrame({'feature': feature_names, 'coefficient': coefficients/1000})

    # Sort features by absolute coefficient values
    svm_imp_features['abs_coefficient'] = np.abs(svm_imp_features['coefficient'])
    svm_imp_features.sort_values('abs_coefficient', ascending=False, inplace=True)

    # Plot feature importance
    plt.figure(figsize=(10, 6))
    plt.barh(svm_imp_features['feature'], svm_imp_features['abs_coefficient'], color='skyblue')
    plt.xlabel('Absolute Coefficient')
    plt.ylabel('Feature')
    plt.title('SVM Feature Importance')
    plt.show()

    # Return the DataFrame containing feature importance values
    return svm_imp_features

# Call the function and store the returned DataFrame
svm_feature_importance_df = plot_feature_importance_svm(svm_model, startup_inputs_names)
print(svm_feature_importance_df)
```




```
                           feature  coefficient  abs_coefficient
2          funding_round_code_code    -1.751690         1.751690
1          funding_round_type_code     0.227470         0.227470
5               is_last_round_code    -0.073870         0.073870
4              is_first_round_code    -0.063772         0.063772
3                participants_code    -0.013187         0.013187
6         raised_amount_usd_mmnorm     0.001287         0.001287
0                               id     0.000094         0.000094
8  post_money_valuation_usd_mmnorm    -0.000003         0.000003
7   pre_money_valuation_usd_mmnorm     0.000000         0.000000
```

```
# SVM Prediction (Testing Subset)
svm_predict = svm_model.predict(test_startup_inputs)
print(svm_predict)
```

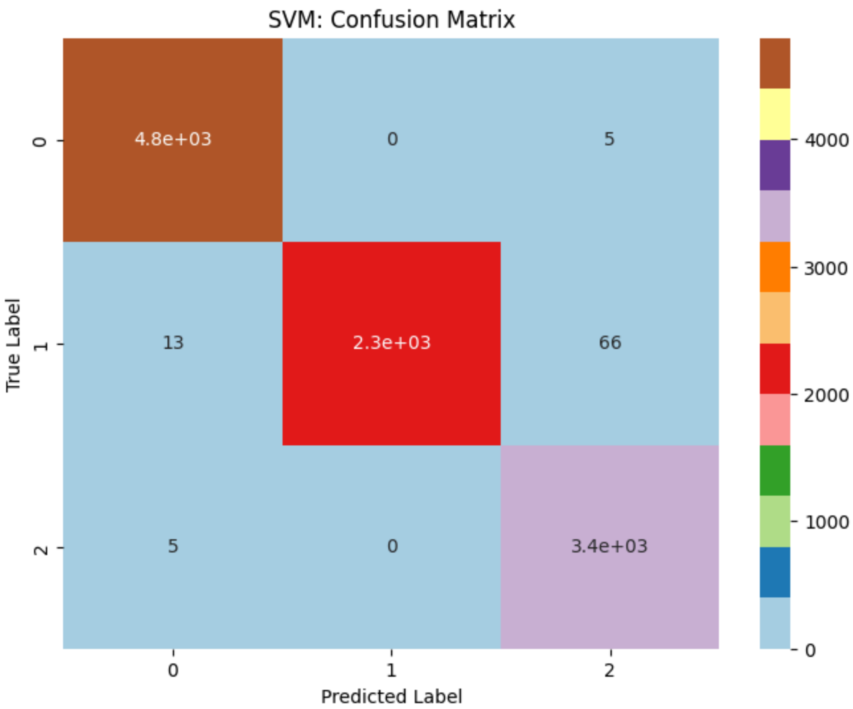
```
[2. 2. 0. ... 0. 0. 0.]
```

```
# SVM Prediction Evaluation (Testing Subset)
svm_predict_conf_mat = pd.DataFrame(confusion_matrix(test_startup_output, svm_predict))
print(svm_predict_conf_mat)

svm_predict_perf = classification_report(test_startup_output, svm_predict)
print(svm_predict_perf)
```

```
      0     1    2
0  4794     0    5
1    13  2343   66
```

```
   2     5     0  3360
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      4799
         1.0       1.00      0.97      0.98      2422
         2.0       0.98      1.00      0.99      3365

    accuracy                           0.99     10586
   macro avg       0.99      0.99      0.99     10586
weighted avg       0.99      0.99      0.99     10586
```

```
# Compute confusion matrix for SVM
svm_conf_mat = confusion_matrix(test_startup_output, svm_predict)


# Plot confusion matrix for SVM
plt.figure(figsize=(8, 6))
sns.heatmap(svm_conf_mat, annot=True, cmap='Paired')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('SVM: Confusion Matrix')
plt.show()
```




```
import pandas as pd

# Specify the file path where you want to save the CSV file
file_path = 'df_ppd.csv'

# Save the DataFrame as a CSV file
df_ppd.to_csv(file_path, index=False)

print(f"Dataset saved to {file_path}")

    Dataset saved to df_ppd.csv
```