National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Nixon Balu
Student ID: x20247788

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nixon Balu		
Student ID:	x20247788		
Programme:	M.Sc. Data Analytics	**Year:**	2022
Module:	Research Project		
Supervisor:	Christian Horn		
Submission Due Date:	1st February 2023		
Project Title:	Indian Start-up's Success Prediction Using Machine Learning		
Word Count:	711	**Page Count:**	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Nixon Balu

Date: 31st January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nixon Balu
x20247788

1 Introduction

This document gives a clear demonstration of the hardware and software requirements along with the various steps taken to successfully implement the research project - *Indian Start-up's Success Prediction Using Machine Learning.*

2 System Configuration

2.1 Hardware Requirements

- System OS: Windows 10
- Processor: i5
- RAM: 8 GB

2.2 Software Requirements

The project is implemented in Jupyter Notebook version 6.4.5 with the Python kernel version 3.9.7. Jupyter notebook can be installed using the Anaconda environment (fig. 1) which by default installs the python kernel as well.

3 Project Implementation

The first step will be importing all the necessary libraries for the project (fig. 2).

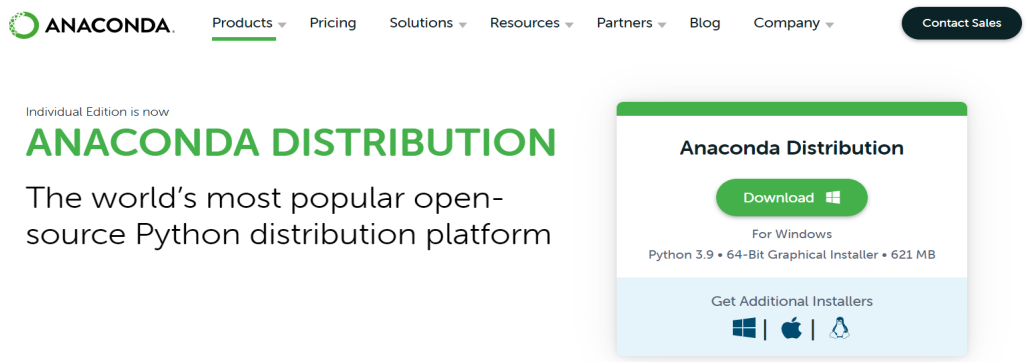

Figure 1: Installing Jupyter Notebook Using Anaconda

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from plotly import express as px
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import f1_score
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

Figure 2: Importing Necessary Libraries

3.1 Data Collection

The data is obtained from Crunchbase [1] platform through a subscription for academic research. The dataset consists of multiple CSV files out of which only two files are selected and loaded in separate Pandas dataframes(fig. 3).

[1]https://crunchbase.wufoo.com/forms/s1qn010s0t6fkq6/

```
funding_df = pd.read_csv('funding_rounds.csv')
org_df = pd.read_csv('organizations.csv')
```

Figure 3: Loading Files to Pandas

3.2 Feature Selection

Specific features (fig. 4) are selected from both the dataframes and are subjected to pre-processing.

```
funding1_df = funding_df[['investment_type','announced_on', 'raised_amount_usd', 'org_name']]
org_df = org_df[['name','country_code','founded_on','category_list','total_funding_usd','status']]
```

Figure 4: Feature Selection

3.3 Data Pre-processing

As there are multiple investment types in the funding rounds dataframe, only 4 key investments are chosen and loaded in separate dataframes (fig. 5). After this,

```
# seperating selected investment types to different dataframes

seed_df = funding1_df[funding1_df['investment_type']=='seed']
seriesA_df = funding1_df[funding1_df['investment_type']=='series_a']
seriesB_df = funding1_df[funding1_df['investment_type']=='series_b']
seriesC_df = funding1_df[funding1_df['investment_type']=='series_c']
```

Figure 5: Selecting Investment Types from Funding Rounds

renaming of the columns is done for easier understanding (fig. 6), followed by dropping of duplicate entries of organizations (fig. 7). Similarly, duplicate entries in the organizations dataframe are dropped.

This is followed by filling the missing values in the raised_amount columns of each investment type with their corresponding mean values (fig. 8).

The next step is separating categories from the *category_list* attribute in the organization dataframe and storing them in a new feature (fig. 9).

```
# renaming columns

seed_df.columns = ['seed','seed_date', 'raised_amount_seed', 'name']
seriesA_df.columns = ['series_a','series_a_date', 'raised_amount_seriesA', 'name']
seriesB_df.columns = ['series_b','series_b_date', 'raised_amount_seriesB', 'name']
seriesC_df.columns = ['series_c','series_c_date', 'raised_amount_seriesC', 'name']
```

Figure 6: Renaming Columns of Individual Dataframes of Investment Types

```
# dropping duplicate values

seed_df.drop_duplicates(subset = 'name', inplace = True)
seriesA_df.drop_duplicates(subset = 'name', inplace = True)
seriesB_df.drop_duplicates(subset = 'name', inplace = True)
seriesC_df.drop_duplicates(subset = 'name', inplace = True)
```

Figure 7: Dropping Duplicates

The separate dataframes are now merged together (fig. 10). The correspond-

```
# filling missing values in raised amount for individual investment types with their corresponding mean values

seed_mean = seed_df['raised_amount_seed'].mean()
seriesA_mean = seriesA_df['raised_amount_seriesA'].mean()
seriesB_mean = seriesB_df['raised_amount_seriesB'].mean()
seriesC_mean = seriesC_df['raised_amount_seriesC'].mean()
seed_df['raised_amount_seed'].fillna(value=seed_mean, inplace=True)
seriesA_df['raised_amount_seriesA'].fillna(value=seriesA_mean, inplace=True)
seriesB_df['raised_amount_seriesB'].fillna(value=seriesB_mean, inplace=True)
seriesC_df['raised_amount_seriesC'].fillna(value=seriesC_mean, inplace=True)
```

Figure 8: Imputing Missing Values

ing values of organizations that received the different types of investments are replaced with 1 and if no investment received, with 0 (fig.11). Merging of final_fund with the organization dataframe is done (fig. 12). The datatype of the feature *founded_on* is converted to *datetime64* datatype and the missing values in them are dropped (fig. 13). Similarly, the other attributes that hold date information are converted to the *datetime64* datatype.

```
org_df.category_list.value_counts()
```

```
Software                                                                    15789
Health Care                                                                 15541
Information Technology                                                      14608
Real Estate                                                                 13706
Manufacturing                                                               13474
                                                                            ...
Cloud Management,Document Management,Office Administration                      1
Advice,Corporate Training,Environmental Engineering,Management Consulting,Project Management    1
Apps,E-Commerce,Financial Services,FinTech,Information Services                 1
Agriculture,AgTech,Artificial Intelligence,Big Data,Mobile Apps                 1
Consumer,Finance,Financial Services,Real Estate                                 1
Name: category_list, Length: 549715, dtype: int64
```

```
# to separate categories from category_list

new = org_df["category_list"].str.split(",", n = 1, expand = True)
org_df["category"]= new[0]
```

Figure 9: Creating a Separate Feature for Category of Organization

```
# merging the different investment type dataframes

fund_2 = pd.merge(left=seed_df, right=seriesA_df, how='outer', left_on='name', right_on='name')
fund_3 = pd.merge(left=fund_2, right=seriesB_df, how='left', left_on='name', right_on='name')
final_fund = pd.merge(left=fund_3, right=seriesC_df, how='left', left_on='name', right_on='name')
```

Figure 10: Merging of Investment Types' Dataframes

```
# replacing if received seed, series_a, series_b and series_c received to 1

final_fund.replace('seed', '1', inplace = True)
final_fund.replace('series_a', '1', inplace = True)
final_fund.replace('series_b', '1', inplace = True)
final_fund.replace('series_c', '1', inplace = True)
```

```
# replacing seed, series_a, series_b and series_c missing values to 0

final_fund.fillna(value = {'seed': '0','series_a':'0','series_b':'0','series_c':'0' }, inplace = True)
```

Figure 11: Replacing Investments Received or not with 1 and 0

3.4 Feature Engineering

New features are created from the attributes that had dates to get the corresponding duration between different rounds of investments in months (fig. 14). Subsetting

```
# merging final_fund with org_df

final_df = pd.merge(left=org_df, right=final_fund, how='left', left_on='name', right_on='name')
```

Figure 12: Merging with Organization Dataframe

```
final_df.founded_on = pd.to_datetime(final_df.founded_on, errors = 'coerce')
```

Figure 13: Datatype Conversion of Founded_on Attribute

of the dataframe is done to retain only those startups that have no missing values for *total_funding_usd* (fig. 15). All the numeric features with null values are replaced with 0 (fig. 16). Those records with negative values for the duration are removed.

The new features are converted to *int64* datatypes to create new set of categor-

```
final_df['founding_to_seed_months'] = (final_df['seed_date'] - final_df['founded_on']) / np.timedelta64(1, 'M')
final_df['founding_to_seriesA_months'] = (final_df['series_a_date'] - final_df['founded_on']) / np.timedelta64(1, 'M')
final_df['seriesA_to_seriesB_months'] = (final_df['series_b_date'] - final_df['series_a_date']) / np.timedelta64(1, 'M')
final_df['seriesB_to_seriesC_months'] = (final_df['series_c_date'] - final_df['series_b_date']) / np.timedelta64(1, 'M')
```

Figure 14: New Features for Duration Between Investments in Months

```
final3_df = final2_df[final2_df['total_funding_usd']>= 0]
```

Figure 15: Subsetting Dataframe with no Missing Values in *Total_Funding_USD*

ical features (fig. 17). All the features that are no longer required are dropped (fig. 18). The target variable is then defined and the final dataframe is created (fig. 19). Another dataframe with only Indian companies are also created (fig. 20).

3.5 Exploratory Data Analysis

The balance between the classes of start-ups globally and Indian start-ups is visualized using the code in fig. 21. Top 10 industries are seen using fig. 22. Additionally, top 10 countries in the world with most number of start-ups are seen using the code in fig. 23.

```
final3_df.fillna(value = {'seed': '0','series_a':'0','series_b':'0','series_c':'0' }, inplace = True)
```

```
final3_df.raised_amount_seed = final3_df.raised_amount_seed.fillna(0)
final3_df.raised_amount_seriesA = final3_df.raised_amount_seriesA.fillna(0)
final3_df.raised_amount_seriesB = final3_df.raised_amount_seriesB.fillna(0)
final3_df.raised_amount_seriesC = final3_df.raised_amount_seriesC.fillna(0)
```

```
final3_df.founding_to_seed_months = final3_df.founding_to_seed_months.fillna(0)
final3_df.founding_to_seriesA_months = final3_df.founding_to_seriesA_months.fillna(0)
final3_df.seriesA_to_seriesB_months = final3_df.seriesA_to_seriesB_months.fillna(0)
final3_df.seriesB_to_seriesC_months = final3_df.seriesB_to_seriesC_months.fillna(0)
```

Figure 16: Filling Null Values with 0

```
# changing float to int to suport easy grouping of data into bins

final4_df.founding_to_seed_months = final4_df.founding_to_seed_months.astype('int64')
final4_df.founding_to_seriesA_months = final4_df.founding_to_seriesA_months.astype('int64')
final4_df.seriesA_to_seriesB_months = final4_df.seriesA_to_seriesB_months.astype('int64')
final4_df.seriesB_to_seriesC_months = final4_df.seriesB_to_seriesC_months.astype('int64')
```

```
final4_df.loc[final4_df['founding_to_seed_months'].between(0, 12, 'right'), 'seed_duration'] = 'F2S_1'
final4_df.loc[final4_df['founding_to_seed_months'].between(12, 24, 'right'), 'seed_duration'] = 'F2S_2'
final4_df.loc[final4_df['founding_to_seed_months'].between(24, 36, 'right'), 'seed_duration'] = 'F2S_3'
final4_df.loc[final4_df['founding_to_seed_months'].between(36, 10000, 'right'), 'seed_duration'] = 'F2S_4'
final4_df.loc[final4_df['founding_to_seed_months'].between(0, 0, 'both'), 'seed_duration'] = 'F2S_5'
```

```
final4_df.loc[final4_df['founding_to_seriesA_months'].between(0, 12, 'right'), 'to_seriesA'] = 'F2A_1'
final4_df.loc[final4_df['founding_to_seriesA_months'].between(12, 24, 'right'), 'to_seriesA'] = 'F2A_2'
final4_df.loc[final4_df['founding_to_seriesA_months'].between(24, 36, 'right'), 'to_seriesA'] = 'F2A_3'
final4_df.loc[final4_df['founding_to_seriesA_months'].between(36, 10000, 'right'), 'to_seriesA'] = 'F2A_4'
final4_df.loc[final4_df['founding_to_seriesA_months'].between(0, 0, 'both'), 'to_seriesA'] = 'F2A_5'
```

```
final4_df.loc[final4_df['seriesA_to_seriesB_months'].between(0, 12, 'right'), 'seriesA_to_B'] = 'A2B_1'
final4_df.loc[final4_df['seriesA_to_seriesB_months'].between(12, 24, 'right'), 'seriesA_to_B'] = 'A2B_2'
final4_df.loc[final4_df['seriesA_to_seriesB_months'].between(24, 36, 'right'), 'seriesA_to_B'] = 'A2B_3'
final4_df.loc[final4_df['seriesA_to_seriesB_months'].between(36, 10000, 'right'), 'seriesA_to_B'] = 'A2B_4'
final4_df.loc[final4_df['seriesA_to_seriesB_months'].between(0, 0, 'both'), 'seriesA_to_B'] = 'A2B_5'
```

```
final4_df.loc[final4_df['seriesB_to_seriesC_months'].between(0, 12, 'right'), 'seriesB_to_C'] = 'B2C_1'
final4_df.loc[final4_df['seriesB_to_seriesC_months'].between(12, 24, 'right'), 'seriesB_to_C'] = 'B2C_2'
final4_df.loc[final4_df['seriesB_to_seriesC_months'].between(24, 36, 'right'), 'seriesB_to_C'] = 'B2C_3'
final4_df.loc[final4_df['seriesB_to_seriesC_months'].between(36, 10000, 'right'), 'seriesB_to_C'] = 'B2C_4'
final4_df.loc[final4_df['seriesB_to_seriesC_months'].between(0, 0, 'both'), 'seriesB_to_C'] = 'B2C_5'
```

Figure 17: New Categorical Features for Duration Between Investments

```
final5_df = final4_df.drop(columns = ['founded_on','seed_date', 'series_a_date','series_b_date',
                                      'series_c_date','name','founding_to_seed_months','founding_to_seriesA_months',
                                      'seriesA_to_seriesB_months','seriesB_to_seriesC_months'])
```

Figure 18: Dropping Irrelevant Features

```
def set_status(row):
    if (row["status"] == "acquired") | (row["status"] == "ipo") | (row["status"] == "operating") & (row["series_b"] == "1"):
        return "1"
    else:
        return "0"

final6_df = final5_df.assign(is_successfull=final5_df.apply(set_status, axis=1))
```

Figure 19: Defining Target Variable

```
ind_df = final6_df[final6_df['country_code']=='IND']
```

Figure 20: Dataframe for Indian Start-ups

```
fig = plt.subplots(figsize=(5,5))
sns.countplot(x="is_successfull",data=final6_df)
plt.title('Class Balance of Global Start-ups')
plt.show()
```

```
fig = plt.subplots(figsize=(5,5))
sns.countplot(x="is_successfull",data=ind_df)
plt.title('Class Balance of Indian Start-ups')
plt.show()
```

Figure 21: Class Balance for Global and Indian Start-ups

3.6 Data Transformation

All object datatypes are converted to category to support for efficient model building (fig.24). Following this step, the x and y variables are defined (fig. 25). Train test split is done for both dataframes that have global and Indian start-ups (fig. 26). Data scaling is done using the code in fig. 27. A separate dataframe is created to capture the results of all the models (fig. 28).

3.7 Modelling

Three balancing techniques are used wherein each technique has three machine learning models. The first balancing technique is with adjusting class weights to balanced (fig. 29) and this is employed in all three models which are Random Forest, Decision Tree and Logistic regression. The next technique employed is SMOTE (fig. 30). Again, all the three models are developed here. The model

```
fig = plt.subplots(figsize=(15,10))
ax = sns.countplot(final6_df['category'],order=final6_df['category'].value_counts()[:10].index,
                   palette = ['orange','red','lightcoral','pink','purple',
                              'lime','slategrey','khaki','cyan','violet'])
plt.xticks(rotation = 50)
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center', xytext = (0, 10), textcoords = 'offset points')
plt.title("Top 10 Industries of Start-ups In The World")
plt.show()
```

```
fig = plt.subplots(figsize=(15,10))
ax = sns.countplot(ind_df['category'],order=ind_df['category'].value_counts()[:10].index,
                   palette = ['lightcoral','red','pink','purple','violet',
                              'indigo','blue','cyan','lime','green'])
plt.xticks(rotation = 50)
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center', xytext = (0, 10), textcoords = 'offset points')
plt.title("Top 10 Industries of Indian Start-ups")
plt.show()
```

Figure 22: Top 10 Industries of Global and Indian Start-ups

```
fig = plt.subplots(figsize=(15,10))
ax = sns.countplot(final6_df['country_code'],order=final6_df['country_code'].value_counts()[:10].index)
plt.xticks(rotation = 50)
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center', xytext = (0, 10), textcoords = 'offset points')
plt.title("Top 10 Countries With Highest No. Of Startups")
plt.show()
```

Figure 23: Top 10 Countries with Most no. of Start-ups

with Logistic Regression can be seen in fig. 3. Finally, Random Under Sampling technique is used to balance the data (fig. 32). All three models are employed using this technique and the model with Decision Tree can be seen in fig. 33.

Similar approach is taken for Indian start-ups as well but here only two balancing techniques are employed which are adjusting class weights to balanced and SMOTE.

3.8 Evaluation

All the models are evaluated using state-of-the-art evaluation metrics in this domain and the results are tabulated (fig. 34).

```
## changing datatypes for model building

final6_df.seed = final6_df.seed.astype('category')
final6_df.series_a = final6_df.series_a.astype('category')
final6_df.series_b = final6_df.series_b.astype('category')
final6_df.series_c = final6_df.series_c.astype('category')
```

```
final6_df.seed_duration = final6_df.seed_duration.astype('category')
final6_df.to_seriesA = final6_df.to_seriesA.astype('category')
final6_df.seriesA_to_B = final6_df.seriesA_to_B.astype('category')
final6_df.seriesB_to_C = final6_df.seriesB_to_C.astype('category')
```

```
# converting categorical features to numerical features for model building

cat_df = final6_df.select_dtypes(include = ['object', 'category'])
l_encoder = LabelEncoder()
for column in cat_df:
    final6_df[column] = l_encoder.fit_transform(final6_df[column])
```

Figure 24: Data Transformation

```
x = final6_df.drop(columns = ['status', 'is_successfull'])
y = final6_df['is_successfull']
```

```
x1 = ind_df.drop(columns = ['status','is_successfull'])
y1 = ind_df['is_successfull']
```

Figure 25: Defining x and y Variables

```
x_train, x_test, y_train, y_test = train_test_split(x, y, shuffle = True, test_size = .2, random_state = 42 )
x_train1, x_test1, y_train1, y_test1 = train_test_split(x1, y1, shuffle = True, test_size = .2, random_state = 42 )
```

Figure 26: Data Split to Train and Test

```
scaler=StandardScaler()
x_train[['total_funding_usd', 'raised_amount_seed', 'raised_amount_seriesA',
        'raised_amount_seriesB', 'raised_amount_seriesC']]=scaler.fit_transform(x_train[['total_funding_usd',
                                                                                         'raised_amount_seed',
                                                                                        'raised_amount_seriesA',
                                                                                        'raised_amount_seriesB',
                                                                                        'raised_amount_seriesC']])
x_test[['total_funding_usd', 'raised_amount_seed', 'raised_amount_seriesA',
        'raised_amount_seriesB', 'raised_amount_seriesC']]=scaler.transform(x_test[['total_funding_usd',
                                                                                   'raised_amount_seed',
                                                                                      'raised_amount_seriesA',
                                                                                      'raised_amount_seriesB',
                                                                                      'raised_amount_seriesC']])
```

```
x_train1[['total_funding_usd', 'raised_amount_seed', 'raised_amount_seriesA',
        'raised_amount_seriesB', 'raised_amount_seriesC']]=scaler.fit_transform(x_train1[['total_funding_usd',
                                                                                          'raised_amount_seed',
                                                                                        'raised_amount_seriesA',
                                                                                        'raised_amount_seriesB',
                                                                                        'raised_amount_seriesC']])
x_test1[['total_funding_usd', 'raised_amount_seed', 'raised_amount_seriesA',
        'raised_amount_seriesB', 'raised_amount_seriesC']]=scaler.transform(x_test1[['total_funding_usd',
                                                                                    'raised_amount_seed',
                                                                                      'raised_amount_seriesA',
                                                                                      'raised_amount_seriesB',
                                                                                      'raised_amount_seriesC']])
```

Figure 27: Data Scaling

```
# Creating Dataframes to display evaluation of the models

models_summary = pd.DataFrame([],
                              columns=['Model Name',
                                       'F1 Score',
                                       'Accuracy',
                                       'Recall',
                                       'Precision'
                                      ])
```

```
models_summary1 = pd.DataFrame([],
                              columns=['Model Name',
                                       'F1 Score',
                                       'Accuracy',
                                       'Recall',
                                       'Precision'
                                      ])
```

Figure 28: Dataframe to Store Evaluation Results of Models

```
rf1 = RandomForestClassifier(class_weight = 'balanced', random_state = 8)
rf1.fit(x_train, y_train)

acc_rf1 = accuracy_score(y_test, rf1.predict(x_test))
conf_rf1 = confusion_matrix(y_test, rf1.predict(x_test))
report_rf1 = classification_report(y_test, rf1.predict(x_test))
F1_rf1 = f1_score(y_test, rf1.predict(x_test), average = 'weighted')
rec_rf1 = metrics.recall_score(y_test, rf1.predict(x_test))
pre_rf1 = metrics.precision_score(y_test, rf1.predict(x_test))

print(f"Accuracy Score: {acc_rf1}")
print(f'F1 score: {F1_rf1}')
print(f"Precision:  {pre_rf1}")
print(f"Recall:  {rec_rf1}")
print(f"Confusion Matrix : \n{conf_rf1}")
print(f"Classification Report : \n{report_rf1}")
```

Figure 29: Random Forest Model with Class Weights Balanced Technique

```
sm = SMOTE(random_state = 42)
x_train_resampled , y_train_resampled = sm.fit_resample(x_train , y_train)
```

Figure 30: Resampling Data Using SMOTE

```
lr2 = LogisticRegression(random_state = 8)
lr2.fit(x_train_resampled, y_train_resampled)

acc_lr2 = accuracy_score(y_test, lr2.predict(x_test))
conf_lr2 = confusion_matrix(y_test, lr2.predict(x_test))
report_lr2 = classification_report(y_test, lr2.predict(x_test))
F1_lr2 = f1_score(y_test, lr2.predict(x_test), average = 'weighted')
rec_lr2 = metrics.recall_score(y_test, lr2.predict(x_test))
pre_lr2 = metrics.precision_score(y_test, lr2.predict(x_test))

print(f"Accuracy Score: {acc_lr2}")
print(f'F1 score: {F1_lr2}')
print(f"Precision:  {pre_lr2}")
print(f"Recall:  {rec_lr2}")
print(f"Confusion Matrix : \n{conf_lr2}")
print(f"Classification Report : \n{report_lr2}")
```

Figure 31: Logistic Regression Using SMOTE

```
rus = RandomUnderSampler(random_state=42)
x_res, y_res = rus.fit_resample(x_train, y_train)
```

Figure 32: Resampling Data Using Random Under Sampling

```
dt3=DecisionTreeClassifier(random_state = 8)
dt3.fit(x_res, y_res)

acc_dt3 = accuracy_score(y_test, dt3.predict(x_test))
conf_dt3 = confusion_matrix(y_test, dt3.predict(x_test))
report_dt3 = classification_report(y_test, dt3.predict(x_test))
F1_dt3 = f1_score(y_test, dt3.predict(x_test), average = 'weighted')
rec_dt3 = metrics.recall_score(y_test, dt3.predict(x_test))
pre_dt3 = metrics.precision_score(y_test, dt3.predict(x_test))

print(f"Accuracy Score: {acc_dt3}")
print(f'F1 score: {F1_dt3}')
print(f'Precision: {pre_dt3}')
print(f'Recall: {rec_dt3}')
print(f"Confusion Matrix : \n{conf_dt3}")
print(f"Classification Report : \n{report_dt3}")
```

Figure 33: Decision Tree Using Random Under Sampling

```
models_summary = models_summary.append({
    'Model Name': 'E1 - Decision Tree with Random Under Sampling',
    'F1 Score': F1_dt3,
    'Accuracy' : acc_dt3,
    'Recall' : rec_dt3,
    'Precision' : pre_dt3
}, ignore_index=True)

models_summary.sort_values('F1 Score', ascending=False)
```

	Model Name	F1 Score	Accuracy	Recall	Precision
4	E1 - Logistic Regression with SMOTE	0.843365	0.853175	0.463938	0.647289
7	E1 - Logistic Regression with Random Under Sam...	0.840973	0.851338	0.453947	0.642414
1	E1 - Logistic Regression with Weighted Balance	0.835999	0.843878	0.468811	0.603797
0	E1 - Random Forest with Weighted Balance	0.833835	0.837324	0.511209	0.570264
3	E1 - Random Forest with SMOTE	0.812729	0.804240	0.608431	0.479593
2	E1 - Decision Tree with Weighted Balance	0.788988	0.781338	0.516813	0.427664
6	E1 - Random Forest with Random Under Sampling	0.780767	0.761519	0.672758	0.413540
5	E1 - Decision Tree with SMOTE	0.775315	0.759342	0.586623	0.400083
8	E1 - Decision Tree with Random Under Sampling	0.725632	0.694807	0.666058	0.337782

Figure 34: Results for Models Built for Start-ups Around the World