National College of Ireland

Configuration Manual

MSc Research Project
Financial Technology

Afolashade Akinbowale
Student ID: x22234187

School of Computing
National College of Ireland

Supervisor: 1) Faithful Chiagoziem Onwuegbuche
2) Noel Cosgrave

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Afolashade Akinbowale……

Student ID: ……x22234187…….……

Programme: MSc.FinTech……**Year:**…2024……..

Module: Research Project…….……

Lecturer: 1) Faithful ChiagoziemOnwuegbuche…….……
Submission 2) Noel Cosgrave
Due Date: ……August 12th,2024 …….……

Project Title: Investigating the level of financial inclusion in the African Continental Free Trade Zone Areas…….……

Word Count: ……1,381…… **Page Count:** ……15…….……

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

Signature: Afolashade Akinbowale……

Date: ……12/08/24……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

computer.	

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

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

Configuration Manual

Afolashade Akinbowale
Student ID: x22234187

1 The Introduction

This part of the research is focused on the technicality, methods, and techniques required to conduct an in-depth research analysis titled: *"Investigating the Level of Financial Inclusion in the African Continental Free Trade Zone Areas"*.

2 System Requirements

2.1 Hardware

- Windows 11 Pro, 64-bit operating system, x64-based processor
- Intel(R) Core (TM) i7-7600U CPU @ 2.80GHz 2.90 GHz
- 8.00 GB (7.88 GB usable)
- DESKTOP-48VD1J2
- Windows Feature Experience Pack 1000.22001.1000.0

2.2 Software

- Research report was written using Microsoft office word document
- Microsoft excel was downloaded from data sources and used to examine the data
- Python programming language and Google Colab for used for data pre-processing, mining and statistical analysis

3 Data Source

The data used in this research are from two comprehensive sources. One is the World Bank Findex database[1], and the second one is the International Monetary Fund Financial Access Survey[2]. Findex data was downloaded and saved as DatabankWide Worldbank while the second dataset was saved as FAS_Latest_Data.

[1] https://www.worldbank.org/en/publication/globalfindex
[2] https://data.imf.org/?sk=E5DCAB7E-A5CA-4892-A6EA-598B5463A34C&sId=1460043522778

4 Analysis

4.1 Python packages installed for Analysis

Installed packages on Google Colab	Relevance to studies
Pandas	For data manipulation, mining and analysis
numpy	Numerical computation of data
Sklearn.decomposition	To decompose the 2nd stage of rPCA
matplotlib.pyplot	For visualization of analysis and data missingness
seaborn	For statistical graphing
pandas.plotting	For plotting of results and correlation analysis
missingno	For pattern and distribution of missing data
scipy.stats	For descriptive statistics
ppca	For imputing missing data
scikit-learn	Data pre-processing
IterativeSVD	Python form for robust principal component analysis
factor_analyzer	For validity test - Bartletts and KMO test
statsmodels.stats.outliers	For multicollinearity test

4.2 Pre-processing and Transformation of Data

The data used in this study were from two diverse set of sources, therefore, handling each data individually before merging both is a key step in data pre-processing and transformation.

4.2.1 World Bank Findex Database

At this point, the dataset is imported, and saved as DatabankWide Worldbank, this is the name from source. The view the dataframe.

```
df = pd.read_csv('/content/DatabankWide Worldbank.csv')

df
```

4.2.1.1 Data Cleaning

- View basic information about the dataset to understand the data structure
- Select and filter to keep the necessary columns (variable) for the study.

```
[ ]  # Index columns to identify unneeded ones
     print("Columns with numerical indices:")
     for idx, column_name in enumerate(df.columns):
         print(f"Column {idx}: {column_name}")

     # Columns to keep
     columns_to_keep_indices = [0, 2, 6, 7, 29, 34, 87, 52, 73, 74, 81, 78, 22, 206, 86, 88, 87, 116, 155, 213, 223, 107, 40, 46, 42, 48]

     # Created a new DataFrame with only the specified columns
     df_reduced = df.iloc[:, columns_to_keep_indices]

     # To view the first few rows of the new DataFrame
     df_reduced

     # Save the new DataFrame to a new CSV file
     df_reduced.to_csv('reduced_data.csv', index=False)
```

- Select and filter to keep the necessary rows (observations) for the study. At this point, all AfCFTA countries were included.

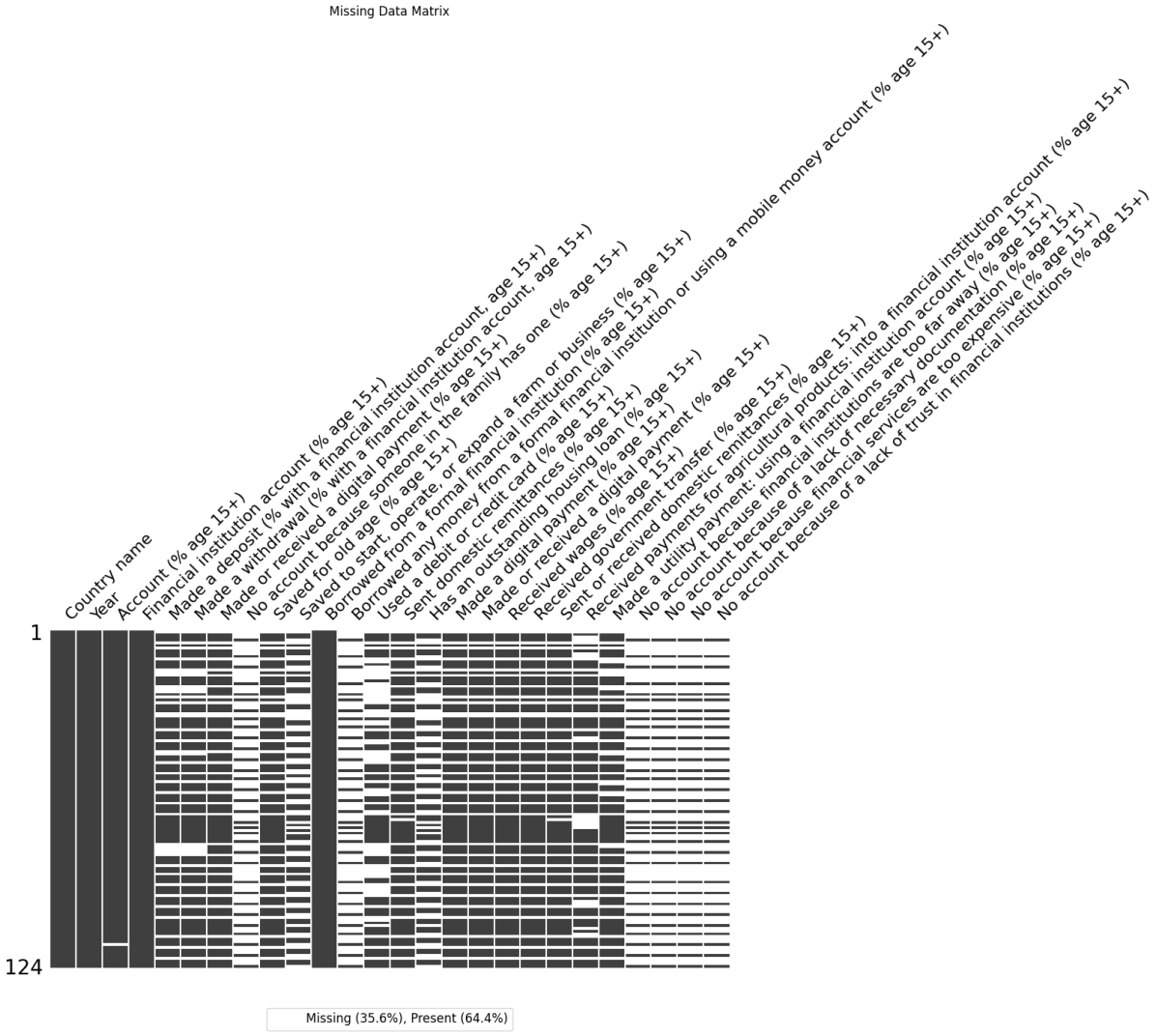
```
# List of countries to keep
countries_to_keep = [
    'Ghana', 'Kenya', 'Rwanda', 'Niger', 'Chad', 'Eswatini', 'Guinea',
    'Côte d'Ivoire', 'Mali', 'Namibia', 'South Africa', 'Congo, Rep.',
    'Djibouti', 'Mauritania', 'Uganda', 'Senegal', 'Togo', 'Egypt',
    'Ethiopia', 'Gambia', 'Sierra Leone', 'Zimbabwe', 'Burkina Faso',
    'São Tomé & Príncipe', 'Equatorial Guinea', 'Gabon', 'Mauritius',
    'Central African Rep.', 'Angola', 'Lesotho', 'Tunisia', 'Cameroon',
    'Nigeria', 'Malawi', 'Zambia', 'Algeria', 'Burundi', 'Seychelles',
    'Tanzania', 'Cabo Verde', 'Democratic Republic of the Congo', 'Morocco',
    'Guinea-Bissau', 'Botswana', 'Comoros', 'Mozambique'
]

# Filtered the DataFrame to keep only rows where 'Country name' is in the list of countries to keep
df_filtered = df_reduced[df_reduced['Country name'].isin(countries_to_keep)]

# Reset the index
df_filtered.reset_index(drop=True, inplace=True)
```

- Convert the data to numerical from % - percentages to strings in python
- Countries were re-labelled to match the FAS dataset and in a saved in a df_cleaned dataframe

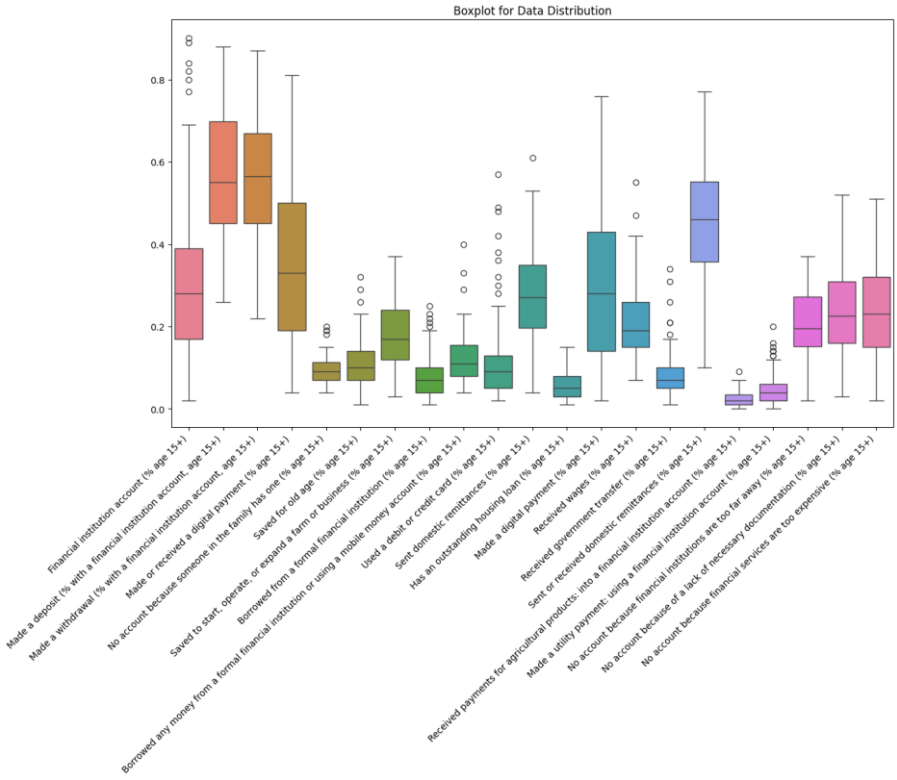
```
# Changed the country names
df_cleaned['Country name'].replace({
    'Swaziland': 'Eswatini',
    'Egypt, Arab Rep.': 'Egypt',
    'Congo, Dem. Rep.': 'Congo, Democratic Republic of',
    'Congo, Rep.': 'Congo, Republic of',
    "Cote d'Ivoire": "Côte d'Ivoire",
    'Mozambique': 'Mozambique, Republic of'
}, inplace=True)

# Print the modified DataFrame
print("\nModified DataFrame with Harmonized Country Names:")
df_cleaned
```

4.2.1.2 Exploratory Data Analysis

- Visualize data missingness using missing data matrix



- Visualize the distribution of data



4.3 DATABASE 2----Financial Assess Survey (FAS)

- Import data into Colab for analysis

```
]  # Load the dataset with appropriate encoding
   FAS_data = pd.read_csv('/content/FAS_Latest_Data.csv', encoding='latin1')

   # Display the first few rows of the dataframe to verify
   FAS_data
```

- View the structure and number of columns to understand the dataset structure
- Define indices of columns to keep and create a new dataframe
- Define the list of countries to keep in the rows. The FAS database is collated yearly therefore, the relevant years 2011, 2014, 2017 and 2021 was used.

```
countries_to_keep = [
    'Ghana', 'Kenya', 'Rwanda', 'Niger', 'Chad', 'Eswatini', 'Guinea',
    'Côte d'Ivoire', 'Mali', 'Namibia', 'South Africa', 'Congo, Rep.',
    'Djibouti', 'Mauritania', 'Uganda', 'Senegal', 'Togo', 'Egypt',
    'Ethiopia', 'Gambia', 'Sierra Leone', 'Zimbabwe', 'Burkina Faso',
    'São Tomé & Príncipe', 'Equatorial Guinea', 'Gabon', 'Mauritius',
    'Central African Rep.', 'Angola', 'Lesotho', 'Tunisia', 'Cameroon',
    'Nigeria', 'Malawi', 'Zambia', 'Algeria', 'Burundi', 'Seychelles',
    'Tanzania', 'Cabo Verde', 'Democratic Republic of the Congo', 'Morocco',
    'Guinea-Bissau', 'Botswana', 'Comoros', 'Mozambique'
]

# Define the list of years to keep
years_to_keep = [2011, 2014, 2017, 2021]  # Replace with the specific years you want to keep


# Filter the DataFrame based on countries and years
FAS_reduced_filtered = FAS_data_filtered[ •
    (FAS_data_filtered['Country.Name'].isin(countries_to_keep)) &
    (FAS_data_filtered['Calendar.Calendar element name'].isin(years_to_keep))
]

# Display the first few rows of the filtered dataframe to verify
FAS_reduced_filtered
```

- Rename columns for consistency as labelled in the World Bank data

```
# Rename columns for consistency
FAS_reduced_filtered = FAS_reduced_filtered.rename(columns={
    'Country.Name': 'Country name',
    'Calendar.Calendar element name': 'Year'
})

# Display the first few rows of the dataframe to verify
FAS_reduced_filtered
```

- Convert the 'Year' column to string
- Visualize missingness for each column using tables heatmap, bar chart
- Visualize and observe the relationship between variables using boxplot
- Check and drop duplicates in the dataset
- Convert numerical columns to float and categorical columns to category type

```
Percentage of missing values in each column:
Country name                                                                0.00000
Year                                                                        0.00000
Number of registered mobile money agent outlets                            32.81250
Number of debit cards                                                      47.65625
Number of commercial bank branches per 1,000 km2                            6.25000
Number of ATMs per 1,000 km2                                               10.15625
Number of ATMs per 100,000 adults                                          10.15625
Number of depositors with commercial banks per 1,000 adults                27.34375
Outstanding loans from commercial banks (% of GDP)                          7.81250
Number of mobile money transactions (during the reference year) per 1,000 adults    30.46875
dtype: float64
```

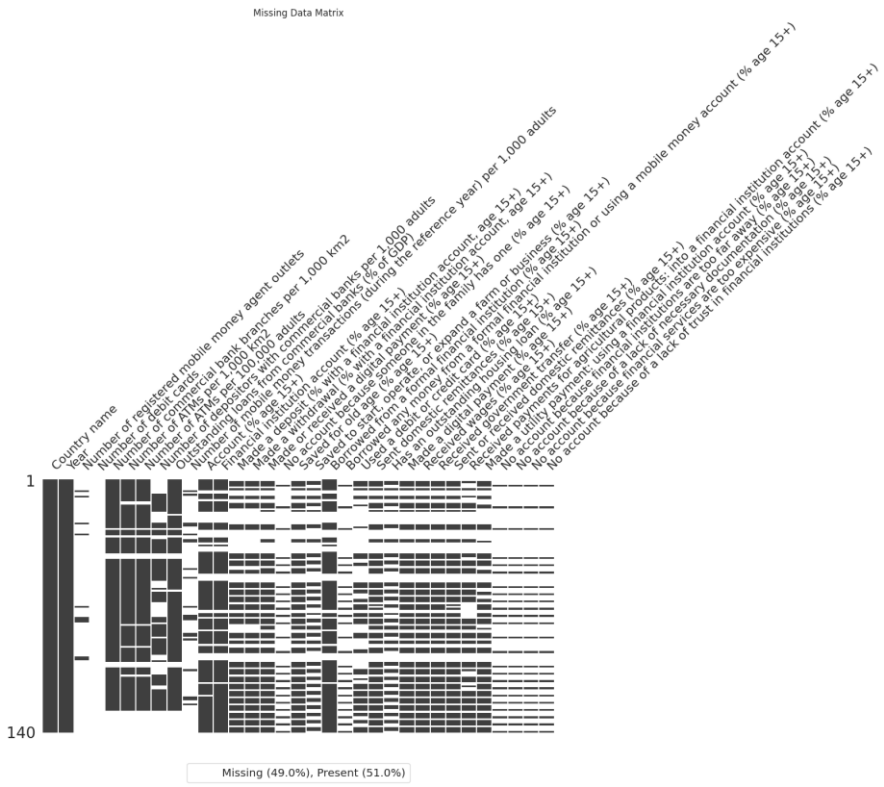
4.4 Combined dataset – Findex cleaned data and FAS cleaned data

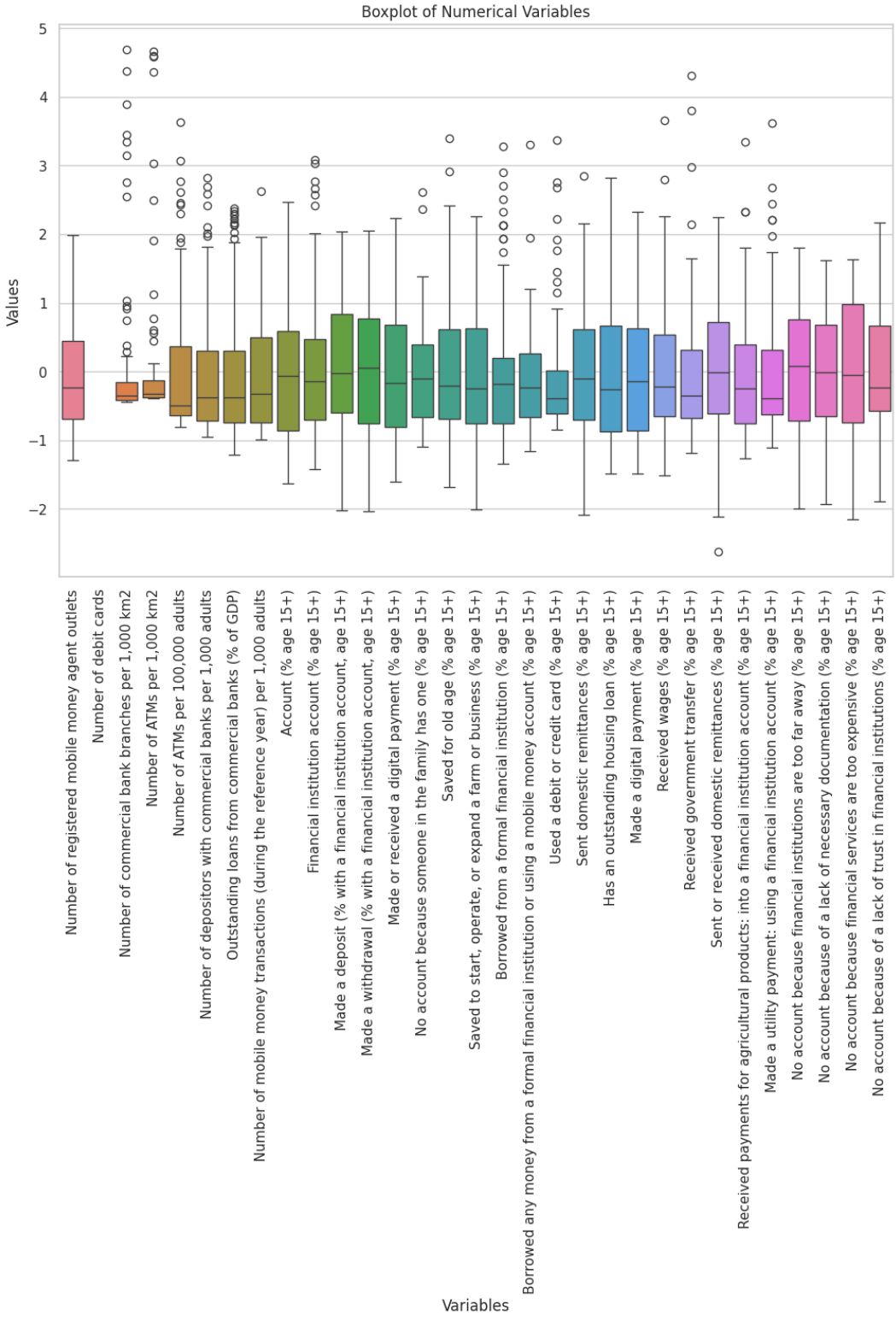
Data Pre-processing

- Convert 'Year' to string data type in both Data frames
- Name and save new data frame
- Drop the following countries with only one data point (year) out of four years

```
[ ] # Define countries and years to drop
    countries_to_drop = ['Comoros', 'Eswatini', 'Ethiopia', 'Lesotho', 'Mozambique, Rep. of', 'Mozambique']
    countries_years_to_drop = {
        'Botswana': [2022],
        'Chad': [2022],
        'Niger': [2022]
    }

    # Drop the specified countries
    Finalproject_data = Finalproject_data[~Finalproject_data['Country name'].isin(countries_to_drop)]

    # Drop the specified countries and years
    for country, years in countries_years_to_drop.items():
        Finalproject_data = Finalproject_data[~((Finalproject_data['Country name'] == country) & (Finalproject_data['Year'].isin(map(str, years))))]


    print(Finalproject_data.info())
```

- Check and remove duplicates countries
- Ensure consistent entries for categorical data and Convert “NA’ strings to NaN
- Convert columns that should be numeric to numeric types
- Normalize the data using standardscaler
- View combined missing data chart and boxplot




Boxplot of Numerical Variables
Values
5
4
3
2
1
0
−1
−2
Number of registered mobile money agent outlets
Number of debit cards
Number of commercial bank branches per 1,000 km2
Number of ATMs per 1,000 km2
Number of ATMs per 100,000 adults
Number of depositors with commercial banks per 1,000 adults
Outstanding loans from commercial banks (% of GDP)
Number of mobile money transactions (during the reference year) per 1,000 adults
Account (% age 15+)
Financial institution account (% age 15+)
Made a deposit (% with a financial institution account, age 15+)
Made a withdrawal (% with a financial institution account, age 15+)
Made or received a digital payment (% age 15+)
No account because someone in the family has one (% age 15+)
Saved for old age (% age 15+)
Saved to start, operate, or expand a farm or business (% age 15+)
Borrowed from a formal financial institution (% age 15+)
Borrowed any money from a formal financial institution or using a mobile money account (% age 15+)
Used a debit or credit card (% age 15+)
Sent domestic remittances (% age 15+)
Has an outstanding housing loan (% age 15+)
Made a digital payment (% age 15+)
Received wages (% age 15+)
Received government transfer (% age 15+)
Sent or received domestic remittances (% age 15+)
Received payments for agricultural products: into a financial institution account (% age 15+)
Made a utility payment: using a financial institution account (% age 15+)
No account because financial institutions are too far away (% age 15+)
No account because of a lack of necessary documentation (% age 15+)
No account because financial services are too expensive (% age 15+)
No account because of a lack of trust in financial institutions (% age 15+)
Variables

5 Probabilistic Principal Component Analysis (PPCA) to compute the combined dataset

PPCA was selected over other methods due to its ability to accurately impute missing data of large datasets. Robust principal component analysis is designed to handle outliers and data that are not normally distributed (Candes *et al*., 2011).

- Install PPCA and fancyimpute in google Colab to impute missing data.

```
[ ] !pip install ppca
```

```
Collecting ppca
  Downloading ppca-0.0.4-py3-none-any.whl.metadata (400 bytes)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/d
Requirement already satisfied: scipy in /usr/local/lib/python3.10/d
Downloading ppca-0.0.4-py3-none-any.whl (6.7 kB)
Installing collected packages: ppca
Successfully installed ppca-0.0.4
```

```
pip install fancyimpute
```

```
Collecting fancyimpute
  Downloading fancyimpute-0.7.0.tar.gz (25 kB)
  Preparing metadata (setup.py) ... done
Collecting knnimpute>=0.1.0 (from fancyimpute)
  Downloading knnimpute-0.1.0.tar.gz (8.3 kB)
```

- Visualize the dataframe to eyeball missing data (NA's)
- Fill missing data using probabilistic principal component analysis. Apply PPCA using the fancy impute and IterativeSVD in python.
- Save and rename the imputed data

```
from fancyimpute import IterativeSVD

# Separate the columns to impute and the columns to keep as they are
columns_to_impute = FinalProject.columns[2:]  # Assuming first two columns are 'Country name' and 'Year'
columns_to_keep = FinalProject.columns[:2]

# Extract the data to impute
data_to_impute = FinalProject[columns_to_impute].values

# Impute missing data using PPCA (IterativeSVD)
imputer = IterativeSVD(rank=5)
imputed_data = imputer.fit_transform(data_to_impute)

# Combine the imputed data with the original non-imputed columns
imputed_df = pd.DataFrame(imputed_data, columns=columns_to_impute)
complete_data = pd.concat([FinalProject[columns_to_keep].reset_index(drop=True), imputed_df], axis=1)
```

As seen above; [2:] is used to exclude the first two columns named country and year as they both don't require to be filled. Rank is used to denote the number of key components used to

explain the variance. This can be increased accordingly. However, if 3 (rank = 3) explains over 95% of the variance then, then 3 should be adequate. Otherwise, the rank should be increased till the variance is well explained. Also, reset_index is used to ensure that the index is properly aligned between the two datasets

- Reclassify and group the usage variables
- Delete the variables that makes up the new classification to avoid duplication

```
# Drop the specified columns
columns_to_drop = [
    'Number of depositors with commercial banks per 1,000 adults',
    'Outstanding loans from commercial banks (% of GDP)',
    'Number of mobile money transactions (during the reference year) per 1,000 adults',
    'Account (% age 15+)',
    'Made a deposit (% with a financial institution account, age 15+)',
    'Made a withdrawal (% with a financial institution account, age 15+)',
    'Made or received a digital payment (% age 15+)',
    'No account because someone in the family has one (% age 15+)',
    'Saved for old age (% age 15+)',
    'Saved to start, operate, or expand a farm or business (% age 15+)',
    'Borrowed from a formal financial institution (% age 15+)',
    'Borrowed any money from a formal financial institution or using a mobile money account (% age 15+)',
]
```

- View the shape of the dataset after dropping duplicates. Shape is used to give details of rows and columns in python

```
[ ]  complete_data_dropped.shape

(140, 24)
```

- Standardize the data features to properly scale and prepare for rPCA analysis

```
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
numeric_data_scaled = scaler.fit_transform(numeric_data)
```

- Check normality for each feature using Shapiro-Wilk test. Output looks like this;

```
Shapiro-Wilk Test for Number of registered mobile money agent outlets: Statistics=0.9642720611563298, p=0.0017774731839560111
Number of registered mobile money agent outlets does not look normally distributed (reject H0)
Shapiro-Wilk Test for Number of debit cards: Statistics=0.9852456529256954, p=0.1771994637742818
Number of debit cards looks normally distributed (fail to reject H0)
Shapiro-Wilk Test for Number of commercial bank branches per 1,000 km2: Statistics=0.7767396270002526, p=9.863115375904284e-13
Number of commercial bank branches per 1,000 km2 does not look normally distributed (reject H0)
Shapiro-Wilk Test for Number of ATMs per 1,000 km2: Statistics=0.7401979701044195, p=8.075100679713665e-14
Number of ATMs per 1,000 km2 does not look normally distributed (reject H0)
Shapiro-Wilk Test for Number of ATMs per 100,000 adults: Statistics=0.7744844795531796, p=8.384681043882496e-13
Number of ATMs per 100,000 adults does not look normally distributed (reject H0)
Shapiro-Wilk Test for Financial institution account (% age 15+): Statistics=0.9536215481973493, p=0.00023092381576896663
Financial institution account (% age 15+) does not look normally distributed (reject H0)
Shapiro-Wilk Test for Used a debit or credit card (% age 15+): Statistics=0.8892153626857013, p=2.3758739189795696e-08
Used a debit or credit card (% age 15+) does not look normally distributed (reject H0)
Shapiro-Wilk Test for Sent domestic remittances (% age 15+): Statistics=0.973350060051798, p=0.012039704893561775
Sent domestic remittances (% age 15+) does not look normally distributed (reject H0)
Shapiro-Wilk Test for Has an outstanding housing loan (% age 15+): Statistics=0.9446835638292622, p=4.8429486590942235e-05
Has an outstanding housing loan (% age 15+) does not look normally distributed (reject H0)
Shapiro-Wilk Test for Made a digital payment (% age 15+): Statistics=0.9519171760133196, p=0.00016975051673975833
Made a digital payment (% age 15+) does not look normally distributed (reject H0)
Shapiro-Wilk Test for Received wages (% age 15+): Statistics=0.9482073254271527, p=8.831485298755094e-05
Received wages (% age 15+) does not look normally distributed (reject H0)
Shapiro-Wilk Test for Received government transfer (% age 15+): Statistics=0.9056037724684276, p=1.6954967930706488e-07
```


Shapiro-Wilk test reveals that the distribution of the data is not normal. Robust principal component analysis is designed to handle outliers and data that are not normally distributed (Candes *et al*., 2011).

5.1 Principal Component Analysis for feature selection

- Identify important features (loadings)

```
# Apply PCA
pca = PCA(n_components=min(numeric_data_scaled.shape[0], numeric_data_scaled.shape[1]))
pca.fit(numeric_data_scaled)

# Evaluate explained variance
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

# Plot explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance, marker='o', linestyle='--', color='b')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by Principal Components')
plt.grid(True)
plt.show()
```

- Plot the loadings for the first few components
- Select top features based on their loadings in the first few components. Output looks like;

```
Top features based on their loadings in the first few principal components:
Has an outstanding housing loan (% age 15+)                                0.280455
Number of ATMs per 1,000 km2                                               0.240981
Number of debit cards                                                      0.238749
Number of commercial bank branches per 1,000 km2                           0.235814
Sent or received domestic remittances (% age 15+)                          0.232731
Sent domestic remittances (% age 15+)                                      0.213082
Number of registered mobile money agent outlets                            0.208284
Received government transfer (% age 15+)                                   0.179103
No account because financial institutions are too far away (% age 15+)     0.177531
Made a digital payment (% age 15+)                                         0.176975
dtype: float64
```

- Check for data validity using Kaiser-Meyer-Olkin and Bartlett test. Colab uses factor analyzer to do that.

```
[ ]  from factor_analyzer import FactorAnalyzer
     from factor_analyzer.factor_analyzer import calculate_kmo, calculate_bartlett_sphericity

     # Perform the KMO test
     kmo_all, kmo_model = calculate_kmo(numeric_data_scaled)
     print(f'KMO Model: {kmo_model}')

     # Perform Bartlett's test of sphericity
     chi_square_value, p_value = calculate_bartlett_sphericity(numeric_data_scaled)
     print(f'Bartlett\'s Test: Chi-square={chi_square_value}, p-value={p_value}')
```

5.2 Stage one – robust principal component analysis

- Define the function using rPCA for each dimension in stage one.

```
# Select columns that make up the access dimension
access_columns = [2, 3, 4, 5, 6]  # Adjust based on your actual data
access_data = complete_data_dropped.iloc[:, access_columns]

# Standardize the data
scaler = StandardScaler()
access_data_scaled = scaler.fit_transform(access_data)

# Function to perform robust PCA
def robust_pca(X, n_components):
    U, s, Vt = np.linalg.svd(X, full_matrices=False)
    S = np.diag(s)
```

```python
# Function to perform robust PCA
def robust_pca(X, n_components):
    U, s, Vt = np.linalg.svd(X, full_matrices=False)
    S = np.diag(s)
    L = U[:, :n_components] @ S[:n_components, :n_components] @ Vt[:n_components, :]
    S_sparse = X - L
    return L, S_sparse

# Perform robust PCA
n_components = access_data.shape[1]  # Number of principal components to extract
L, S_sparse = robust_pca(access_data_scaled, n_components)

# Obtain the principal components (scores)
U, Sigma, VT = np.linalg.svd(L, full_matrices=False)
scores = U @ np.diag(Sigma)

# Print summary
explained_variance = Sigma ** 2 / (access_data_scaled.shape[0] - 1)
print("Eigenvalues (Explained Variance):", explained_variance)
print("Principal Components (Scores):")
print(scores)
```

- Repeat this for usage and quality dimensions by defining what makes up the component of each
- Obtain the eigenvalues and principal component output
- Weight the outputs based on each dimension; the exact formula is replicated in python as below.

```python
explained_variance = Sigma ** 2 / (data_scaled.shape[0] - 1)

# Calculate subindices for each component
subindices = np.zeros(data_scaled.shape[0])
for i in range(data_scaled.shape[0]):
    subindex = np.sum(explained_variance[:n_components] * scores[i, :n_components]) / np.sum(explained_vari
    subindices[i] = subindex
```

5.3 Second stage principal component analysis

- Define the function for stage two

```python
# Create DataFrame with the subindices
dimensions = pd.DataFrame({
    'Access': subindices_access,
    'Usage': subindices_usage,
    'Quality': subindices_quality
})

# Extract the variables for PCA
pca_data = dimensions[['Access', 'Usage', 'Quality']]

# Perform PCA
pca = PCA()
pca.fit(pca_data)

# Obtain the eigenvalues (explained variance)
variance_final = pca.explained_variance_

# Obtain the loadings (eigenvectors)
vec = pca.components_.T

# Display eigenvalues and loadings
print("Eigenvalues (Explained Variance):")
print(variance_final)
```

- Apply PCA to estimate the Eigenvectors and PCA of subindices

```
# Calculate principal components
for i in range(3):  # For the first three principal components
    dimensionss[f'PC{i+1}'] = (
        (vec[0, i] * dimensionss['Access']) +
        (vec[1, i] * dimensionss['Usage']) +
        (vec[2, i] * dimensionss['Quality'])
    )

# Display the DataFrame with principal components
print(dimensionss)

# Optional: Save eigenvalues, eigenvectors, and principal components to CSV
eigenvalues_df = pd.DataFrame(variance_final, columns=['Eigenvalue'])
eigenvectors_df = pd.DataFrame(vec, index=['Access', 'Usage', 'Quality'], columns=[f'PC{i+1}' for i in range(vec.shape[1])])
dimensionss.to_csv('dimensionss_with_pcs.csv', index=False)
eigenvalues_df.to_csv('eigenvalues.csv', index=False)
eigenvectors_df.to_csv('eigenvectors.csv', index=True)

print("Eigenvalues, eigenvectors, and dimensions with principal components have been saved to 'eigenvalues.csv', 'eigenvectors.csv', and 'dimensions_
```

- Estimate the inclusion index using the weighted average of the output above

```
[ ]  # Calculate the Financial Inclusion Index (FI) for each country and year
     dimensionss['FI'] = (
         (variance_final[0] * dimensionss['PC1']) +
         (variance_final[1] * dimensionss['PC2']) +
         (variance_final[2] * dimensionss['PC3'])
     ) / sum(variance_final)
```

- Categorize the level of financial inclusion

```
# Define the function to categorize the scores based on the provided criteria
def categorize(score):
    if score < 0:
        return 'Very low'
    elif 0 <= score < 0.3:
        return 'Low'
    elif 0.3 <= score <= 0.6:
        return 'Medium'
    elif 0.6 < score <= 1:
        return 'High'
    else:
        return 'Very high'

# Apply the categorization function to the Access, Usage, Quality, and FI columns
dimensionss['Access_Rank'] = dimensionss['Access'].apply(categorize)
dimensionss['Usage_Rank'] = dimensionss['Usage'].apply(categorize)
dimensionss['Quality_Rank'] = dimensionss['Quality'].apply(categorize)
dimensionss['FI_Rank'] = dimensionss['FI'].apply(categorize)

# Display the updated DataFrame with ranks and FI ranking
print(dimensionss[['Country name', 'Year', 'Access', 'Access_Rank', 'Usage', 'Usage_Rank', 'Quality', 'Quality_Rank', 'FI', 'FI_Rank']])
```

- Plot a correlation matrix of the dimensions to visualize the relationships between access, usage and quality
- Filter top and bottom countries based on FI
- Display results of top and bottom countries with high FI

References

Candès, E.J., Li, X., Ma, Y. and Wright, J. (2011) 'Robust Principal Component Analysis', *Journal of the ACM (JACM)*, 58(3), Article 11, pp. 1-37. Available at: https://doi.org/10.1145/1970392.1970395 (Accessed: 12 August 2024).