**Title of ML project:** *Exploring Global Trends in Financial Inclusion: A Focus on Account Ownership at Financial Institutions*

**—-----------------------------------------------------------------------------------------------------------**

**Name: Anupama S**

**Organization: Entri Elevate**

**Date: 04/01/2024**

—--------------------------------------------------------------------------------------------------------

1. \***\*Overview of Problem Statement**:\*\*

*Access to formal financial services, including account ownership, is a critical aspect of financial inclusion. Account ownership enables individuals to securely save money, receive payments, and access credit and insurance services. However, disparities in access persist, affecting economic growth and equity. The key issues related to account ownership include:*

***1. Contextual Importance***

* ***Economic Growth****: Higher account ownership correlates with better savings habits and investment, driving economic development.*
* ***Social Inclusion****: Bridging the gap in account ownership promotes equal opportunities for underserved populations.*
* ***Digital Economy****: Accounts facilitate participation in the digital economy, enabling e-commerce, digital payments, and access to government subsidies.*

***2. Key Challenges***

* ***Gender Disparities****: Women are often underrepresented in account ownership due to socio-cultural norms and economic inequalities.*
* ***Regional Inequalities****: Account ownership varies significantly across countries and regions, with lower rates in Sub-Saharan Africa and South Asia.*
* ***Financial Literacy****: Limited knowledge of financial products hinders the adoption of accounts.*
* ***Infrastructure****: Poor banking infrastructure and lack of digital connectivity limit access in rural and underserved areas.*
* ***Trust Issues****: Lack of trust in financial institutions prevents many from opening accounts.*

***3. Relevant Data Points from the World Bank***

* ***Global Findex Database****: Data on the percentage of adults with accounts in financial institutions or mobile money platforms.*
* ***Indicators****: Gender-specific data, rural-urban splits, and barriers such as cost, distance, and lack of documentation.*

***4. Problem Impact***

*Addressing account ownership gaps can lead to:*

* *Improved economic resilience and reduced poverty.*
* *Better financial inclusion policies at the national and international levels.*
* *Enhanced individual empowerment and access to resources.*

2. \*\***Objective**:\*\*

*Eg: To develop the best river water quality prediction model using machine learning techniques.*

*To analyze and address disparities in* ***account ownership*** *across different demographics, regions, and socio-economic groups using data from the World Bank's* ***Financial Sector*** *indicators. The goal is to identify barriers to financial inclusion and propose actionable strategies to enhance access to financial services globally.*

***Specific Objectives:***

1. ***Quantify Current Trends***
   * *Analyze global and regional data on account ownership, focusing on disparities by gender, income level, and geographic location.*
   * *Identify key drivers of account ownership growth or stagnation in various regions.*
2. ***Identify Barriers***
   * *Investigate the underlying causes of limited account ownership, such as lack of infrastructure, low financial literacy, or socio-cultural constraints.*
   * *Explore challenges faced by specific groups, including women, rural populations, and low-income households.*
3. ***Promote Financial Inclusion***
   * *Recommend targeted interventions to increase account ownership, leveraging digital solutions, policy changes, and financial literacy programs.*
   * *Highlight successful strategies and case studies from countries that have made significant progress in improving account ownership rates.*
4. ***Monitor and Evaluate***
   * *Establish key performance indicators (KPIs) to measure the impact of interventions.*
   * *Suggest frameworks for continuous monitoring of account ownership and financial inclusion efforts.*

3. \*\***Data Description**:\*\*

- Source: [Specify the source of data]

- Features: [List the features available in the dataset]

* **Source:**  
  *The data originates from the* ***World Bank Financial Inclusion Dataset****, providing global financial sector indicators and statistics. The dataset is retrieved from the World Development Indicators section on financial inclusion.*
* **Features:**  
  *The dataset includes the following features:*
  1. ***Country Name****: The name of the country or region.*
  2. ***Country Code****: A three-character code identifying the country or region.*
  3. ***Indicator Name****: Describes the specific financial inclusion indicator (e.g., Account ownership, Mobile money usage).*
  4. ***Indicator Code****: The unique code corresponding to each indicator.*
  5. ***Years (e.g., 1960, 1961, ..., 2023)****: Time-series data for financial indicators for each year.*
  6. ***Unnamed Columns****: Placeholder columns that may need renaming or removal based on relevance.*
  7. ***Indicator Values****: The numeric values representing financial statistics such as account ownership percentages or remittance costs.*

4. \***\*Data Collection**:\*\*

Import the dataset from the specified source.

- Gain insights into the data distribution, relationships, and potential patterns.

1. ***Importing the Dataset***
2. *Use pandas to import the dataset from the specified source (e.g., .csv, .xlsx, database, or API).*
3. *Verify the data structure using methods like .head(), .info(), and .describe() to understand its format, column types, and general statistics*

*.*

**2. *Understanding Data Distribution***

* **Numeric Columns**:
  + *Visualize distributions using histograms or kernel density plots (matplotlib, seaborn).*
  + *Check for skewness, central tendency (mean, median), and spread (variance, standard deviation).*
* **Categorical Columns**:
  + *Use bar charts to examine the frequency of each category.*
  + *Understand the proportion of different categories within the data.*

**3. *Exploring Relationships***

* **Correlation Matrix**:
  + *Calculate correlations between numerical features to identify linear relationships.*
  + *Use heatmaps for visual representation.*
* **Scatter Plots**:
  + *Examine relationships between pairs of variables for trends, clusters, or outliers.*
* **Group Analysis**:
  + *Group data by key categories (e.g., Country, Year) to observe trends within groups.*

**4*. Identifying Patterns***

* **Temporal Trends**:
  + *Plot time-series data to explore trends and seasonality.*
* **Geographical Patterns**:
  + *If applicable, map data to visualize geographic trends using tools like folium or geopandas.*
* **Clusters or Anomalies**:
  + *Use clustering techniques (e.g., K-Means) or anomaly detection to identify patterns that stand out.*

**5. Preliminary Observations**

* *Summarize any insights or hypotheses that emerge from the initial exploration.*
* *Document potential data quality issues like missing values, inconsistencies, or outliers for preprocessing.*

**5. \*\*Data Preprocessing - Data Cleaning:\*\***

- Handle missing values using appropriate imputation techniques.

- Check for and remove outliers using statistical methods.

- Address skewed data in numerical features through transformations.

1. **Handling Missing Values**

* **Identify Missing Values**:
  + *Use df.isnull().sum() to count missing values per column.*
* **Imputation Techniques**:
  + **Numeric Columns**:
    - **Mean/Median Imputation**: Replace missing values with the mean or median of the column.

**df['column\_name'].fillna(df['column\_name'].mean(), inplace=True)**

* + - ***Interpolation:*** *Fill missing values using interpolation methods like linear, polynomial, or time-based interpolation.*

**df['column\_name'] = df['column\_name'].interpolate(method='linear')**

* + **Categorical Columns**:
    - ***Mode Imputation****: Replace missing values with the most frequent category*.

**df['column\_name'].fillna(df['column\_name'].mode()[0], inplace=True)**

* **Drop Rows/Columns**:
* *Drop rows or columns with excessive missing values if imputation isn't suitable.*

**df = df.dropna(axis=0) # Drop rows**

**df = df.dropna(axis=1) # Drop columns**

**2. Removing Outliers**

* **Statistical Methods**:
  + *Use* ***Z-Score*** *or* ***IQR (Interquartile Range)*** *to identify and handle outliers.*

**Z-Score**:

from scipy.stats import zscore

**df = df[(zscore(df['column\_name']) < 3)]**

**IQR**:

**Q1 = df['column\_name'].quantile(0.25)**

**Q3 = df['column\_name'].quantile(0.75)**

**IQR = Q3 - Q1**

**df = df[(df['column\_name'] >= (Q1 - 1.5 \* IQR)) & (df['column\_name'] <= (Q3 + 1.5 \* IQR))]**

**Visualization**:

* *Use boxplots to visually identify and confirm outliers.*

**import seaborn as sns**

**sns.boxplot(x=df['column\_name'])**

**3. Addressing Skewed Data**

**Check Skewness**:

**Use .skew() to check the skewness of numerical features.**

**print(df['column\_name'].skew())**

**Transformations**:

* *Apply appropriate transformations to normalize skewed data.*

**Log Transformation**:

**df['column\_name'] = np.log1p(df['column\_name']) # log(1 + x) for non-negative values**

**Square Root Transformation**:

**df['column\_name'] = np.sqrt(df['column\_name'])**

**Box-Cox Transformation** (only for positive values):

**from scipy.stats import boxcox**

**df['column\_name'], \_ = boxcox(df['column\_name'])**

**Yeo-Johnson Transformation** (handles zero and negative values):

**from sklearn.preprocessing import PowerTransformer**

**pt = PowerTransformer(method='yeo-johnson')**

**df['column\_name'] = pt.fit\_transform(df[['column\_name']])**

1. **Verify Cleaned Data**

* *Re-check for missing values using .isnull().sum().*
* *Visualize data distributions again to confirm the impact of transformations.*
* *Generate summary statistics using .describe().*

**6. \*\*Exploratory Data Analysis (EDA):\*\***

- Gain insights into the data distribution, relationships, and potential patterns.

- Visualizations: Histogram, Boxplot, Pair Plot, Heatmap Correlation, Pie Diagram, Bar Plot, Count Plot, Line Plot, Kernel Density Estimation (KDE).

**Data Distribution and Insights:**

*We'll start by plotting the following visualizations:*

* ***Histogram****: Shows the distribution of data for numerical columns.*
* ***Boxplot****: Helps identify outliers and shows the spread of the data.*
* ***Pair Plot****: Visualizes relationships between numerical features.*
* ***KDE Plot****: Displays the probability density of the data.*

**2. Relationships Between Features:**

We'll also create:

* ***Heatmap****: To visualize correlations between numerical features.*
* ***Bar Plot****: To show categorical data and the frequency of each category.*
* ***Count Plot****: For counting occurrences of unique values in a categorical column.*

**3. Time-based Analysis (if applicable):**

* ***Line Plot****: To observe trends over time (if we have time-series data).*

1. **Identify Patterns and Disparities**:
   * ***Regional and Income-Based Analysis****: We can further investigate disparities in* ***account ownership*** *based on* ***income groups*** *or* ***regions****. If we don’t have a region column, we can use the* ***Country Name*** *to map countries to regions (if needed).*
   * ***Box Plots or Bar Plots****: Use these to check* ***income-based*** *or* ***regional*** *disparities in* ***account ownership****.*
2. **Advanced Visualizations**:
   * ***Heatmap****: Explore correlations between features such as* ***account ownership*** *and other socio-economic variables.*
   * ***Correlation Plots****: Visualize the relationship between* ***account ownership*** *and other financial indicators.*
3. **Insights for Stakeholders**:
   * ***Data Insights****: Generate actionable insights for policymakers. For example, which regions/countries have low* ***account ownership*** *and what could be done to improve it?*

**Possible Visualizations to Add:**

1. **Income and Account Ownership**:
   * ***Bar Plot*** *comparing* ***account ownership*** *across* ***income groups*** *(e.g., low, medium, high income).*
   * *If income data is not available, we can use* ***GDP per capita*** *as a proxy for income levels.*
2. **Geographical Distribution**:
   * ***Choropleth Map****: If we want to visualize* ***account ownership*** *on a global map, a* ***choropleth map*** *can be created to show the percentage of* ***account ownership*** *by country.*
3. **Outliers and Trends**:
   * ***Boxplot*** *to identify countries or regions with outlier values in* ***account ownership****.*

**7. \*\*Feature Engineering:\*\***

- Identify and encode categorical features using techniques like one-hot encoding or label encoding.

**Feature Engineering for Categorical Features**

*In dataset, categorical features like* ***Country Name****,* ***Country Code****,* ***Indicator Name****, and* ***Indicator Code*** *are likely to be useful for further analysis and modeling. Encoding these categorical features is crucial for machine learning models since most models require numerical input.*

*Here are two common techniques for encoding categorical features:*

**1. One-Hot Encoding**

*One-hot encoding creates new binary columns (0 or 1) for each category in a feature. It is typically used when there is no ordinal relationship between categories, which is true for features like* ***Country Name****,* ***Country Code****, etc.*

**Advantages**:

* *Preserves all categorical values as individual columns.*
* *Works well for categorical variables with no inherent ordering.*

**2. Label Encoding**

*Label encoding assigns each category a unique integer. This method is useful for ordinal data where there is a meaningful order (e.g.,* ***'Low', 'Medium', 'High'****).*

**Advantages**:

* *Reduces dimensionality (compared to one-hot encoding).*
* *Suitable for ordinal features where the order matters.*

**8. \*\*Feature Selection:\*\***

- Use algorithms like Random Forest and Select K Best to identify relevant features.

- Remove redundant or irrelevant features.

Random Forest

***Results:***

* ***Accuracy****: 96.85%*
* ***Classification Report****:*
  + ***Precision****:*
    - *Class 0: 0.90*
    - *Class 1: 0.98*
  + ***Recall****:*
    - *Class 0: 0.90*
    - *Class 1: 0.98*
  + ***F1-Score****:*
    - *Class 0: 0.90*
    - *Class 1: 0.98*
  + ***Support****:*
    - *Class 0: 3,620 samples*
    - *Class 1: 18,748 samples*

**Analysis:**

* *The* ***accuracy*** *of 96.85% is excellent, indicating that the model performs well overall.*
* ***Class 1 (majority class)*** *has very high precision (0.98), recall (0.98), and F1-score (0.98), suggesting that the model is able to correctly identify most of the positive cases (account ownership).*
* ***Class 0 (minority class)*** *has lower precision (0.90) and recall (0.90), which could be due to the imbalance in the classes. While the model is still able to identify a significant portion of the minority class, the performance is not as strong as for the majority class.*

To proceed with **Select K Best** and reducing **redundancy** in our features, we can follow these steps:

***1. Select K Best:***

*This method selects the top 'k' features based on their importance in predicting the target variable. It uses statistical tests (such as chi-square, ANOVA F-value) to evaluate the importance of each feature.*

*We'll use SelectKBest from sklearn.feature\_selection to select the top 'k' features.*

***2. Remove Redundant Features:***

*This can be achieved through correlation analysis. Features that are highly correlated with one another (above a specified threshold) can be removed to reduce redundancy.*

1. ***Correlation Analysis****: Helps remove highly correlated features, thereby reducing redundancy.*
2. ***PCA****: Can reduce the number of features while retaining most of the variance in the dataset.*
3. ***VIF****: Identifies features that are highly collinear and can be dropped to avoid multicollinearity issues in models.*

Removed 64 features with high **Variance Inflation Factor (VIF)**! This step significantly reduces multicollinearity, which can improve the stability and interpretability of our models, especially when using algorithms like **linear regression** or **logistic regression**.

**9. \*\*Split Data into Training and Testing Sets:\*\***

- Divide the dataset into training and testing subsets.

**Prepare feature set (X) and target variable (y*)****: Since we've already handled redundancy and missing data, our dataset should be ready. Make sure our feature set (X) excludes the target variable (e.g., 'Account Ownership'), and the target variable (y) is selected accordingly.*

**X = one\_hot\_encoded\_data.drop(columns=['Account Ownership']) # Features**

**y = one\_hot\_encoded\_data['Account Ownership'] # Target variable**

**Split the data into training and testing sets***: Use train\_test\_split to split the dataset. Set the test\_size parameter to define the proportion of data to be used for testing (e.g., 0.2 means 20% for testing). we can also set a random\_state for reproducibility.*

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**10. \*\*Feature Scaling:\*\***

- Scale numerical features if necessary to ensure uniform magnitude using techniques like Min-Max scaling or Standardization.

*Feature scaling is important when dealing with algorithms that are sensitive to the magnitude of features, such as distance-based algorithms (e.g., KNN, SVM) or gradient descent-based models (e.g., linear regression, logistic regression).*

1. ***Standardization****: Best used when the algorithm makes assumptions about the distribution of data (e.g., linear regression, logistic regression, SVM).*
2. ***Min-Max Scaling****: Best used when the data does not have outliers, and we need the data to be constrained to a specific range (e.g., neural networks).*

**11. \*\*Build the ML Model:\*\* (at least 5)**

- Implement various classification and regression algorithms:

- Classification: Logistic Regression, SVM, Decision Tree, MLP Classifier, Naive Bayes, Random Forest, K-Nearest Neighbors, Gradient Boost, Adaboost.

- Regression: SVR, MLP Regressor, Random Forest Regressor, Linear Regression, Gradient Boost, Adaboost.

*To implement the regression models we listed (SVR, MLP Regressor, Random Forest Regressor, Linear Regression, Gradient Boosting, and AdaBoost) and evaluate their performance*

Based on the performance metrics, here's a summary of the models' effectiveness:

**1. *Support Vector Regression (SVR)***

* ***MSE:*** *0.2098*
* ***R²:*** *0.1601*
* ***Comments:*** *SVR has a relatively lower R², meaning it explains only a small amount of the variance in the target. This suggests it's not the best fit for this dataset, though its MSE is relatively decent.*

***2. MLP Regressor***

* ***MSE:*** *0.5598*
* ***R²:*** *-1.2414*
* ***Comments:*** *The MLP Regressor performs poorly with a negative R², indicating it performs worse than a simple mean-based model. This model seems overfitted or under-optimized for the dataset.*

***3. Random Forest Regressor***

* ***MSE:*** *0.1261*
* ***R²:*** *0.4953*
* ***Comments:*** *Random Forest has a better performance, with an R² around 0.495, meaning it explains almost half of the variance in the target. This is a relatively strong performer in comparison to other models.*

***4. Linear Regression***

* ***MSE:*** *1.8e+25*
* ***R²:*** *-7.21e+25*
* ***Comments:*** *Linear Regression has extremely high MSE and an extremely negative R², indicating that the model is not a good fit. This might be due to the presence of outliers, non-linearity in the data, or other factors that the linear model cannot handle.*

***5. Gradient Boosting***

* ***MSE:*** *0.1275*
* ***R²:*** *0.4897*
* ***Comments:*** *Gradient Boosting provides a similar performance to Random Forest, with R² around 0.49. It’s a solid performer, though slightly less than Random Forest.*

***6. AdaBoost Regressor***

* ***MSE:*** *0.1653*
* ***R²:*** *0.3383*
* ***Comments:*** *AdaBoost performs well but not as well as Random Forest and Gradient Boosting. Its R² indicates a relatively moderate performance.*

***Conclusion***

* ***Best Performing Models:***
  + ***Random Forest Regressor*** *and* ***Gradient Boosting Regressor*** *both have an R² around 0.49, indicating they are the most effective models for this task.*
  + ***SVR*** *also performed decently, but its R² is lower compared to Random Forest and Gradient Boosting*.

**12. \*\*Model Evaluation:\*\***

- Regression Metrics: MAE, MSE, RMSE, R2 Score.

- Classification Metrics: Confusion Matrix, Accuracy, Precision, Recall, F1-Score, ROC Curve.

*To evaluate the performance of our regression models, it's essential to use several metrics. Below is an explanation of the key regression metrics and how we can compute them:*

***1. Mean Absolute Error (MAE)***

*The Mean Absolute Error is the average of the absolute errors. It gives an idea of how much the predictions deviate from the true values.*

**MAE=1n∑i=1n∣yi−y^i∣MAE = \frac{1}{n} \sum\_{i=1}^n |y\_i - \hat{y}\_i|MAE=n1​i=1∑n​∣yi​−y^​i​∣**

**Where:**

* *yiy\_iyi​ is the true value,*
* *y^i\hat{y}\_iy^​i​ is the predicted value,*
* *nnn is the number of observations.*

**2. Mean Squared Error (MSE)**

*MSE is the average of the squared differences between the actual and predicted values. It gives more weight to larger errors because the differences are squared.*

**MSE=1n∑i=1n(yi−y^i)2MSE = \frac{1}{n} \sum\_{i=1}^n (y\_i - \hat{y}\_i)^2MSE=n1​i=1∑n​(yi​−y^​i​)2**

**3. Root Mean Squared Error (RMSE)**

*RMSE is the square root of MSE. It's useful because it provides an error metric in the same units as the target variable.*

**RMSE=MSERMSE = \sqrt{MSE}RMSE=MSE​**

**4. R-squared (R²) Score**

*The R² score measures how well the regression model explains the variance of the data. A score of 1 means the model explains all the variance, while a score of 0 means it explains none of the variance.*

**R2=1−∑i=1n(yi−y^i)2∑i=1n(yi−yˉ)2R^2 = 1 - \frac{\sum\_{i=1}^n (y\_i - \hat{y}\_i)^2}{\sum\_{i=1}^n (y\_i - \bar{y})^2}R2=1−∑i=1n​(yi​−yˉ​)2∑i=1n​(yi​−y^​i​)2​**

Where:

* *yiy\_iyi​ is the true value,*
* *y^i\hat{y}\_iy^​i​ is the predicted value,*
* *yˉ\bar{y}yˉ​ is the mean of the true values.*

**Model Evaluation Results:**

* **MAE (Mean Absolute Error):** 0.29
  + *This indicates that on average, our model's predictions are off by 0.29 units from the actual values. The lower this value, the better.*
* **MSE (Mean Squared Error):** 0.29
  + *This metric suggests that the average of the squared differences between predicted and actual values is 0.29. Like MAE, the lower this value, the better the model.*
* **RMSE (Root Mean Squared Error):** 0.54
  + *This is the square root of the MSE and represents the error in the same units as the target variable. The lower the RMSE, the better the model is at predicting values close to the actual values.*
* **R² Score:** -0.16
  + *The R² score being negative indicates that our model performs worse than a simple model that would predict the mean value for every observation. In other words, our model is not explaining the variance in the data well. Typically, a positive R² value close to 1 is desirable for a good model.*

**13. \*\*Hyperparameter Tuning:\*\***

- Optimize model performance by tuning hyperparameters.

**Steps for Tuning to Improve Model Performance:**

1. **Choose the Model:**
   * *Based on our previous evaluations, I would recommend focusing on* ***Random Forest Regressor*** *or* ***Gradient Boosting Regressor****, since they generally perform well for many tasks.*
2. **Hyperparameter Tuning:**
   * *Use* ***GridSearchCV*** *or* ***RandomizedSearchCV*** *for hyperparameter tuning. We'll focus on optimizing the hyperparameters that most affect model performance.*

**Hyperparameters**

* **Best Hyperparameters**:
  + *C = 1*
  + *penalty = 'l2'*
  + *These hyperparameters were found to give the best model performance in the GridSearchCV.*

**Model Performance:**

* **Classification Report:**
  + **Precision:**
    - **For class 0: 0.62**
    - **For class 1: 0.94**
  + **Recall:**
    - **For class 0: 0.77**
    - **For class 1: 0.89**
  + **F1-score:**
    - **For class 0: 0.69**
    - **For class 1: 0.92**
  + **Accuracy: 0.87**
  + **Macro Average (average of all classes):**
    - **Precision: 0.78**
    - **Recall: 0.83**
    - **F1-score: 0.80**
  + **Weighted Average (taking class imbalance into account):**
    - **Precision: 0.89**
    - **Recall: 0.87**
    - **F1-score: 0.88**
* **Confusion Matrix:**
  + **For class 0 (negative class): 33 correct predictions, 10 misclassifications.**
  + **For class 1 (positive class): 170 correct predictions, 20 misclassifications.**

**Cross-Validation Accuracy:**

* **Cross-validation Accuracy***: 0.88, which means that the model performs well across multiple splits of the data.*

**Warnings:**

* **ConvergenceWarnings**:
  + *These warnings indicate that the model did not fully converge within the maximum number of iterations (max\_iter=2000). This can happen if the data is difficult for the model to learn, possibly due to scaling issues or complexity.*
  + *We can resolve this by increasing the number of iterations (max\_iter), scaling the data, or using a different solver.*

**Saving the Model:**

* **Best Model Saved**: *The best model was saved as 'best\_logistic\_regression\_model.pkl', which can be loaded later for predictions or further analysis.*

 *The model is saved as 'best\_logistic\_regression\_model.pkl' after training.*

* We can load it later and use it for predictions or further analysis without retraining.*

1. **Model Evaluation:**
   * *After tuning, use metrics like* ***MAE****,* ***MSE****,* ***RMSE****, and* ***R²*** *to evaluate and compare model performance.*

**Model Evaluation for Regression Tasks**

*Since we are working with a classification model (logistic regression), metrics like* ***Accuracy****,* ***Precision****,* ***Recall****, and* ***F1-Score*** *are typically used for classification tasks. However, for a* ***regression task****, the following metrics are commonly used:*

1. **MAE (Mean Absolute Error)**: *Measures the average magnitude of the errors in a set of predictions, without considering their direction.*
2. **MSE (Mean Squared Error)**: *Measures the average of the squares of the errors.*
3. **RMSE (Root Mean Squared Error)**: *Square root of MSE, it gives the error in the same unit as the output variable.*
4. **R² (R-squared)**: *Indicates how well the model explains the variance of the data.*

*However, if we are working with* ***classification****, we would typically use the following metrics:*

* ***Accuracy****: Percentage of correct predictions.*
* ***Precision****: Percentage of true positives among all predicted positives.*
* ***Recall****: Percentage of true positives among all actual positives.*
* ***F1-Score****: Harmonic mean of Precision and Recall.*

**Accuracy: 0.87**

**Classification Report:**

**precision recall f1-score support**

**0 0.62 0.77 0.69 43**

**1 0.94 0.89 0.92 190**

**accuracy 0.87 233**

**macro avg 0.78 0.83 0.80 233**

**weighted avg 0.89 0.87 0.88 233**

**Confusion Matrix:**

**[[ 33 10]**

**[ 20 170]]**

**MAE (Mean Absolute Error): 0.13**

**MSE (Mean Squared Error): 0.13**

**RMSE (Root Mean Squared Error): 0.36**

**R² (R-squared): 0.14**

**1. Accuracy: 0.87**

* This means that **87% of the predictions** made by the model were correct.

**2. Classification Report:**

* **Precision**:
  + ***Class 0****: 0.62 — 62% of the predicted '0's were actually '0'.*
  + ***Class 1****: 0.94 — 94% of the predicted '1's were actually '1'.*
* **Recall**:
  + ***Class 0****: 0.77 — 77% of actual '0's were correctly predicted.*
  + ***Class 1****: 0.89 — 89% of actual '1's were correctly predicted.*
* **F1-Score**:
  + ***Class 0****: 0.69 — The harmonic mean of precision and recall for class '0'.*
  + ***Class 1****: 0.92 — The harmonic mean of precision and recall for class '1'.*
* **Accuracy**: 0.87 — *The overall percentage of correct predictions*.
* **Macro Average**:
  + **Precision**: 0.78
  + **Recall**: 0.83
  + **F1-Score**: 0.80
* **Weighted Average**:
  + **Precision**: 0.89
  + **Recall**: 0.87
  + **F1-Score**: 0.88

**3. Confusion Matrix:**

* ***True Negatives****: 33 (Correctly predicted '0')*
* ***False Positives****: 10 (Predicted '1' but it was actually '0')*
* ***False Negatives****: 20 (Predicted '0' but it was actually '1')*
* ***True Positives****: 170 (Correctly predicted '1')*

**4. Regression Metrics (despite being a classification problem, these metrics are often used for continuous predictions):**

* ***MAE (Mean Absolute Error)****: 0.13 — The average absolute difference between the predicted and actual values.*
* ***MSE (Mean Squared Error)****: 0.13 — The average of the squared differences between the predicted and actual values.*
* ***RMSE (Root Mean Squared Error)****: 0.36 — The square root of MSE, providing an error measure in the same units as the target variable.*
* ***R² (R-squared)****: 0.14 — Indicates how well the model explains the variance in the target variable. A value of 0.14 suggests that the model explains only 14% of the variance in the data.*

**Summary:**

* **Good Precision for Class 1 (0.94)**: *The model performs well when predicting class '1'.*
* **Recall for Class 1***: 0.89, meaning the model identifies most of the '1' instances.*
* **Confusion Matrix** *shows that the model is fairly good at detecting class '1' but does make some mistakes in predicting class '0' (10 false positives and 20 false negatives).*
* **Model Performance**: *The model seems to be underperforming in terms of class '0' predictions and could benefit from further tuning.*

**Potential Next Steps:**

1. **Class Imbalance***: We might want to explore strategies like* ***SMOTE (Synthetic Minority Oversampling Technique)*** *or* ***class weights adjustment*** *to improve the recall for class '0'.*
2. **Further Tuning**: *We could tune hyperparameters further or consider other classification algorithms (e.g., Random Forest, XGBoost) that might perform better on imbalanced classes.*

**14. \*\*Save the Model:\*\***

- Save the trained model for future use.

**import joblib**

**# Assuming 'grid\_search.best\_estimator\_' is the trained model after hyperparameter tuning**

**best\_logistic\_regression\_model = grid\_search.best\_estimator\_**

**# Save the model using joblib**

**joblib.dump(best\_logistic\_regression\_model, 'best\_logistic\_regression\_model.pkl')**

**print("Model saved successfully!")**

**Load the model:**

*When we need to load the model for future predictions*

**# Load the saved model**

**loaded\_model = joblib.load('best\_logistic\_regression\_model.pkl')**

**# Use the loaded model to make predictions**

**predictions = loaded\_model.predict(X\_test)**

**15. \*\*Test with Unseen Data:\*\***

- Assess the model's performance on unseen data.

**import pandas as pd**

**import numpy as np**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import classification\_report, confusion\_matrix**

**from sklearn.compose import ColumnTransformer**

**from sklearn.pipeline import Pipeline**

**from sklearn.preprocessing import OneHotEncoder**

**# Create synthetic data with the same structure as our original dataset**

**synthetic\_data = pd.DataFrame({**

**'Country Code': ['ABW', 'AFG', 'AFW'],**

**'2020': np.random.randn(3),**

**'2021': np.random.randn(3),**

**'2022': np.random.randn(3),**

**'Account Ownership': [0, 1, 0]**

**})**

**# Separate features (X) and target (y)**

**X\_unseen = synthetic\_data.drop(columns=['Account Ownership']) # Features**

**y\_unseen = synthetic\_data['Account Ownership'] # Target**

**# Step 1: Handle categorical features**

**# We apply OneHotEncoding to 'Country Code' column (non-numeric)**

**preprocessor = ColumnTransformer(**

**transformers=[**

**('cat', OneHotEncoder(), ['Country Code']), # Apply OneHotEncoding to 'Country Code'**

**('num', StandardScaler(), ['2020', '2021', '2022']) # Apply scaling to numeric columns**

**])**

**# Step 2: Define a pipeline with preprocessing and the model**

**model = LogisticRegression(max\_iter=1000)**

**pipeline = Pipeline(steps=[**

**('preprocessor', preprocessor),**

**('model', model)**

**])**

**# Assuming we've trained the best model before (use the same model in this case)**

**pipeline.fit(X\_unseen, y\_unseen)**

**# Step 3: Make predictions on the unseen data**

**y\_pred\_unseen = pipeline.predict(X\_unseen)**

**# Evaluate the model**

**print("Classification Report:\n", classification\_report(y\_unseen, y\_pred\_unseen))**

**print("Confusion Matrix:\n", confusion\_matrix(y\_unseen, y\_pred\_unseen))**

The model has performed perfectly on the synthetic unseen data, with an accuracy of 100%.

**Classification Report:**

* ***Precision, Recall, F1-Score*** *for both classes (0 and 1) are all 1.00, indicating perfect classification for both classes in the synthetic dataset.*
* ***Accuracy*** *is also 1.00, meaning the model correctly predicted all the instances in the synthetic data*.

**Confusion Matrix:**

* *The matrix shows that for:*
  + *Class 0: All 2 instances were correctly predicted as 0 (no false positives or false negatives).*
  + *Class 1: The single instance was correctly predicted as 1.*

*This result is expected since the synthetic data was randomly generated and could be easily separated by the model.*

**16. \*\*Interpretation of Results (Conclusion):\*\***

- Analyze the model's performance and draw conclusions. Discuss any limitations of the dataset.

**Interpretation of Results:**

1. ***Model Performance:*** *The model's performance on the synthetic unseen data was flawless, with an accuracy of 100%. Both precision, recall, and F1-score for both classes (0 and 1) were 1.00, which means the model was able to perfectly predict all instances in the synthetic dataset. This result is expected given that the synthetic data was created with a simple structure, allowing the model to easily learn the patterns and perform well.*
2. **Confusion Matrix:**
   * ***Class 0:*** *The model predicted all instances of class 0 correctly (2 instances).*
   * ***Class 1:*** *The model correctly predicted the only instance of class 1.*

*The confusion matrix confirms that there were no false positives or false negatives, indicating that the model was perfectly able to distinguish between the two classes in this simplified scenario.*

**Conclusion:**

* ***Accuracy:*** *The model achieved an accuracy of 1.00 on the synthetic data, but this is not necessarily indicative of real-world performance, as the synthetic data might not represent the full complexity of real-world data.*
* ***Overfitting Risk:*** *Since the synthetic dataset was small and simplified, there is a risk of overfitting. The model might have learned the specific patterns in the synthetic data too well, which may not generalize well to more complex, real-world datasets.*
* ***Precision, Recall, and F1-Score:*** *These metrics were also 1.00 for both classes, further supporting the idea that the model was able to make perfect predictions on the synthetic data. However, in a real-world scenario, the class distribution could be more imbalanced, leading to different results.*

**Limitations of the Dataset:**

* ***Synthetic Nature:*** *The synthetic data used in this example might not be representative of the actual complexities and variations in the real data, which may have more noise, outliers, and complex relationships.*
* ***Imbalanced Data:*** *If the real dataset is imbalanced (where one class is much more prevalent than the other), the model's performance may degrade. The model may struggle to learn the minority class adequately without further tuning or resampling techniques like SMOTE.*
* ***Feature Complexity:*** *The synthetic data was designed to be simple, but real data may have more features and interactions that make predictions more difficult, requiring more sophisticated preprocessing, feature engineering, or more complex models.*

**17. \*\*Future Work:\*\***

1. **Explore Deep Learning Algorithms:**
   * ***Why:*** *Deep learning algorithms, such as neural networks, convolutional neural networks (CNNs), or recurrent neural networks (RNNs), have shown exceptional performance in complex datasets, especially when dealing with large amounts of data and capturing intricate patterns.*
   * ***Action:*** *Implement deep learning models like multi-layer perceptrons (MLPs) or use pre-trained models for specific tasks like image processing (if relevant to the data). Start by comparing their performance against traditional models to assess potential improvements in accuracy and generalization.*
2. **Update the Model Periodically with New Data:**
   * ***Why:*** *Data trends may evolve over time, and the model’s performance could degrade if not updated regularly. By incorporating fresh data, the model can adapt to new patterns and maintain or improve its predictive accuracy.*
   * ***Action:*** *Establish a process for regular model retraining, such as monthly or quarterly updates, using new labeled data. Automating this pipeline will ensure the model remains current without requiring manual intervention.*
3. **Address Imbalanced Data through Resampling Techniques:**
   * ***Why:*** *Imbalanced datasets can lead to biased model predictions, often favoring the majority class. Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or under-sampling the majority class can improve model performance in such scenarios.*
   * ***Action:*** *Implement resampling techniques or consider cost-sensitive learning methods where misclassifying the minority class is penalized more heavily. Evaluate how resampling affects model performance through cross-validation.*
4. **Consider Adding More Features to Enhance Predictive Power:**
   * ***Why:*** *More informative features can help the model to make better predictions by providing additional context or capturing unseen patterns in the data.*
   * ***Action:*** *Analyze domain-specific features that could be added to the dataset, such as demographic information, economic indicators, or time-based features. Feature engineering techniques like polynomial features, interaction terms, and embedding categorical variables might also be useful.*

**Dataset Details:**

1. ***Columns****: The dataset contains a mix of numeric and non-numeric columns, including time-series data from 1960 to 2023 and a target variable (Account Ownership).*
2. ***Missing Values****: There are no missing values in the dataset.*
3. **Class Distribution (Before SMOTE)**:
   * *Class 1 (Account Ownership = 1): 781 samples.*
   * *Class 0 (Account Ownership = 0): 151 samples.*
4. **Class Distribution (After SMOTE)**:
   * Both classes now have 781 samples, achieving a balanced distribution.

**Next Steps:**

1. **Feature Scaling**:
   * *If needed, scale the numeric features (e.g., columns from 1960 to 2023) to standardize their range for machine learning models.*
2. **Model Training**:
   * *Train our machine learning model using the balanced dataset (X\_resampled and y\_resampled).*
   * *Consider using algorithms like logistic regression, random forests, or gradient boosting, depending on the problem.*
3. **Validation**:
   * *Use the testing set (X\_test, y\_test) to evaluate the model's performance.*
   * *Metrics like accuracy, precision, recall, and F1-score are essential to assess model effectiveness.*
4. **Feature Importance**:
   * *If interpretability is crucial, analyze feature importance to identify the most influential predictors.*

*- Consider adding more features to enhance predictive power.*

*To enhance the predictive power of my model, consider the following strategies for feature engineering and adding more features:*

**1. Aggregate Features**

* ***Yearly Averages****: Compute the average value of indicators for specific years or across a range of years (e.g., mean of indicators for the 2000s).*
* ***Rolling Averages****: Create rolling averages over a fixed period (e.g., 5-year rolling averages for smoother trends).*

**2. Create Ratios**

* *Ratios between different indicators (e.g., ratio of GDP growth to population growth if such data exists in the dataset).*
* *Indicator-based comparisons to highlight disparities (e.g., ratio of urban vs. rural population if applicable).*

**3. Temporal Trends**

* ***Growth Rates****: Compute growth rates between consecutive years for important indicators.*
* ***Difference Features****: Calculate year-over-year differences to capture changes.*

**4. Category-Based Aggregation**

* *Group countries by regions (e.g., continents) or income levels (if such information is available) and compute group-based averages or deviations.*

**5. External Data**

* *Integrate additional datasets that relate to the indicators. Examples include:*
  + ***World Bank Data****: Economic indicators, population, employment rates.*
  + ***Geographical Data****: Region-based socio-economic or environmental data.*
  + ***Demographics****: Age distribution, literacy rates, or income levels.*

**6. Interaction Features**

* *Combine features using interactions (e.g., Indicator A \* Indicator B) to capture nonlinear relationships.*
* *Polynomial combinations of numeric features to explore higher-order relationships.*

**7. Time-Specific Flags**

* *Add binary flags for key historical events or milestones in certain years.*
* *Flags for certain ranges (e.g., pre-2000 vs. post-2000 economic policies).*

**8. Dimensionality Reduction**

* *Use techniques like PCA to create features that capture maximum variance from the existing time-series data.*

**9. Clustering**

* *Perform clustering on countries using their feature values to create cluster IDs and use those IDs as a feature.*

## **Files Utilized**

*Several datasets and model files were used, including:*

*- cleaned\_data\_dec27.xlsx*

*- cleaned\_data\_no\_outliers\_isolation\_forest.xlsx*

*- clustered\_dataset.csv*

*- financial\_inclusion\_Dec27.ipynb*

*- best\_logistic\_regression\_model.pkl*