Startup Ecosystem and Success in the Global South:
Determinants, Machine Learning Insights, and Policy Implications

An extended research project report submitted to the University of Manchester
for the degree of Data Science (Social Analytics) in the Faculty of Humanities

Technical Appendix

ID: 11513161

School of Social Sciences

Description

This appendix briefly summarises the process included in the titled study to obtain the outcomes presented in the study, on data collection, preprocessing, exploratory data analysis, model development and performance evaluation process. It outlines the detailed steps to perform each process, including baseline settings, package information, and parameter settings, to ensure the reproducibility of the study.

Contents

Illustrations

Appendix A: Data Collection

This section highlights the process and details in the original data collection phase, before subsequent steps, including preprocessing and model development.

Data source and original datasets

The data employed were collected from the following two sources: *Orbis,* one of the most comprehensive firm databases covering around 4.3 million companies globally, for institutional features, and *World Bank Open Data* and *World Bank Data Bank,* open database providing global development data such as economics, environmental, health, infrastructure, or governance, for macroeconomic features. Upon data collection, the study focuses on 26 low-income, 51 lower-middle-income, and 55 upper-middle-income countries as defined by the World Bank (2025), totalling 132 countries in the 2025 fiscal year, as listed in Table A-1.

Table A-1: Full list of countries collected by World Bank income category

World Bank Income Category	Countries	Count
LOW-INCOME ECONOMIES ($1,145 OR LESS)	Afghanistan, Burkina Faso, Burundi, Central African Republic, Chad, Congo, Dem. Rep., Eritrea, Ethiopia, Gambia, The, Guinea-Bissau, Korea, Dem. People's Rep., Liberia, Madagascar, Malawi, Mali, Mozambique, Niger, Rwanda, Sierra Leone, Somalia, South Sudan, Sudan, Syrian Arab Republic, Togo, Uganda, Yemen, Rep.	26
LOWER-MIDDLE INCOME ECONOMIES ($1,146 TO $4,515)	Angola, Bangladesh, Benin, Bhutan, Bolivia, Cabo Verde, Cambodia, Cameroon, Comoros, Congo, Rep., Côte d'Ivoire, Djibouti, Egypt, Arab Rep., Eswatini, Ghana, Guinea, Haiti, Honduras, India, Jordan, Kenya, Kiribati, Kyrgyz Republic, Lao PDR, Lebanon, Lesotho, Mauritania, Micronesia, Fed. Sts., Morocco, Myanmar, Nepal, Nicaragua, Nigeria, Pakistan, Papua New Guinea, Philippines, Samoa, São Tomé and Principe, Senegal, Solomon Islands, Sri Lanka, Tajikistan, Tanzania, Timor-Leste, Tunisia, Uzbekistan, Vanuatu, Vietnam, West Bank and Gaza, Zambia, Zimbabwe	51
UPPER-MIDDLE-INCOME ECONOMIES ($4,516 TO $14,005)	Albania, Algeria, Argentina, Armenia, Azerbaijan, Belarus, Belize, Bosnia and Herzegovina, Botswana, Brazil, China, Colombia, Costa Rica, Cuba, Dominica, Dominican Republic, Ecuador, El Salvador, Equatorial Guinea, Fiji, Gabon, Georgia, Grenada, Guatemala, Indonesia, Iran, Islamic Rep., Iraq, Jamaica, Kazakhstan, Kosovo, Libya, Malaysia, Maldives, Marshall Islands, Mauritius, Mexico, Moldova, Mongolia, Montenegro, Namibia, North Macedonia, Paraguay, Peru, Serbia, South Africa, St. Lucia, St. Vincent and the Grenadines, Suriname, Thailand, Tonga, Türkiye, Turkmenistan, Tuvalu, Ukraine, Venezuela, RB	55

Note: Although Venezuela, RB is unclassified due to the unavailability for the 2025 fiscal year, the study classifies it as an upper-middle income country based on the actual value until FY21, due to the availability of the data employed explicitly for this study.

Notes: Full list of 132 countries data collected for the study by three World Bank income categories in the 2025 fiscal year. Twenty-six countries account for LOW-INCOME ECONOMIES ($1,145 OR LESS), while 51 are for LOWER-MIDDLE INCOME ECONOMIES ($1,146 TO $4,515), and 55 are for UPPER-MIDDLE INCOME ECONOMIES ($4,516 TO $14,005). This study only employs countries from LOWER-MIDDLE and UPPER-MIDDLE income countries, excluding the LOW-INCOME economies, due to data governance vulnerabilities in those countries.

A data filter (Table A-2) is applied upon data collection from *Orbis*, targeting firms under 5 years old (Calvino *et al.* 2025), both active and inactive, to avoid survival bias. Firms without *Total Assets* data are excluded following International Finance Corporation (n.d.) to ensure data quality. The sample is limited to companies with 5–50 employees, aligning with SME definitions by IFC, ILO (Kok and Berrios 2019), and OECD (n.d.).

Table A-2: Data collection filter

Column	Filter Value
Year of incorporation	*from 2019*
World region/Country/Region in country	*The defined countries in Table A-1*
Status	*Active companies, Inactive companies*
Total assets	*All companies with a known value, Last available year, excluding companies with no recent financial data and Public authorities/States/Governments*
Number of employees	*min=5, max=50, Last available year, exclusion of companies with no recent financial data and Public authorities/States/Governments*

Notes: Features are applied to the filter during data collection to ensure the study's objectives are met and data quality is maintained. *Year of incorporation* is filtered to be later than 2019, to obtain firms under 5 years old explicitly. On *World region/Country/Region in country,* the countries defined in Table A-1, which correspond to the low-income, lower-middle-income, and upper-middle-income countries based on the World Bank income category (World Bank 2025), are matched with countries only in the Global South. *Status* is of both active and inactive companies to correctly capture the dynamics of both the success and failure of startups. *Total assets*, and *Number of employees* are filtered based on the justification obtained from past literature (Kok and Berrios 2019; OECD n.d.).

Table A-3 and A-4 represent the features collected from *Orbis* and *World Bank* data sources, respectively, which are then to be preprocessed in the subsequent steps. Overall, the dataset maintains a well-balanced composition of categorical and numerical features. Each feature in Figure A-4 is associated with a unique code designated by either the *World Bank Open Data* or *DataBank*. The variables span a range of macroeconomic, demographic, and environmental indicators.

Table A-3: Full list of data from Orbis

Variable Name	Type	Description	Potential Categories (Examples)
duo_bvd_id	Categorical	Unique identifier for each company in the Orbis database.	*e.g.* "VN123456", "BR987654"
city	Categorical	The city where the company is located.	*e.g.* “HA NOI”, “BARUERI”, “HANGZHOU “
country_isocode	Categorical	The ISO 3166-1 alpha-2 or alpha-3 country code where the company is located.	*e.g.* “VN”, “BR”, “CN”
region	Categorical	The broader geographical region where the company is located.	*e.g.* “Red River Delta”, “Sao Paulo”, “East China”
legal_form	Categorical	The legal structure or type of the company.	*e.g.* “Private limited companies”, “Public limited companies”, “Sole traders/proprietorship”
accounting_template	Categorical	The specific accounting template used by the company for financial reporting.	*e.g.* “IND”
dm_gender	Categorical	The gender of the primary decision-maker or director (*e.g.* Male, Female, Unknown).	*e.g.* “Male”, “Female”, “Unknown”
wb_category	Categorical	A categorisation based on World Bank classifications by income level.	*e.g.* “UPPER-MIDDLE-INCOME”, “LOWER-MIDDLE-INCOME”
main_activity	Categorical	The primary industry or business sector of the company.	*e.g.* “Manufacturing”, “Services”
description	Categorical	A textual business description of the company.	*e.g.* “Engages in software development and consulting.”
website	Categorical	Company website URL.	*e.g.* “www.example.com”
net_income_usd	Numeric	Net income of the company, expressed in US dollars.	NA
total_assets_usd	Numeric	Total assets held by the company, expressed in US dollars.	NA
shareholders_funds_usd	Numeric	Funds belonging to the shareholders of the company, in US dollars.	NA
capital_usd	Numeric	The capital of the company, expressed in US dollars.	NA
revenue_usd	Numeric	Total revenue generated by the company, in US dollars.	NA
cash_flow_usd	Numeric	Operating cash flow of the company, in US dollars.	NA
tax_usd	Numeric	Taxes paid by the company, in US dollars.	NA
dm_age	Numeric	Age of the decision-maker or director (typically in years).	NA
age	Numeric	Age of the company.	NA

Notes: Figure lists all the institutional variables and their metadata, with the majority of categoricals and a smaller portion of numerics. Overall, all types of features, categoricals and numerics, are employed in a balanced tone.

Table A-4: Full list of data from World Bank

Feature	World Bank ID/Code	Source
Learning-Adjusted Years of School (2020)	WB_HCI_LAYS	Open Data
Urban population growth (annual %) (2022)	WB_WDI_SP_URB_GROW	Open Data
Logistics Performance Index (2018)	WB_LPI	Open Data
Life expectancy at birth, total (years) (2022)	WB_WDI_SP_DYN_LE00_IN	Open Data
Female labour force participation, % (2023)	WEF_TTDI_FEMLABOR	Open Data
Government expenditure on education as a percentage of GDP (%) (2022)	UIS_EDSTATS_XGDP_FSGOV	Open Data
GDP per capita (constant 2015 US$) (2023)	WB_WDI_NY_GDP_PCAP_KD	Open Data
Use of digital payments, % pop 15+ (2022)	WEF_TTDI_DIGITALPAY	Open Data
Agriculture, value added to GDP (2022)	FAO_AS_4548	Open Data
Ease of doing business score (2019)	IC.BUS.EASE.DFRN.XQ.DB1719	DataBank
Rule of law index (2023)	RL.EST	DataBank
Access to electricity (% of population) (2023)	EG.ELC.ACCS.ZS	DataBank
Control of Corruption: Estimate (2023)	CC.EST	DataBank
Domestic credit to private sector (% of GDP) (2021)	FS.AST.PRVT.GD.ZS	DataBank
Individuals using the Internet (% of population) (2023)	IT.NET.USER.ZS	DataBank
CPIA gender equality rating (1=low to 6=high) (2023)	IQ.CPA.GNDR.XQ	DataBank

Notes: List of all the features obtained from the World Bank, each of which is assigned a unique identifier or code defined by the World Bank. They come from either World Bank Open Data or DataBank. The data includes macroeconomic, demographic, or environmental indicators.

Appendix B: Data Preprocessing

This section summarises the preprocessing process performed on the collected original datasets prior to the model development.

Packages

Table B-1 lists the packages used for the analysis, along with the corresponding versions. Data processing is primarily handled using Pandas, while computational tasks and data transformations are performed using scikit-learn and statsmodels. Additional libraries such as numpy, datetime, and pycountry are utilised for specific tasks as outlined. Duplicate packages are not listed multiple times below.

Table B-1: Package list employed in data preprocessing

Library Name	Version	Module/Function Name	Purpose
pandas	2.3.0	pd.read_excel()	Read data from an Excel file into a DataFrame.
		pd.read_csv()	Read data from a CSV file into a DataFrame.
		pd.to_csv()	Write a DataFrame to a CSV file.
		pd.concat()	Concatenate pandas objects along a particular axis.
		pd.merge()	Merge DataFrames by columns or indexes.
		pd.to_numeric()	Convert the argument to a numeric type.
		pd.DataFrame()	Create a new DataFrame.
		pd.get_dummies()	Convert categorical variables into dummy/indicator variables.
		.isna()	Detect missing values.
		.drop()	Remove rows or columns.
		.copy()	Make a deep copy of the object.
		.duplicated()	Return a Boolean Series denoting duplicate rows.
		.sum()	Return the sum of values.

Table B-1 (continued)

		.apply()	Apply a function along an axis of the DataFrame.
		.replace()	Replace values given in a dictionary, Series, or regex.
		.groupby()	Group DataFrame using a mapper or by a Series of columns.
		.transform()	Call a function on each group in a GroupBy object.
		.loc	Access a group of rows and columns by labels.
		.map()	Map values of Series according to an input mapping.
		.notna()	Detect existing (non-missing) values.
		.astype(int)	Cast a pandas object to a specified dtype.
		.value_counts()	Return a Series containing counts of unique values.
		.fillna()	Fill NA/NaN values using the specified method.
		.columns	Return the column labels of the DataFrame.
		.dropna()	Remove missing values.
		.reset_index()	Reset the index of the DataFrame.
		.strip()	Remove leading and trailing characters (typically whitespace).
		.sort_index()	Sort object by labels (along an axis).
		.dtypes	Return the dtypes in the DataFrame.
		.append()	Append rows of another DataFrame.
		.skew()	Return unbiased skewness over the requested axis.
pycountry	24.6.1	pycountry.countries.get(alpha_3=alpha3).alpha_2	Get the ISO alpha-2 code for a given alpha-3 code.
geonamescache	2.0.0	gc.get_countries().values()	Get a list of country values from geonamescache.
scikit-learn	1.3.2	.fit_transform()	Fit the data, then transform it.
scikit-learn		svd.fit_transform()	Fit and apply SVD dimensionality reduction to the data.

Table B-1 (continued)

scikit-learn.impute		IterativeImputer()	Multivariate feature imputer using iterative modelling.
scikit-learn.linear_model		BayesianRidge()	Bayesian ridge regression model.
scikit-learn.preprocessing		StandardScaler()	Standardise features by removing the mean and scaling to unit variance.
scikit-learn.preprocessing		LabelEncoder()	Encode labels with a value between 0 and n_classes-1.
scikit-learn.decomposition		TruncatedSVD()	Dimensionality reduction using truncated SVD.
datetime	2.9.0	datetime.now().year	Get the current year.
statsmodels.stats.outliers_influence	0.14.5	variance_inflation_factor()	Compute the VIF for each feature.
statsmodels.api		sm.add_constant()	Add a constant column to the input data.
numpy	1.24.3	np.percentile()	Compute the q-th percentile of the data.
		np.log1p()	Compute log(1 + x) for each element.
scipy.stats.mstats	1.15.3	winsorize()	Limit extreme values in data.

Notes: List of packages employed in data preprocessing. Pandas is mainly used for processing the data frames, while scikit-learn and statsmodels are used for performing computations or transformations on the data frames. Other packages, such as numpy, datetime, and pycountry, are used for different specific purposes described above.

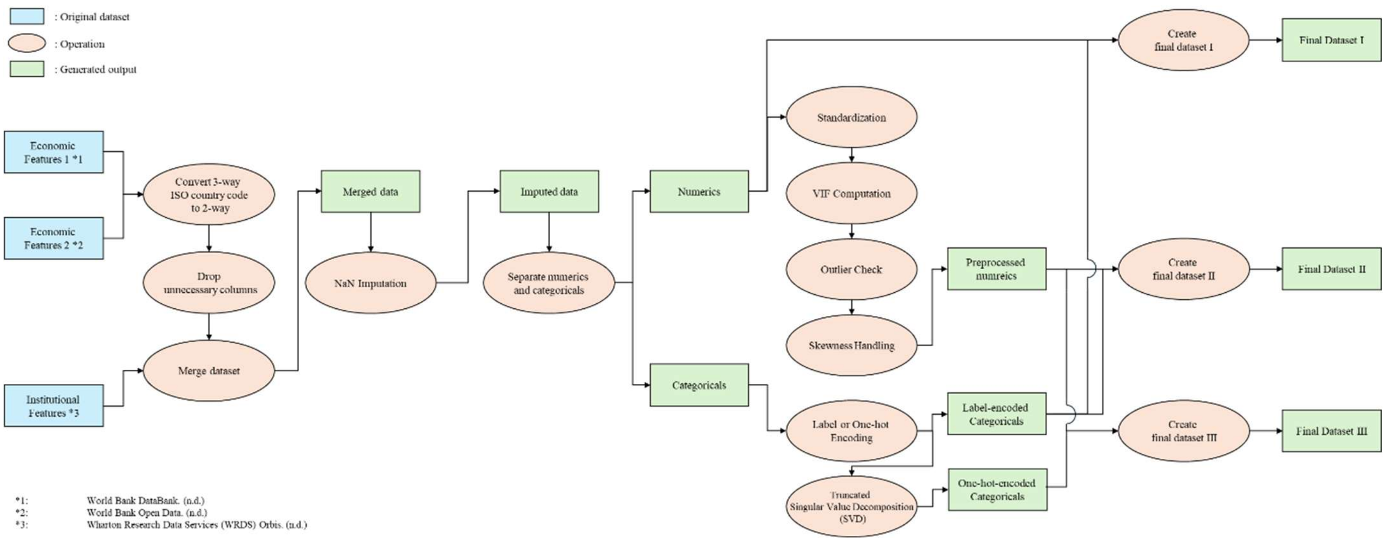
Preprocessing pipelines

This section provides the full details and preprocessing performed in the analysis, along with the preprocessing pipelines, shown in Figure B-1. Preprocessing is performed with methods defined by García *et al.* (2015), with pipelines developed for this analysis. First, institutional and macroeconomic datasets are merged and separated into numerics and categoricals after NaN imputation, using different imputation strategies, including regional/income-based means for numeric features, Expectation-Maximisation (EM) for institutional variables, and type-specific rules for categoricals (*e.g.* country-mode for location). Then numerics go through a series of processes, including standardisation, VIF computation, outlier, skewness handling, log-transforming highly skewed variables and winsorising (Blaine 2018) the still high-skewed ones to address non-normality (Osborne 2010), shown in Table B-4. Categoricals are converted by either label or one-hot encoding (Dahouda and Joe 2021); the former for tree-based and DL models[1], and the latter (with truncated SVD (Hansen 1987)) for Logit,

[1] For low-overhead computation.

SVM, kNN, and Naïve Bayes. Finally, the study prepares three model-specific datasets: (1) raw numerics with label-encoded categoricals for tree-based models, (2) processed numerics with one-hot encoding (SVD-applied) for Logit, SVM, kNN, and Naïve Bayes, and (3) processed numerics with label encoding for DL models, based on prior studies (Pargent *et al.* 2022; Guo and Berkhahn 2016; De Amorim *et al.* 2023), with splitting the data into training, validation, and test sets by "Three-Way Holdout Method" (Raschka 2020), where an unseen test set is used to evaluate generalisability and avoid overfitting (Xu and Goodacre 2018).

Figure B-1: Data preprocessing pipelines



Notes: Institutional and economic environmental data are merged first and separated into numerics and categoricals after NaN imputation. Then numerics go through a series of processes, including standardisation, VIF computation, outlier check, and skewness handling, while either label or one-hot encoding converts categoricals. Finally, the final three datasets are created for appropriate model types, combining different data frames.

Merging Data

The two types of original datasets, institutional features obtained from *Orbis* and economic features from World Bank data sources, are merged on the ISO country codes, after converting the 3-letter code from World Bank into a 2-letter code to fit *Orbis* by using pycountry (2024). As this library does not provide a code for Kosovo, it has been encoded manually.

NaN Imputation

Table B1 shows the column names and the number of tuples containing NaN in the merged dataset, especially for the institutional features, out of 539,487 tuples in total. This process has not imputed *Legal_form*, as it includes an ignorable portion of NaN values. Features with high NaN ratios, namely *main_activity*, *cash_flow_usd*, *tax_usd,* and *dm_age,* that are not to be flagged as "missing" in the subsequent process, are excluded from the analysis.

Table B-2: List of institutional features with NaN ratio

Column	Count	NaN Ratio (%)
city	1287	0.24
region	7069	1.31
legal_form	4	0.00
main_activity	525683	97.44
description	356828	66.14
net_income_usd	31991	5.93
cash_flow_usd	527991	97.87
shareholders_funds_usd	8659	1.61
accounting_template	47616	8.83
tax_usd	539416	99.99
capital_usd	322236	59.73
duo_bvd_id	515289	95.51
dm_gender	306703	56.85
dm_age	530388	98.31
revenue_usd	42399	7.86
website	493999	91.57

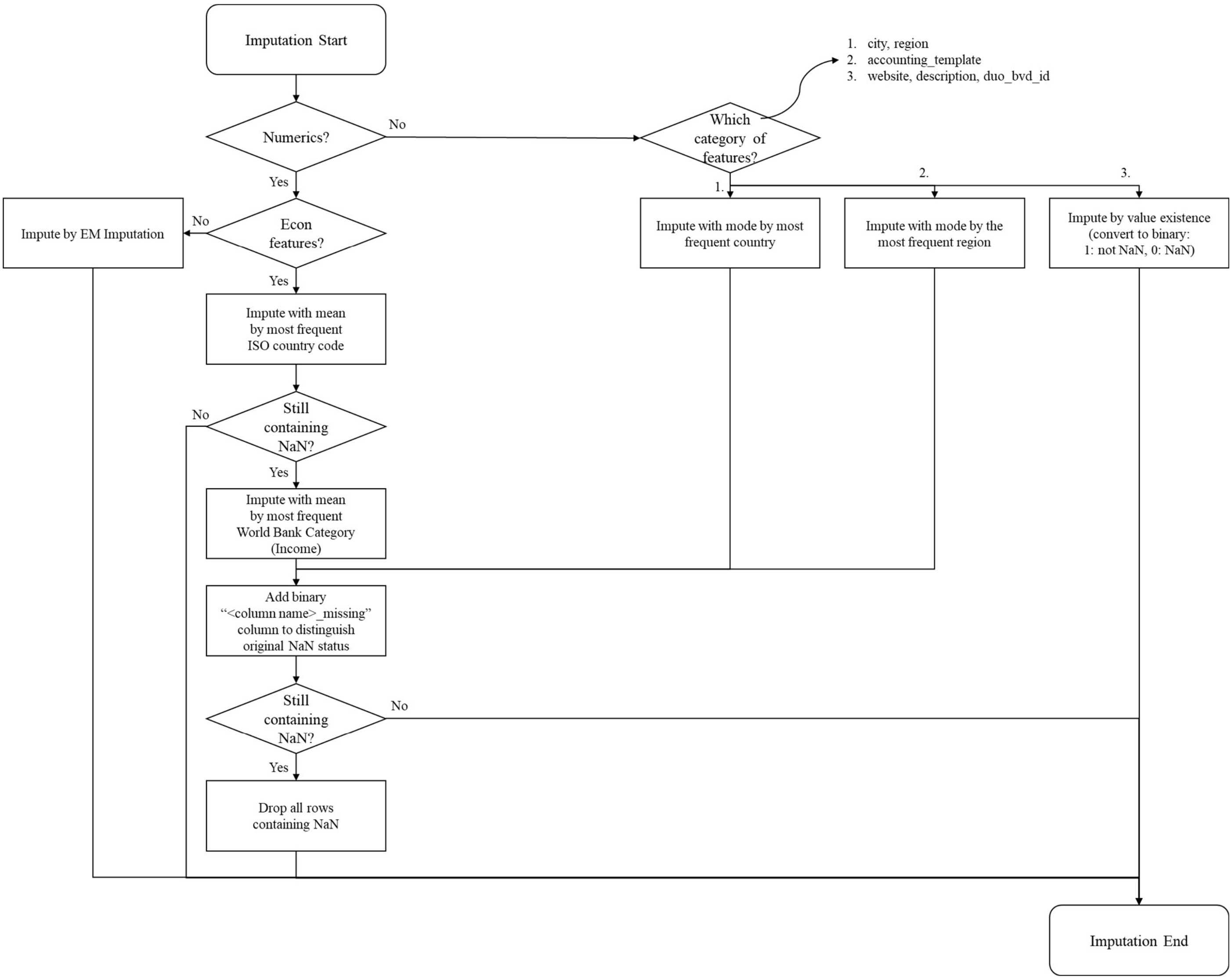
Notes: The comprehensive list of features of the merged dataset focuses on the institutional features, with a NaN ratio for each variable. *Legal_form* is omitted from the list of features in the imputation procedure. *Main_activity*, *cash_flow_usd*, *tax_usd*, and *dm_age* are excluded from the analysis due to the substantial NaN ratio, which is difficult to impute and maintain statistical significance. Other variables have a comparatively mild NaN ratio; thus, they are imputed through the defined pipeline.

The study uses a pipeline (Figure B-2) to perform imputation on this dataset, using different strategies for numerics and categoricals, with several different operations on the categoricals. For numerics, economic environmental features are imputed by mean of the most frequent value of World Bank region category, followed by the income category defined by World Bank (2025), adopting the same methodology by

HRH2030 Consortium (2019), since such characteristics are often captured by geographic and then income classifications, for which the World Bank provides representative values. For institutional features, Expectation-Maximisation (EM) Imputation is adopted as recommended by García *et al.* (2015), an iterative approach that alternates between estimating the expected complete-data log-likelihood (E-step) and maximising it (M-step) to handle missing or hidden data effectively.

The categoricals are imputed by three different strategies based on the types of features. *City* and *region* are imputed with mode by the most frequent ISO country code, and *accounting_template* is imputed by regions as the sectoral and regulatory environment is considered to segregate by locations (World Bank 2020a). *Website*, *description,* and *duo_bvd_id* are converted into binary to show availability, since the data unavailability itself is assumed to be statistically significant, especially when using a public dataset (Ross *et al.* 2021). Several remaining tuples containing NaN (*region*: 329, *legal_form*: 4, *accounting_template*: 457) after performing this process are thus dropped for their portions against the total number of tuples.

Figure B-2: NaN imputation pipeline



Notes: The imputation pipeline applies distinct strategies to numeric and categorical variables. For numeric features, economic indicators are imputed using the mean of the most frequent World Bank region, followed by income group averages (World Bank 2025), replicating the approach of HRH2030 Consortium (2019), based on the assumption that such values are geographically and economically determined. Institutional indicators are imputed via Expectation-Maximisation (EM), as suggested by García et al. (2015). For categorical features, *city* and *region* are filled using mode by ISO country code, while *accounting_template* is imputed by region, considering regulatory variance in the same regions (World Bank 2020a). Text-related fields such as *website*, *description*, and *duo_bvd_id* are binarised to reflect their presence, assuming informative missingness (Ross et al. 2021). Remaining missing tuples (*region*: 329, *legal_form*: 4, *accounting_template*: 457) are dropped due to their relatively low counts.

Standardisation

All the numeric features in the data are standardised by Z-score normalisation with the formula below, considering the number of outliers globally existing in most of the features (García *et al.* 2015):

$$z = \frac{(x - \mu)}{\sigma} \tag{1}$$

where:

μ : Mean of the feature calculated from the data

σ : Standard deviation of the feature calculated from the data

Standardisation of data is crucial as the scale difference of features can affect the analysis result obtained from machine learning models (García *et al.* 2015).

VIF Computation

According to O'brien (2007), Variance Inflation Factor (VIF) is used to measure the existence and extent of multicollinearity among independent variables when conducting regressions, computed by the following equation:

$$VIF_j = \frac{1}{\left(1 - R_j^2\right)} \tag{2}$$

where:

R_j^2 : quantifies the extent to which the variance of the ith independent variable is explained by the other independent variables within the model.

Table B-3 shows the VIFs of all the numeric variables in the data, and based on the "*Rules of thumb*" of less-than-10 tolerance (O'brien 2007), suggests that macroeconomic features exhibited extremely high VIF scores, which is expected, as these features are constant across multiple startups belonging to the same country. This study does not actively remove features with high VIFs but relies on less aggressive methods like LASSO or ridge regression. Since "*rules of thumb*" should be interpreted contextually, variance inflation is less concerning when statistically significant results or narrow confidence intervals are still achieved. It becomes critical only when it likely undermines such outcomes.

Table B-3: VIF summary

Feature Name	VIF
gdp_per_capita_2023	19348.65
private_credit_2021	19024.14
life_expectancy_2022	10314.97
urban_pop_growth_2022	4325.52
logistics_index_2018	3386.88
electricity_access_2023	2725.77
gender_equality_rating_2023	2014.47
edu_expenditure_gdp_2022	1876.95
agri_value_gdp_2022	1734.86
female_labor_participation_2023	1315.21
rule_of_law_2023	1247.61
schooling_years_2020	1054.69
corruption_control_2023	846.34
digital_payment_2022	500.84
internet_users_2023	328.57
doing_business_score_2019	200.12
capital_usd	2.20
shareholders_funds_usd	2.08
age	1.96
total_assets_usd	1.55

Notes: Variance Inflation Factors (VIFs) for all numeric variables. Following the common "*rule of thumb*" threshold of 10 (O'Brien 2007), macroeconomic indicators show notably high VIFs, which is an expected result, as features are constant across startups within the same country. Rather than removing these variables, this study adopts less aggressive approaches like LASSO regression. High VIFs are interpreted in context; they are less problematic when statistical significance or narrow confidence intervals are still obtained, and only impose concern when they compromise inference reliability.

Outlier Handling

By employing Tukey's Fences based on Interquartile Range (IQR) criteria (Tukey 1977), outliers in numeric features are detected:

$$\neg\big((x < Q1 - 1.5 \times IQR) \vee (x > Q3 + 1.5 \times IQR)\big) \forall x \; in \; the \; row$$

(3)

where:

x : A feature in the row

The number of tuples containing at least one outlier for any feature is 202,470 out of 477,077, which is almost 42.43% of the total data. Based on the findings by García *et al.* (2015), however, it is justified not to remove outliers for this study, as most of the models assumed to yield higher performance (Regression, Deep Learning, and SVM) are considered robust against outliers, including Random Forest (Breiman 2001).

Skewness Handling

Table B-4 lists the complete list of skewness of features, five of which are log-transformed, and two of which are then winsorised based on the size of skewness.

Table B-4: Skewness of full numerics

Variable	Value before logging	Value after logging	Value after Winsorisation
total_assets_usd	139.19	21.86	6.36
capital_usd	-467.58	34.03	5.44
shareholders_funds_usd	-471.04	-5.56	NA
revenue_usd	97.10	-0.24	NA
net_income_usd	-335.33	-0.74	NA
schooling_years_2020	0.05	NA	NA
urban_pop_growth_2022	-0.05	NA	NA
logistics_index_2018	-1.08	NA	NA
life_expectancy_2022	-0.96	NA	NA
female_labor_participation_2023	-0.79	NA	NA
edu_expenditure_gdp_2022	0.95	NA	NA
gdp_per_capita_2023	-0.16	NA	NA
digital_payment_2022	-2.01	NA	NA
agri_value_gdp_2022	0.44	NA	NA
doing_business_score_2019	-0.30	NA	NA
rule_of_law_2023	-3.59	NA	NA
electricity_access_2023	-3.56	NA	NA
corruption_control_2023	-0.36	NA	NA
private_credit_2021	-1.23	NA	NA
internet_users_2023	-0.36	NA	NA
gender_equality_rating_2023	3.59	NA	NA
age	-1.08	NA	NA

Notes: The list of numerics with skewness values. *Total_assets_usd* and *revenue_usd* exhibit a significant degree of positive skewness, whereas others (*capital_usd*, *shareholders_funds_usd*, and *net_income_usd*) display negative skewness. All listed features are log-transformed to mitigate the non-normality, and the top two of them are further winsorised. After log-transformation and winsorisation, the skewness values of those variables are set in an acceptable range. The other variables have mild skewness values compared to those that undergo transformation.

Encoding

All categorical features are transformed into dummy variables using two main approaches: converting nominal values into numeric representations, or creating binary indicators for each category (García *et al.* 2015), which are named as label and one-hot encoding (Dahouda and Joe 2021), respectively. Those two converted data frames are used for different types of models (label-encoded for XGBoost, Random Forest, and DL, and one-hot-encoded for Naïve Bayes, kNN, OLS, and SVM). One-hot encoding generally performs better than label encoding for XGBoost and Random Forest (Pargent *et al.* 2022); however, this study employs label encoding due to limited computing resources. It is also adopted for deep learning with entity embedding, as it is more computationally efficient than one-hot encoding (Guo and Berkhahn 2016).

One-hot-encoded variables are then performed the truncated Singular Value Decomposition (SVD) (Hansen 1987), a method to explicitly "cut off" unstable factors corresponding to small singular values in the dataset by decomposing the data A onto three matrices:

$$A = U\,\Sigma V^{\{\top\}} \tag{4}$$

where:

$A \in \mathbb{R}^{\{m \times n\}}$: Original data (as a matrix)

$U \in \mathbb{R}^{\{m \times m\}}$: Left singular (orthogonal) matrix

$\Sigma \in \mathbb{R}^{\{m \times n\}}$: Diagonal matrix of (non-negative) singular values

$V \in \mathbb{R}^{\{n \times n\}}$: Right singular (orthogonal) matrix

This method is used for dimensionality reduction to mitigate the "*curse of dimensionality*," referring to the fact that the sample size needed for statistical model training grows exponentially with the input data's dimensionality (Bellman and Kalaba 1959). With one-hot encoding, every new category level adds an input dimension, rapidly leading to a highly sparse and high-dimensional input, which significantly increases the computational workload.

Target Variable

The target variable *success* is defined as 1 (n = 86,668) if a firm has positive net income and revenue over 455,230 USD (threshold below the top quartile), and 0 otherwise (n = 390,252). While investment-based (*e.g.*

acquisition, IPO) and growth-based (*e.g.* revenue or employee growth) indicators are commonly used (*e.g.* Arroyo *et al.* 2019; Meressa 2020), this study adopts a combined revenue-profitability measure due to the lack of longitudinal data in the Global South (Independent Expert Advisory Group on a Data Revolution for Sustainable Development 2014; Kar 2023), aligning with benchmarks like the "Rule of 40" (Feld 2015), which combines growth and profit to effectively assess firm success (Lee 2024). Steps to generate the target are implemented in and can be reproduced using the notebooks (*ann.ipynb*, *cnn.ipynb*, *lstm.ipynb*, *rf.ipynb*, *xgb.ipynb*, *knn.ipynb*, *naive-bayes.ipynb*, *svm.ipynb*, *linear-logit.ipynb*, and *poly-logit.ipynb*) in the GitHub repository for the study (kokosan123 2025).

Appendix C: Exploratory Data Analysis

This section outlines the tools and processes used in the exploratory data analysis to understand the dataset characteristics before the model development.

Packages

Table C-1 lists the packages used for the exploratory data analysis, along with the corresponding versions. Visualisations, including both univariate and multivariate analyses, are generated using matplotlib.pyplot and seaborn, while pandas is utilised for data frame manipulation and processing. Additional libraries such as numpy and statistics are employed to compute specific statistical measures within the dataset.

Table C-1: Package list employed in exploratory data analysis

Library Name	Version	Module/Function Name	Purpose
matplotlib.pyplot	3.10.3	plt.subplots	Generate graph frames
		plt.title(), plt.xlabel(), plt.ylabel()	Set the plot title and axis labels
		plt.grid(True)	Add grid lines for visual clarity
		plt.tight_layout()	Adjust the layout to avoid the overlap of plot elements
		plt.show()	Display the visualisations
		plt.figure(figsize=(12, 8))	Set the overall figure size for better readability
		plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')	Move the legend outside the main plot area for improved layout
		plt.savefig('...')	Save the visualisation as a PNG file for later reference or inclusion in reports
seaborn	0.13.2	sns.kdeplot	Create KDE plots
		sns.histplot	Create histograms
		sns.boxplot	Create box plots
		sns.scatterplot()	Plot scatter plots between numerical variable pairs
		sns.regplot()	Add a global regression line with a 95% confidence interval without overlapping scatter points
pandas	2.3.0	df.select_dtypes()	Extract numerical columns
		Series.value_counts()	Calculate frequencies of categorical values
		Series.plot.pie()	Draw pie charts
		.nunique()	Count the number of unique categories
		.mode()	Get the most frequent category
		.value_counts()	Get category frequencies
		.isnull().sum()	Count the number of missing values
		pd.DataFrame()	Store the results as a DataFrame
numpy	1.24.3	np.mean()	Calculate the arithmetic mean (average) of a numeric array or series
		np.quantile()	Return the specified quantile of a dataset.
statistics	1.15.3	statistics.stdev()	Compute the sample standard deviation of a data series

Notes: The complete list of the packages employed in exploratory data analysis. Matplotlib.pyplot, and seaborn are employed to create visualisations, including univariate and multivariate data analysis, while pandas is used to perform processing on the data frames. Other packages, such as numpy and statistics, are used to calculate specific statistics of the data.

Summary Statistics

The study creates summary statistics on both numeric and categorical variables, calculating mean, standard deviation, min, max, and quartiles for each numeric variable in the DataFrame. For categorical variables, it creates a summary statistics table that includes the number of unique categories, the mode category, its frequency, and the number of missing values, along with their proportion. All steps are implemented in and can be reproduced using the notebook *preprocessing.ipynb* in the GitHub repository for the study (kokosan123 2025).

Univariate Data Analysis

The study performs univariate data analysis on both numeric and categorical variables. For numeric variables, it visualises each numerical column in the DataFrame using (1) a KDE plot (for distribution analysis), (2) a Histogram (for frequency analysis), and (3) a Boxplot (for outlier detection). For categorical variables, it visualises each categorical column in the DataFrame using: (1) a Bar plot (to see category distribution) and (2) a Pie chart (to see proportion of each category). All steps are implemented in and can be reproduced using the notebook *preprocessing.ipynb* in the GitHub repository for the study (kokosan123 2025).

Multivariate Data Analysis

The study performs multivariate data analysis by generating a scatter plot of countries, with each point colour-coded by country name, to visualise the relationship between two country-level macroeconomic variables, along with a regression line across all countries to show the general trend between the two variables. All steps are implemented in and can be reproduced using the notebook *additional-figures.ipynb* in the GitHub repository for the study (kokosan123 2025).

Appendix D: Model Development

This section outlines the details of the model development process performed in the study, including the employed packages, hyperparameter tuning and hardware environment details. All the subsequent steps are implemented in and can be reproduced using the notebooks (*ann.ipynb*, *cnn.ipynb*, *lstm.ipynb*, *rf.ipynb*, *xgb.ipynb*, *knn.ipynb*, *naive-bayes.ipynb*, *svm.ipynb*, *linear-logit.ipynb*, and *poly-logit.ipynb*) in the GitHub repository for the study (kokosan123 2025).

Packages

Table D-1 lists the packages employed for model development, along with the corresponding versions. Deep learning models are implemented using PyTorch, while tree-based models (*viz.* Random Forest and XGBoost) are developed using RandomForestClassifier from scikit-learn.ensemble and XGBClassifier from xgboost, respectively. Other models, including k-Nearest Neighbours, Support Vector Machines, Naïve Bayes, and Logistic Regression, are built using scikit-learn. Hyperparameter tuning for all models is performed with Optuna.

Table D-1: Package list employed for developing models

Library Name	Version	Module/Function Name	Purpose
torch	2.7.1	torch.tensor()	Create a tensor from data.
		torch.cat()	Concatenate tensors along a specified dimension.
		loss.backward()	Backpropagate the loss.
		torch.argmax(outputs, dim=1)	Return the indices of the maximum values along an axis.
		torch.softmax(outputs, dim=1)	Compute softmax over the specified dimension.
		torch.max	Returns the maximum value of all elements in the input tensor.
torch.nn		nn.ModuleList()	Hold submodules in a list.
		nn.Embedding()	Convert indices into dense vectors.
		nn.Linear()	Applies a linear transformation to the input data.
		nn.ReLU()	Apply the ReLU activation function.
		nn.Dropout(dropout)	Apply dropout to the input to prevent overfitting.
		nn.Sequential()	Sequential container for layers.

Table D-1 (continued)

		nn.CrossEntropyLoss()	Compute the cross-entropy loss.
		criterion()	Loss function for training.
		nn.Conv1d	Applies 1D convolution over input tensor.
		nn.AdaptiveAvgPool1d	Applies 1D adaptive average pooling.
torch.nn.Module		.train()	Set the model to training mode.
torch.nn.LSTM		nn.LSTM	Applies Long Short-Term Memory (LSTM) layers to learn temporal dependencies in sequential input data.
torch.optim		torch.optim.Adam(model.parameters(), lr=lr)	Adam optimiser for stochastic gradient descent.
		optimizer.zero_grad()	Clear old gradients.
		optimizer.step()	Update model parameters.
torch.Tensor		tensor.cpu().numpy()	Converts PyTorch tensor to NumPy array for evaluation.
torch.utils.data		DataLoader()	Load data in batches for training/testing.
torch.utils.data.Dataset		ClassificationDataset	Custom dataset class used to return categorical and numeric tensors for classification tasks.
xgboost.XGBClassifier	1.7.6	.fit()	Fits the XGBoost model to the training data.
		.predict_proba()	Returns class probability estimates for each input sample
optuna	4.4.0	.suggest_int()	Suggest an integer hyperparameter.
		.suggest_float()	Suggest a float hyperparameter.
		.suggest_categorical()	Suggest a categorical hyperparameter.
		optuna.create_study(direction="minimize")	Create an Optuna study for hyperparameter tuning.
		optimize(objective_cls, n_trials=20)	Run the optimisation process for a set number of trials.
		best_trial.params	Retrieve the best parameter set from the study
list	6.2.0	.extend()	Extend a list with iterable elements.
scikit-learn.metrics	1.3.2	f1_score()	Compute the F1 score.
		roc_auc_score(y_true_all, prob_1)	Compute ROC-AUC score.
scikit-learn.model_selection		train_test_split()	Split arrays or matrices into random train and test subsets.
scikit-learn.ensemble.RandomForestClassifier		.fit()	Fits the Random Forest model to the training data.
		.predict()	Predicts class labels on the validation dataset.
		.predict_proba()	Returns class probability estimates for each sample.

Table D-1 (continued)

scikit-learn.neighbors.KNeighborsClassifier		.fit()	Trains the kNN model on the training data.
		.predict()	Predicts class labels for the input samples.
scikit-learn.linear_model.SGDClassifier		.fit()	Train a stochastic gradient descent (SVM, logistic regression) model
		.predict()	Predict class labels
scikit-learn.calibration.CalibratedClassifierCV		.fit()	Train calibrates classifier via cross-validation
		.predict()	Predict class labels
		.predict_proba()	Predict calibrated class probabilities
scikit-learn.naive_bayes.MultinomialNB		.fit()	Train a multinomial Naive Bayes classifier model using input data.
		.predict()	Predicts class labels for input samples.
sklearn.preprocessing		MinMaxScaler()	Scales features to a [0, 1] range; needed for MultinomialNB to work correctly.
		.transform()	Applies scaling to input features.
		.fit_transform()	Fits the scaler and transforms the input in one step.
scikit-learn.linear_model.LogisticRegression		.fit()	Train a logistic regression model supporting L1 regularisation
		.predict()	Predict class labels
numpy	1.24.3	np.random.choice()	Randomly sample indices from an array

Notes: The full list of the packages employed in model development. Torch is used for developing Deep Learning models, while tree-based models, namely Random Forest and XGBoost, are developed by using scikit-learn.ensemble.RandomForestClassifier and xgboost.XGBClassifier, respectively. The study uses scikit-learn packages for other models, including kNN, SVM, Naïve Bayes and Logit. Optuna is used for hyperparameter tuning for all models.

Hyperparameter tuning

Table D-2 lists all the hyperparameters tuned during the learning process, along with search range and best values found. Optuna (Optuna Contributors n.d.) has been used as the search method for all the models.

Table D-2: List of hyperparameters

Model	Hyperparameter	Description	Search Range	Best Value Found
Random Forest	n_estimators	Number of boosting rounds (trees).	[50, 300]	248
	max_depth	Maximum depth of a tree.	[3, 30]	15
	min_samples_split	The minimum number of samples required to split an internal node.	[2, 10]	7
	min_samples_leaf	The minimum number of samples required to be at a leaf node.	[1, 5]	1
	max_features	The number of features to consider when looking for the best split.	["sqrt", "log2", None]	None
	random_state	Random seed for reproducibility.	42	42
	n_jobs	Number of parallel threads used to run XGBoost.	-1	-1
XGBoost	n_estimators	Number of boosting rounds (trees).	[100, 1000]	273
	max_depth	Maximum depth of a tree.	[3, 10]	8
	learning_rate	Step size shrinkage to prevent overfitting.	[0.01, 0.3]	0.23862496
	gamma	Minimum loss reduction to make a further partition on a leaf node.	[0, 0.5]	2.756357453
	min_child_weight	Minimum sum of instance weight (hessian) needed in a child.	[1, 10]	8
	subsample	Subsample ratio of the training instance.	[0.6, 1.0]	0.806536874
	colsample_bytree	Subsample ratio of columns when constructing each tree.	[0.6, 1.0]	0.649370675
	eval_metric	Evaluation metric for validation data.	"logloss"	"logloss"
	random_state	Random seed for reproducibility.	42	42
	n_jobs	Number of parallel threads used to run XGBoost.	-1	-1
ANN	hidden_size	Size of the hidden layers.	[64, 256]	147
	dropout	Dropout rate for regularisation.	[0.0, 0.5]	0.469057219
	lr	Learning rate for the optimiser.	[1e-4, 1e-2] (log scale)	0.003947213
	batch_size	Number of samples per batch.	[32, 64, 128]	128
	n_layers (num_layers)	Number of hidden layers.	[1, 3]	1
CNN	out_channels	Number of output channels (filters) in the convolutional layer.	[8, 64]	24
	kernel_size	Size of the convolutional kernel (filter).	[2, 5]	3

Table D-2 (continued)

	dropout	Dropout rate for regularisation.	[0.0, 0.5]	0.232442358
	lr	Learning rate for the optimiser.	[1e-4, 1e-2] (log scale)	0.004217233
	batch_size	Number of samples per batch.	[32, 64, 128]	128
LSTM	hidden_size	Size of the hidden state and cell state in the LSTM layers.	[32, 128]	122
	num_layers	Number of recurrent layers in the LSTM.	[1, 3]	3
	dropout	Dropout rate for regularisation.	[0.0, 0.5]	0.431453543
	lr	Learning rate for the optimiser.	[1e-4, 1e-2] (log scale)	0.002322332
	batch_size	Number of samples per batch.	[32, 64, 128]	128
kNN	n_neighbors	Number of neighbours to consider for classification.	[1, 20]	13
	weights	Weight function used in prediction.	["uniform", "distance"]	"uniform"
	p	Power parameter for the Minkowski metric (distance metric). p=1 for Manhattan distance, p=2 for Euclidean distance.	[1, 2]	1
SVM	alpha	Constant that multiplies the regularisation term.	[1e-6, 1e-1] (log scale)	0.000118046
	penalty	The penalty (regularisation) to be used.	["l2", "l1", "elasticnet"]	"elasticnet"
	loss	The loss function to be used: "hinge" for linear SVM, "log_loss" for logistic regression.	["hinge", "log_loss"]	"hinge"
	max_iter	The maximum number of passes over the training data (epochs).	1000	1000
	random_state	Random seed for reproducibility.	42	42
Naïve Bayes	alpha	Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).	[1e-4, 10] (log scale)	0.003351875
Logit (Linear)	C	Inverse of regularisation strength. Smaller values specify stronger regularisation.	[1e-4, 10.0] (log scale)	8.96994314
	penalty	The norm of the penalty. "l1" for Lasso.	"l1"	"l1"

Table D-2 (continued)

	solver	Algorithm to use in the optimisation problem. "liblinear" is suitable for L1 penalty.	"liblinear"	"liblinear"
	max_iter	Maximum number of iterations taken for the solvers to converge.	1000	1000
Logit (Polynomial)	C	Inverse of regularisation strength. Smaller values specify stronger regularisation.	[1e-4, 10.0] (log scale)	8.376387867
	penalty	The norm of the penalty. "l1" for Lasso.	"l1"	"l1"
	solver	Algorithm to use in the optimisation problem. "liblinear" is suitable for L1 penalty.	"liblinear"	"liblinear"
	max_iter	Maximum number of iterations taken for the solvers to converge.	1000	1000

Notes: The hyperparameter tuning configurations and the optimal values identified for each model used in this study. Random Forest and XGBoost both have required tuning of tree-specific parameters such as n_estimators and max_depth, with optimal values suggesting moderate model complexity (*e.g.* max_depth=15 in RF, max_depth=8 in XGBoost). DL architectures have converged on relatively shallow networks (n_layers=1 for ANN, num_layers=3 for LSTM, out_channels=24 for CNN) with moderate dropout values (~0.47 for ANN, ~0.43 for LSTM, ~0.23 for CNN), indicating a trade-off between regularisation and expressiveness. kNN has selected a relatively high number of neighbours (n_neighbors=13), and SVM has opted for an elasticnet penalty with a small alpha, balancing sparsity and smoothness. Logistic regression (both linear and polynomial) consistently prefers strong regularisation with C ≈ 8–9 and L1 penalties.

Hardware environment

Table D-3 also summarises the computing environment employed for this analysis, including the local machine and Google Colab (Google Research n.d.). Each environment has been used depending on the memory requirements of the individual training tasks.

Table D-3: Computing environments

Environment	Category	Item	Specification
Local machine	Operating System	OS	Windows 10 Pro
		Version	10.0.19045
	Hardware	Processor	Intel(R) Core(TM) i7-7th Gen @ 2.6 GHz
		Memory (RAM)	16 GB
		Storage	SSD
		GPU	Not available
	Software	Python Version	3.10
Google Colab (High-RAM Runtime)	Hardware	Available RAM	51.0 GB
		Disk Space	225.8 GB
	Software	Runtime Type	Python 3, CPU backend

Notes: Computing environments employed in this study switch depending on the memory requirement for individual tasks. Models learning from datasets with SVD-performed categoricals primarily utilise the Google Colab environment to handle the explosive dimensionality and its associated computational load and memory requirements. In both environments, Python 3 has been used to perform algorithms to process and analyse the data.

Appendix E: Performance Evaluation

This section outlines the details of the performance evaluation process for the developed models performed in the study, including the employed packages, model learning conditions, loss function, evaluation metrics, and feature importance configurations. All the subsequent steps are implemented in and can be reproduced using the notebooks (*ann.ipynb*, *cnn.ipynb*, *lstm.ipynb*, *rf.ipynb*, *xgb.ipynb*, *knn.ipynb*, *naive-bayes.ipynb*, *svm.ipynb*, *linear-logit.ipynb*, and *poly-logit.ipynb*) in the GitHub repository for the study (kokosan123 2025).

Packages

Table E-1 lists the packages used for the performance evaluation, along with the corresponding versions. PyTorch is employed for implementing DL models due to its flexibility and suitability for the task, while scikit-learn.metrics is utilised to compute global evaluation metrics derived from the confusion matrix, including precision, recall, and AUC-ROC. Additional libraries such as numpy and seaborn are used for supporting functions as described.

Table E-1: Package list employed in performance evaluation

Library Name	Version	Module/Function Name	Purpose
torch	2.7.1	torch.no_grad()	Context manager to disable gradient computation for inference.
torch.nn.Module	2.7.1	model.eval()	Sets the model to evaluation mode (e.g., disables dropout).
torch.Tensor	2.7.1	torch.Tensor.int()	Converts a Boolean tensor to an integer type for predicted labels.
		y_batch.numpy()	Converts PyTorch tensor to NumPy array.
scikit-learn.metrics	1.3.2	precision_score()	Measures the proportion of true positives among predicted positives.
		recall_score()	Measures the proportion of true positives among actual positives.
		f1_score()	Harmonic mean of precision and recall
		roc_auc_score()	Measure classification performance based on the area under the ROC curve.
		confusion_matrix()	Computes the confusion matrix for classification results.
numpy	1.24.3	np.sqrt()	Computes the square root (used for G-mean).
seaborn	0.13.2	sns.heatmap()	Creates a heatmap visualisation (used for the confusion matrix).

Notes: Full list of packages used for performance evaluation. For DL models, pyTorch is used for the specific requirement, while scikit-learn.metrics is used to obtain global evaluation metrics based on the confusion matrix, such as precision and recall, along with AUC-ROC. Other packages, such as numpy and seaborn, are used for specific purposes as described.

Table E-2 lists the packages used for the feature importance analysis (SHAP computation), along with the corresponding versions.

Table E-2: Package list employed in feature importance analysis

Library Name	Version	Module/Function Name	Purpose
shap	0.48.0	KernelExplainer	SHAP explainer for black-box models using a kernel SHAP approach
		summary_plot	Visualise SHAP values to summarise feature importance

Notes: A library and functions used to calculate SHAP values are listed. KernelExplainer is used to calculate the value using a kernel SHAP approach, while summary_plot is used to visualise the result.

Model learning conditions

Table E-3 lists learning conditions of all the models employed in the analysis, focusing on the Deep Learning parameters. The DL models (ANN, CNN, and LSTM) generally follow similar parameter configurations across most settings. In contrast, the other models are optimised based on the specific requirements of their respective libraries, with the F1-score primarily serving as the objective for performance evaluation.

Table E-3: Model learning conditions

Model	Max Epoch	Batch Size	Optimiser	Objective Function	Activation Function	Loss Function	Convergence Criterion	Early Stopping
ANN	100	32	Adam	Binary Cross-Entropy Loss	ReLU, sigmoid	*val_loss*	Not explicitly defined (implicitly controlled via early stopping)	*patience = 5*; training stops if *val_loss* and *val_auc* do not improve for 5 consecutive epochs
CNN	100	32	Adam	Binary Cross-Entropy Loss	ReLU, sigmoid	*val_loss*	Not explicitly defined (implicitly controlled via early stopping)	*patience = 5*; training stops if *val_auc* does not improve for 10 consecutive epochs
LSTM	100	32	Adam	Binary Cross-Entropy Loss	ReLU, sigmoid	*val_loss*	Not explicitly defined (implicitly controlled via early stopping)	*patience = 5*; training stops if *val_auc* does not improve for 5 consecutive epochs
Random Forest	NA	NA	NA	Gini impurity	NA	*f1_score*	NA	NA
XGBoost	100	NA	Gradient boosting with tree-based splits	Binary cross-entropy under the hood	NA	*logloss*	NA	NA
kNN	NA	NA	NA	NA	NA	*f1_score*	NA	NA
SVM	NA	NA	Sequential Minimal Optimisation, etc.	Hinge loss + regularisation	RBF (kernel)	*f1_score*	NA	NA
Naïve Bayes	NA	NA	NA	Logistic loss (negative log-likelihood)	NA	*accuracy*	NA	NA
Logit	NA	NA	*solver='liblinear'*	Logistic loss (negative log-likelihood)	NA	*f1_score*	NA	NA

Notes: Full list of model learning conditions focused on DL model parameters. DL models (ANN, CNN, LSTM) roughly share the same parameter settings for most aspects. Other models are optimised by package-specific requirements, and the F1-score is mainly used for the loss function.

Evaluation metrics

Confusion matrix (Fawcett 2006) is employed as the basis of evaluation metrics for this analysis and described as Figure E1, consisting of True Positive (TP) if positive sample (1) is predicted positive (1), False Positive (FP) if negative sample (0) is predicted positive (1), False Negative (FN) if positive sample (1) is predicted as negative (0), and finally True Negative (TN) if negative sample (0) is predicted negative (0). In the case of this study, Positive (1) refers to success, and Negative (0) refers to failure.

Figure E-1: Confusion matrix

	Actual Class	
Predicted Class	Ture Positive (**TP**)	False Positive (**FP**)
	False Negative (**FN**)	Ture Negative (**TN**)

Notes: A confusion matrix (author's creation with reference to Fawcett (2006) to evaluate the model's performance on a binary classification problem. True Positive (TP) means an actual successful sample (labelled 1) was correctly identified as successful (predicted 1). Conversely, a False Positive (FP) occurs when a failed sample (labelled 0) is incorrectly predicted as successful (predicted 1). If a successful sample (labelled 1) is mistakenly predicted as a failure (predicted 0), it's considered a False Negative (FN). A True Negative (TN) indicates that a failed sample (labelled 0) is correctly predicted as a failure (predicted 0).

Based on the matrix, several metrics to measure model ability can be calculated as follows:

$$Precision = \frac{TP}{(TP + FP)} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$Specificity = \frac{TN}{TN + FP} \tag{7}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (8)$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (9)$$

Precision is the ratio of the true positive cases of all positive-predicted samples, Recall is the ratio of the true positive cases of actual positive samples, Specificity is the ratio of true negative cases of actual negative samples, and Accuracy refers to the ratio of correctly predicted instances of the whole sample. F1-score refers to the harmonic mean of precision and recall, taking the balance of "Precision-Recall"; they are inversely related for the decision threshold (Davis and Goadrich 2006). Independent from those confusion-matrix-based metrics, the AUC-ROC metric (Fawcett 2006) corresponds to the probability that a randomly chosen positive sample is ranked higher than a randomly chosen negative one. AUC ranges from 0.5 (random guessing) to 1.0 (perfect classification) and is often used as a robust indicator of model performance, independent from the threshold based on the confusion matrix.

Feature importance configuration

Table E-4 summarises the setting for SHAP computation for both DL and tree-based models. DL models require manual preprocessing, including embedding extraction and custom prediction functions with softmax. Their inputs are high-dimensional due to concatenated embeddings and numeric features, and feature names are manually constructed. SHAP values are returned as a 2D array. Tree-based models use simpler workflows with flattened numeric inputs, including label-encoded categoricals and predict_proba() for predictions. Feature names are auto-assigned or taken from the original dataset, and SHAP values are returned as a list of arrays, one per class.

Table E-4: Feature importance configuration

Aspect	DL models	Tree-based models
Input Format	Embedded categorical features + numeric features (np.concatenate)	Flattened numeric feature array (reshape)
Embedding Handling	Manual extraction of embedding vectors via model.embeddings	Not applicable (RF handles label or numeric directly)
Prediction Function	Custom wrapper: model.model() with softmax, returns prob. of class 1	predict_proba() directly used, returns prob. of class 1
SHAP Explainer	shap.KernelExplainer	shap.KernelExplainer
SHAP Values Structure	2D array (single class case)	list of arrays (per class)
Feature Names	Manually constructed: embedding feature names + numeric names	Taken from feature_cols_cls or generated as a fallback
Input Dimensionality	High: concatenated embeddings + numeric inputs	Lower: just flattened numeric inputs (likely fewer features)

Notes: List of settings for SHAP calculations. DL models perform embedding extraction and custom softmax-based prediction. Their input data is high-dimensional, resulting from the concatenation of embeddings and numerical features, and requires explicit feature name construction. SHAP values for DL models are output as a 2D array. In contrast, tree-based models offer a simpler workflow, utilising flattened numerical inputs that include label-encoded categorical variables, and rely on standard predict_proba() for predictions. Feature names are automatically assigned or derived from the original dataset, and their SHAP values are returned as a list of arrays, with one array per class.

References

Arroyo, J. et al. (2019). Assessment of Machine Learning Performance for Decision Support in Venture Capital Investments. *IEEE Access*, 7, pp.124233–124243.

Bellman, R. and Kalaba, R. (1959). A MATHEMATICAL THEORY OF ADAPTIVE CONTROL PROCESSES. *Proceedings of the National Academy of Sciences*, 45(8), pp.1288–1290.

Blaine, B.E. (2018). Winsorizing. In *The SAGE Encyclopedia of Educational Research, Measurement, and Evaluation*. 2455 Teller Road, Thousand Oaks, California 91320: SAGE Publications, Inc. [online]. Available from: https://methods.sagepub.com/reference/the-sage-encyclopedia-of-educational-research-measurement-and-evaluation/i22326.xml [Accessed July 21, 2025].

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), pp.5–32.

Calvino, F. et al. (2025). *Young firms, job quality and inclusiveness*. OECD Publishing. [online]. Available from: https://ideas.repec.org/p/oec/stiaaa/2025-08-en.html.

Dahouda, M.K. and Joe, I. (2021). A Deep-Learned Embedding Technique for Categorical Features Encoding. *IEEE Access*, 9, pp.114381–114391.

Davis, J. and Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning - ICML ’06*. the 23rd international conference. Pittsburgh, Pennsylvania: ACM Press, pp. 233–240. [online]. Available from: http://portal.acm.org/citation.cfm?doid=1143844.1143874 [Accessed July 21, 2025].

De Amorim, L.B.V., Cavalcanti, G.D.C. and Cruz, R.M.O. (2023). The choice of scaling technique matters for classification performance. *Applied Soft Computing*, 133, p.109924.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), pp.861–874.

Feld, B. (2015). The Rule of 40% For a Healthy SaaS Company. [online]. Available from: https://feld.com/archives/2015/02/rule-40-healthy-saas-company/.

García, S., Luengo, J. and Herrera, F. (2015). *Data Preprocessing in Data Mining*. Cham: Springer International Publishing. [online]. Available from: https://link.springer.com/10.1007/978-3-319-10247-4 [Accessed July 21, 2025].

Google Research. Google Colaboratory. [online]. Available from: https://colab.google/.

Guo, C. and Berkhahn, F. (2016). Entity Embeddings of Categorical Variables. [online]. Available from: http://arxiv.org/abs/1604.06737 [Accessed July 21, 2025].

Hansen, P.C. (1987). The truncatedSVD as a method for regularization. *BIT*, 27(4), pp.534–553.

HRH2030 Consortium. (2019). *Health Workforce Skills Mix and Economic, Epidemiological, and Demographic (EED) Transitions in LMICs: Analysis and Econometric Modeling*. HRH2030 Program. [online]. Available from: https://hrh2030program.org/wp-content/uploads/2019/09/EED-report.pdf.

Independent Expert Advisory Group on a Data Revolution for Sustainable Development. (2014). *A World That Counts: Mobilising the Data Revolution for Sustainable Development*. New York: United Nations. [online]. Available from: https://www.undatarevolution.org/wp-content/uploads/2014/11/A-World-That-Counts.pdf.

International Finance Corporation. IFC's Definitions of Targeted Sectors. [online]. Available from: https://www.ifc.org/en/what-we-do/sector-expertise/financial-institutions/definitions-of-targeted-sectors.

Kar, S. (2023). Data for Better Lives: World Development Report 2021 by World Bank Group. *Journal of Data Science, Informetrics, and Citation Studies*, 2(2), pp.136–139.

Kok, J. and Berrios, M. (2019). *Small Matters: Global Evidence on the Contribution to Employment by the Self-Employed, Micro-Enterprises and SMEs*. Geneva: International Labour Organization. [online]. Available from: https://www.ilo.org/global/publications/books/forthcoming-publications/WCMS_723387/lang--en/index.htm.

Lee, K.F. (2024). Evaluating Stock Selection in the SaaS Industry: The Effectiveness of the Rule of 40. *Applied Finance Letters*, 13, pp.168–185.

Meressa, H.A. (2020). Growth of micro and small scale enterprises and its driving factors: empirical evidence from entrepreneurs in emerging region of Ethiopia. *Journal of Innovation and Entrepreneurship*, 9(1), p.11.

O'brien, R.M. (2007). A Caution Regarding Rules of Thumb for Variance Inflation Factors. *Quality & Quantity*, 41(5), pp.673–690.

OECD. Employees by Business Size. [online]. Available from: https://www.oecd.org/en/data/indicators/employees-by-business-size.html.

Optuna Contributors. Optuna: A Hyperparameter Optimization Framework. [online]. Available from: https://optuna.readthedocs.io/en/stable/.

Osborne, J. (2010). Improving your data transformations: Applying the Box-Cox transformation. , 15. [online]. Available from: https://openpublishing.library.umass.edu/pare/article/id/1546/ [Accessed July 21, 2025].

Pargent, F. et al. (2022). Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, 37(5), pp.2671–2692.

pycountry. (2024). pycountry 22.3.5. [online]. Available from: https://pypi.org/project/pycountry/.

Raschka, S. (2020). Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. [online]. Available from: http://arxiv.org/abs/1811.12808 [Accessed July 21, 2025].

Ross, G. et al. (2021). CapitalVX: A machine learning model for startup selection and exit prediction. *The Journal of Finance and Data Science*, 7, pp.94–114.

Tukey, J.W. (1977). *Exploratory data analysis*. Repr. Reading, Mass.: Addison-Wesley.

World Bank. (2020). *Doing Business 2020: Comparing Business Regulation in 190 Economies*. Washington, D.C.: World Bank. [online]. Available from: http://hdl.handle.net/10986/32436.

World Bank. (2025). World Bank Country and Lending Groups. [online]. Available from: https://datahelpdesk.worldbank.org/knowledgebase/articles/906519-world-bank-country-and-lending-groups.

Xu, Y. and Goodacre, R. (2018). On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. *Journal of Analysis and Testing*, 2(3), pp.249–262.

Data Sources

Wharton Research Data Services (WRDS). (n.d.) Orbis. University of Pennsylvania. Available at: https://wrds-www.wharton.upenn.edu/pages/ [Accessed 18 Mar. 2025].

World Bank (n.d.) DataBank: World Development Indicators. [online] Available at: https://databank.worldbank.org/ [Accessed 16 Jun 2025].

World Bank Open Data. n.d. World Bank Open Data. [online] Available at: https://data.worldbank.org/ [Accessed 17 June 2025].

Source Code

kokosan123. (2025). kokosan123/extended-research-project: Extended Research Project Source Code (Startup Ecosystem and Success in the Global South: Determinants, Machine Learning Insights, and Policy Implications). [online]. Available from: https://zenodo.org/doi/10.5281/zenodo.16534951 [Accessed July 28, 2025].