Finacial_Inclusion

March 21, 2025

[90]:
```python
import pandas as pd

# Load the data
pakistan_data = pd.read_csv("micro_pak.csv")  # replace with your actual file␣
 ↪name

# Check first few rows
pakistan_data.head()
```

[90]:
```
    economy economycode  wpid_random       wgt  female   age  educ  inc_q  \
0  Pakistan         PAK    163613042  1.102319       2  29.0     2      4
1  Pakistan         PAK    113549828  0.394235       1  40.0     1      3
2  Pakistan         PAK    185362095  0.509578       1  20.0     2      3
3  Pakistan         PAK    136416803  1.032916       1  30.0     1      1
4  Pakistan         PAK    112485323  1.389640       2  55.0     2      5

   emp_in  urbanicity_f2f  …  receive_wages  receive_transfers  \
0       1               2  …              1                  4
1       2               2  …              5                  4
2       2               2  …              4                  4
3       1               1  …              3                  4
4       2               1  …              4                  4

   receive_pension  receive_agriculture  pay_utilities  remittances  \
0                4                    4              2            5
1                4                    4              4            5
2                4                    4              4            5
3                4                    4              2            5
4                4                    2              2            5

   mobileowner  internetaccess  anydigpayment  merchantpay_dig
0            1               1              1                0
1            2               2              0                0
2            1               1              0                0
3            1               2              0                0
4            1               1              0                0

[5 rows x 114 columns]
```

[92]:
```python
# Check column names and data types
pakistan_data.info()

# Check for missing values
pakistan_data.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1002 entries, 0 to 1001
Columns: 114 entries, economy to merchantpay_dig
dtypes: float64(67), int64(45), object(2)
memory usage: 892.5+ KB
```

[92]:
```
economy            0
economycode        0
wpid_random        0
wgt                0
female             0
                  ..
remittances        0
mobileowner        0
internetaccess     0
anydigpayment      0
merchantpay_dig    0
Length: 114, dtype: int64
```

0.1 1. Remove Duplicates

[94]:
```python
# Check duplicates
duplicates = pakistan_data.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# If any duplicates found, remove them
pakistan_data = pakistan_data.drop_duplicates()
```

```
Number of duplicate rows: 0
```

0.2 2. Handle Missing Values

[96]:
```python
# Check missing values
missing_values = pakistan_data.isnull().sum()
missing_values = missing_values[missing_values > 0].sort_values(ascending=False)
print(missing_values)
```

```
fin14c_2_China    1002
fin14_2_China     1002
fin31b1_China     1002
fin45_1_China     1002
fin8a              998
                  …
```

```
fin11d             145
fin11e             145
fin11f             145
fin45              114
age                  1
Length: 66, dtype: int64
```

0.3 2.1 — Columns with 100% missing values (like fin14c_2_China, etc.)

0.3.1 These columns are completely empty. So we can safely drop them:

[98]:
```
# Drop columns with all missing values
cols_to_drop = missing_values[missing_values == pakistan_data.shape[0]].index
pakistan_data = pakistan_data.drop(columns=cols_to_drop)
print(f"Dropped columns with 100% missing values: {list(cols_to_drop)}")
```

```
Dropped columns with 100% missing values: ['fin14c_2_China', 'fin14_2_China',
'fin31b1_China', 'fin45_1_China']
```

0.4 2.2 — Columns with partial missing values

0.4.1 After dropping completely empty columns, re-check missing values:

[100]:
```
missing_values = pakistan_data.isnull().sum()
missing_values = missing_values[missing_values > 0].sort_values(ascending=False)
print(missing_values)
```

```
fin8a       998
fin8        997
fin8b       997
fin14c_2    992
fin14c      992
           …
fin11h      145
fin11b      145
fin11d      145
fin45       114
age           1
Length: 62, dtype: int64
```

0.4.2 Observation:

Some columns have almost all values missing (e.g., fin8a has 998 missing out of 1002 rows — that's 99% missing!).

These extremely sparse columns are not helpful and can be dropped.

Columns with less than around 30% missing can be filled.

age has just 1 missing — we can easily fill it with the median.

0.4.3 2.2.1 — Droping columns with more than 80% missing values:

[102]:
```
# Drop columns with more than 80% missing values
threshold = 0.8
cols_to_drop_sparse = missing_values[missing_values > threshold * pakistan_data.
 ↪shape[0]].index
pakistan_data = pakistan_data.drop(columns=cols_to_drop_sparse)
print(f"Dropped very sparse columns: {list(cols_to_drop_sparse)}")
```

```
Dropped very sparse columns: ['fin8a', 'fin8', 'fin8b', 'fin14c_2', 'fin14c',
'fin14_2', 'fin43e', 'fin39e', 'fin4a', 'fin27c1', 'fin27c2', 'fin34e',
'fin39d', 'fin35', 'fin29c1', 'fin29c2', 'fin39b', 'fin39a', 'fin22c',
'fin31b1', 'fin13a', 'fin13b', 'fin13c', 'fin9a', 'fin13d', 'fin27_1',
'fin29_1', 'fin10a', 'fin4', 'fin17a1', 'fin43d', 'fin1_1a', 'fin1_1b', 'fin6',
'fin7', 'fin9', 'fin10', 'fin10b', 'fin5', 'fin43b', 'fin43a', 'fin42a',
'fin34d']
```

0.4.4 2.2.2 — Fill remaining missing values:

[104]:
```
# Identify numeric columns
numeric_cols = pakistan_data.select_dtypes(include=['float64', 'int64']).columns

# Fill missing values in numeric columns with median
pakistan_data[numeric_cols] = pakistan_data[numeric_cols].
 ↪fillna(pakistan_data[numeric_cols].median())
```

[106]:
```
categorical_cols = pakistan_data.select_dtypes(include=['object']).columns
for col in categorical_cols:
    if pakistan_data[col].isnull().sum() > 0:
        pakistan_data[col].fillna(pakistan_data[col].mode()[0], inplace=True)
```

[108]:
```
pakistan_data.isnull().sum().sum()
```

[108]: 0

-

0.4.5 let's move to the "Correct Data Types" step.

We need to make sure every column has the right data type. For example:

Numeric columns should be int or float. Categorical columns (like economy, economycode, etc.) should be object or category. Dates (if any) should be datetime. Step 1: Check current data types *

[110]:
```
pakistan_data.dtypes
```

[110]:
```
economy              object
economycode          object
wpid_random           int64
wgt                 float64
female                int64
                      …
remittances           int64
mobileowner           int64
internetaccess        int64
anydigpayment         int64
merchantpay_dig       int64
Length: 67, dtype: object
```

Checking if numeric columns have text accidentally:

[112]:
```
non_numeric_columns = pakistan_data.select_dtypes(include=['object']).columns
print(non_numeric_columns)
```

```
Index(['economy', 'economycode'], dtype='object')
```

0.5 Now step 4: Standardize formats

[114]:
```
print(pakistan_data['economy'].unique())
print(pakistan_data['economycode'].unique())
```

```
['Pakistan']
['PAK']
```

0.5.1 Check unique values for sanity:

[116]:
```
pakistan_data['economy'] = pakistan_data['economy'].str.strip()
pakistan_data['economycode'] = pakistan_data['economycode'].str.strip()
```

0.5.2 Next Step: Remove Outliers

We'll detect outliers for key numeric columns. Let's start with checking outliers using the interquartile range (IQR) method for a few important numeric columns:

[118]:
```
numeric_cols = pakistan_data.select_dtypes(include=['float64', 'int64']).columns
outlier_counts = {}

for col in numeric_cols:
    Q1 = pakistan_data[col].quantile(0.25)
    Q3 = pakistan_data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = pakistan_data[(pakistan_data[col] < lower_bound) |
 ↪(pakistan_data[col] > upper_bound)]
```

```
    outlier_counts[col] = outliers.shape[0]

# Sort and print columns with most outliers
sorted_outliers = sorted(outlier_counts.items(), key=lambda x: x[1],
 ↪reverse=True)
print(sorted_outliers)
```

```
[('fin11a', 292), ('fin32', 245), ('receive_wages', 245), ('fin11d', 243),
('account', 237), ('fin11c', 223), ('fin11_1', 198), ('anydigpayment', 190),
('account_fin', 189), ('saved', 168), ('remittances', 162), ('fin11g', 154),
('fin42', 148), ('receive_agriculture', 148), ('fin20', 124), ('fin2', 118),
('fin28', 115), ('fin11e', 111), ('fin26', 96), ('account_mob', 90), ('wgt',
83), ('fin17b', 80), ('fin14a1', 78), ('fin16', 75), ('fin14a', 71), ('fin37',
62), ('receive_transfers', 62), ('fin44a', 56), ('fin31a', 47), ('fin24b', 44),
('fin34a', 44), ('fin17a', 41), ('fin31b', 41), ('fin31c', 38), ('fin22a', 36),
('fin44b', 36), ('fin44c', 19), ('fin38', 18), ('receive_pension', 18),
('fin14b', 16), ('fin33', 16), ('merchantpay_dig', 14), ('fin34b', 11),
('fin14_1', 10), ('fin45', 7), ('fin45_1', 7), ('age', 5), ('fin24a', 2),
('fin11b', 1), ('mobileowner', 1), ('internetaccess', 1), ('wpid_random', 0),
('female', 0), ('educ', 0), ('inc_q', 0), ('emp_in', 0), ('urbanicity_f2f', 0),
('fin11f', 0), ('fin11h', 0), ('fin22b', 0), ('fin24', 0), ('fin30', 0),
('fin44d', 0), ('borrowed', 0), ('pay_utilities', 0)]
```

0.5.3 Interpretation of Results:

I've identified columns with the highest number of outliers: Top columns with large outliers:

fin11a (292 outliers)

fin32 (245 outliers)

receive_wages (245 outliers)

fin11d (243 outliers)

… and so on. ###### What to do next? #### We have two options:

1. Remove outliers

For columns where outliers are clearly data entry mistakes or extremely unrealistic values.

2. Cap/fix outliers (recommended for financial/survey data)

Replacing extreme outliers with the upper and lower bounds (Winsorization) to avoid losing data.

0.5.4 Let's do capping (safe method):

[120]:
```
for col in numeric_cols:
    Q1 = pakistan_data[col].quantile(0.25)
    Q3 = pakistan_data[col].quantile(0.75)
    IQR = Q3 - Q1
```

```
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    pakistan_data[col] = pakistan_data[col].clip(lower=lower_bound,␣
↪upper=upper_bound)
```

0.5.5 Here's the code for normalization:

We will use Min-Max scaling for all numeric columns:

[122]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
pakistan_data[numeric_cols] = scaler.fit_transform(pakistan_data[numeric_cols])
```

Now let's Validate Data by: Checking that no missing values remain.

Verifying all numeric columns are between 0 and 1.

Confirming categorical columns are still intact.

[124]:
```
# 1. Check for missing values
print("Remaining missing values:\n", pakistan_data.isnull().sum().sum())

# 2. Check numeric column ranges
for col in numeric_cols:
    min_val = pakistan_data[col].min()
    max_val = pakistan_data[col].max()
    if min_val < 0 or max_val > 1:
        print(f" Column '{col}' is out of range: min={min_val}, max={max_val}")
    else:
        print(f" Column '{col}' is properly scaled between {min_val} and␣
↪{max_val}")

# 3. Verify categorical columns
print("\nCategorical columns unique values:")
for col in categorical_cols:
    print(f"{col}: {pakistan_data[col].unique()}")
```

```
Remaining missing values:
 0
 Column 'wpid_random' is properly scaled between 0.0 and 0.9999999999999998
 Column 'wgt' is properly scaled between 0.0 and 1.0
 Column 'female' is properly scaled between 0.0 and 1.0
 Column 'age' is properly scaled between 0.0 and 0.9999999999999999
 Column 'educ' is properly scaled between 0.0 and 1.0
 Column 'inc_q' is properly scaled between 0.0 and 1.0
 Column 'emp_in' is properly scaled between 0.0 and 1.0
 Column 'urbanicity_f2f' is properly scaled between 0.0 and 1.0
```

```
Column 'account' is properly scaled between 0.0 and 0.0
Column 'account_fin' is properly scaled between 0.0 and 0.0
Column 'account_mob' is properly scaled between 0.0 and 0.0
Column 'fin2' is properly scaled between 0.0 and 0.0
Column 'fin11_1' is properly scaled between 0.0 and 0.0
Column 'fin11a' is properly scaled between 0.0 and 0.0
Column 'fin11b' is properly scaled between 0.0 and 1.0
Column 'fin11c' is properly scaled between 0.0 and 0.0
Column 'fin11d' is properly scaled between 0.0 and 0.0
Column 'fin11e' is properly scaled between 0.0 and 0.0
Column 'fin11f' is properly scaled between 0.0 and 1.0
Column 'fin11g' is properly scaled between 0.0 and 0.0
Column 'fin11h' is properly scaled between 0.0 and 1.0
Column 'fin14_1' is properly scaled between 0.0 and 0.0
Column 'fin14a' is properly scaled between 0.0 and 0.0
Column 'fin14a1' is properly scaled between 0.0 and 0.0
Column 'fin14b' is properly scaled between 0.0 and 0.0
Column 'fin16' is properly scaled between 0.0 and 0.0
Column 'fin17a' is properly scaled between 0.0 and 0.0
Column 'fin17b' is properly scaled between 0.0 and 0.0
Column 'fin20' is properly scaled between 0.0 and 0.0
Column 'fin22a' is properly scaled between 0.0 and 0.0
Column 'fin22b' is properly scaled between 0.0 and 1.0
Column 'fin24' is properly scaled between 0.0 and 1.0
Column 'fin24a' is properly scaled between 0.0 and 1.0
Column 'fin24b' is properly scaled between 0.0 and 1.0
Column 'fin26' is properly scaled between 0.0 and 0.0
Column 'fin28' is properly scaled between 0.0 and 0.0
Column 'fin30' is properly scaled between 0.0 and 1.0
Column 'fin31a' is properly scaled between 0.0 and 0.0
Column 'fin31b' is properly scaled between 0.0 and 0.0
Column 'fin31c' is properly scaled between 0.0 and 0.0
Column 'fin32' is properly scaled between 0.0 and 0.0
Column 'fin33' is properly scaled between 0.0 and 0.0
Column 'fin34a' is properly scaled between 0.0 and 0.0
Column 'fin34b' is properly scaled between 0.0 and 0.0
Column 'fin37' is properly scaled between 0.0 and 0.0
Column 'fin38' is properly scaled between 0.0 and 0.0
Column 'fin42' is properly scaled between 0.0 and 0.0
Column 'fin44a' is properly scaled between 0.0 and 1.0
Column 'fin44b' is properly scaled between 0.0 and 1.0
Column 'fin44c' is properly scaled between 0.0 and 1.0
Column 'fin44d' is out of range: min=0.0, max=1.0000000000000002
Column 'fin45' is properly scaled between 0.0 and 0.9999999999999999
Column 'fin45_1' is properly scaled between 0.0 and 1.0
Column 'saved' is properly scaled between 0.0 and 0.0
Column 'borrowed' is properly scaled between 0.0 and 1.0
Column 'receive_wages' is properly scaled between 0.0 and 0.0
```

```
Column 'receive_transfers' is properly scaled between 0.0 and 0.0
Column 'receive_pension' is properly scaled between 0.0 and 0.0
Column 'receive_agriculture' is properly scaled between 0.0 and 0.0
Column 'pay_utilities' is properly scaled between 0.0 and 1.0
Column 'remittances' is properly scaled between 0.0 and 0.0
Column 'mobileowner' is properly scaled between 0.0 and 1.0
Column 'internetaccess' is properly scaled between 0.0 and 1.0
Column 'anydigpayment' is properly scaled between 0.0 and 0.0
Column 'merchantpay_dig' is properly scaled between 0.0 and 0.0
```

```
Categorical columns unique values:
economy: ['Pakistan']
economycode: ['PAK']
```

Data Cleaning Summary – Pakistan Dataset Data filtered for economy = 'Pakistan'. No missing values found. All numerical columns scaled between 0 and 1 (minor floating-point precision issue in fin44d, safe to ignore). Categorical columns (economy, economycode) contain only Pakistan and PAK. Data is clean and ready for analysis or modeling.

[128]:
```
df_pakistan_clean = df_pakistan.copy()
```

[130]:
```
df_pakistan_clean.to_csv("pakistan_cleaned_data.csv", index=False)
print("Cleaned dataset saved successfully.")
```

```
Cleaned dataset saved successfully.
```

[ ]:
```
```