Python Code

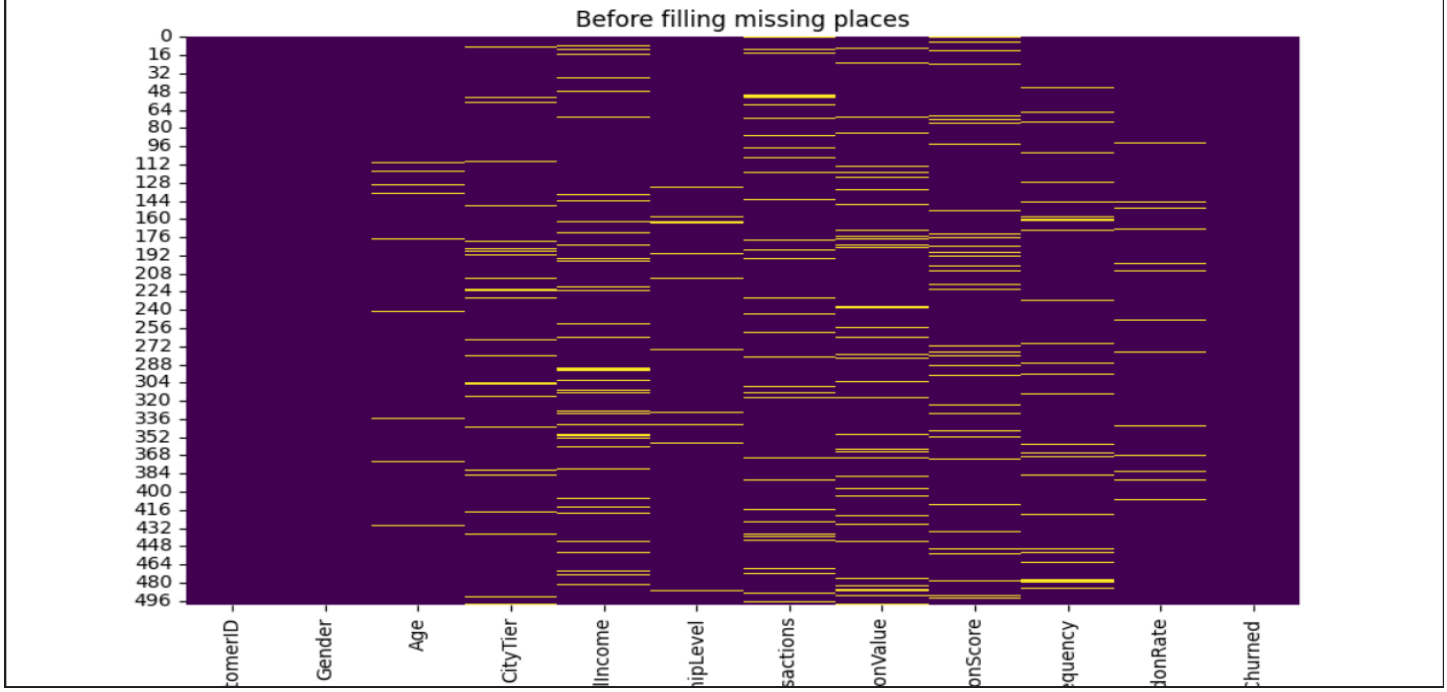
main.py

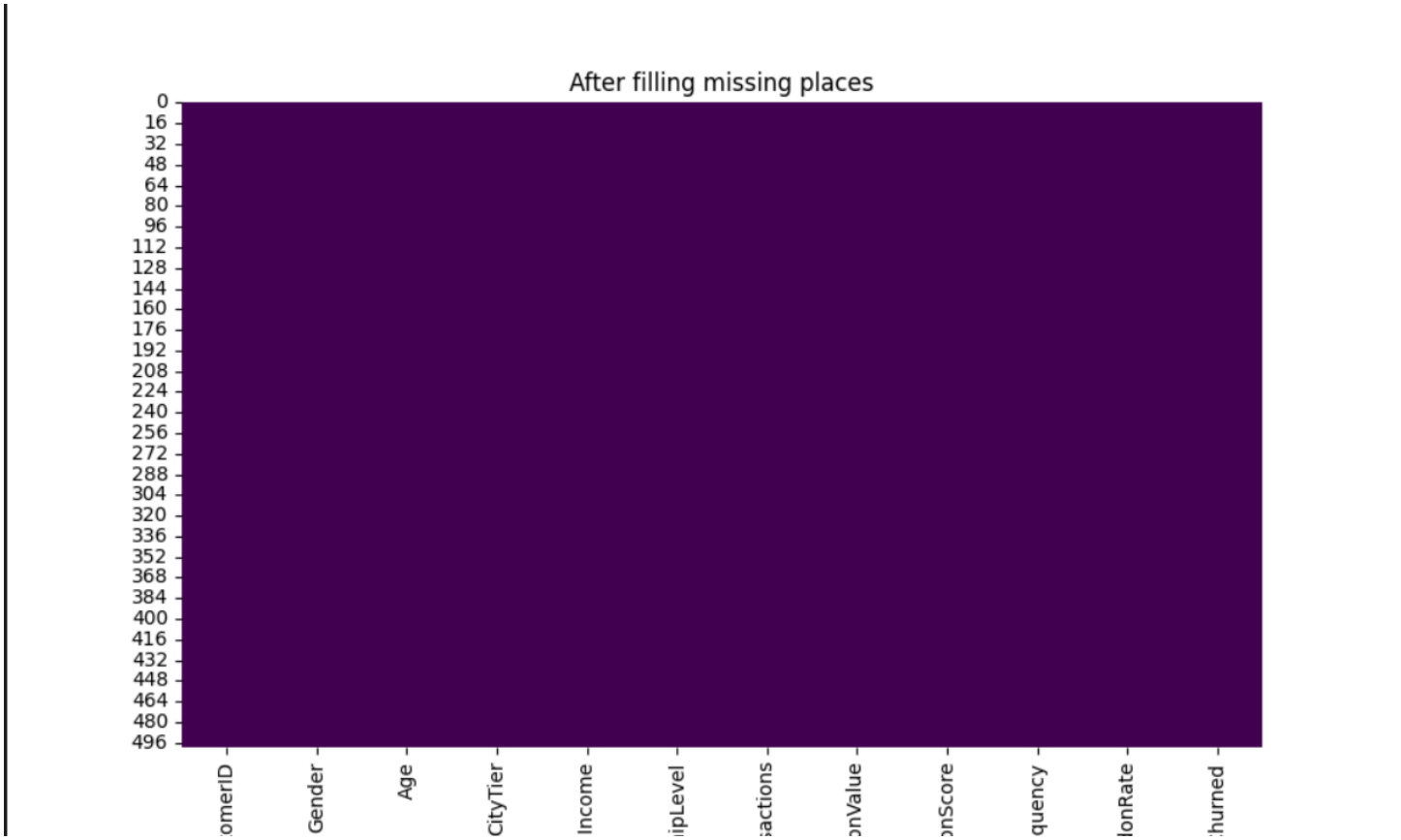
```python
import pandas as pd
from missing_value_handling import handle_missing
from outlier_handling import handle_outliers
from data_normalisation_handling import handle_normalisation
from feature_handling import handle_feature

#Reading csv
df = pd.read_csv('customer_insights_raw.csv')


#calling functions
handle_missing(df.copy())
handle_outliers(df.copy())
handle_normalisation(df.copy())
handle_feature(df.copy())
```

Section 1: Missing Value Handling

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option( pat: 'display.max_columns',  val: None)


def handle_missing(df):  2 usages

    #generating summary of data
    print('Summary of csv data \n', df.info())

    #Finding missing columns
    missing_cols = df.columns[df.isnull().any()]


    #dividing missing columns into numeric ones and category ones
    numeric_col = df[missing_cols].select_dtypes(include=['int64', 'float64']).columns.tolist()
    category_col = df[missing_cols].select_dtypes(include=['object']).columns.tolist()


    #Median and mean dataset for cols
    df_mean = df.copy()
    df_median = df.copy()

    #filling mean values to numeric cols
    for col in numeric_col:
        df_mean[col].fillna(df_mean[col].mean(), inplace=True)

    #filling median values to numeric cols
    for col in numeric_col:
        df_median[col].fillna(df[col].median(), inplace=True)
```

```python
    #filling mode values to category cols
    for col in category_col:
        df_mean[col].fillna(df[col].mode()[0], inplace=True)
        df_median[col].fillna(df[col].mode()[0], inplace=True)

    #Comparing mean and median imputations in missing fields
    print('\nMean imputed dataset \n', df_mean.describe())
    print('\nMedian imputed dataset \n', df_median.describe())


    #dropping rows where more than 2 values are missing in the row
    df_drop_rows = df_median[df.isnull().sum(axis = 1) <= 2]

    #making new csv with cleaned data
    df_drop_rows.to_csv('customer_cleaned_imputed.csv', index=False)

    #ploting heatmap
    plot( title: 'Before filling missing places', df)
    plot( title: 'After filling missing places', df_drop_rows)

#using functions so that we can reuse it
def plot(title, data):  2 usages
    plt.figure(figsize = (10,6))
    sns.heatmap(
        data.isnull(),
        cbar = False,
        cmap = 'viridis'
    )
    plt.title(title)
    plt.show()
```

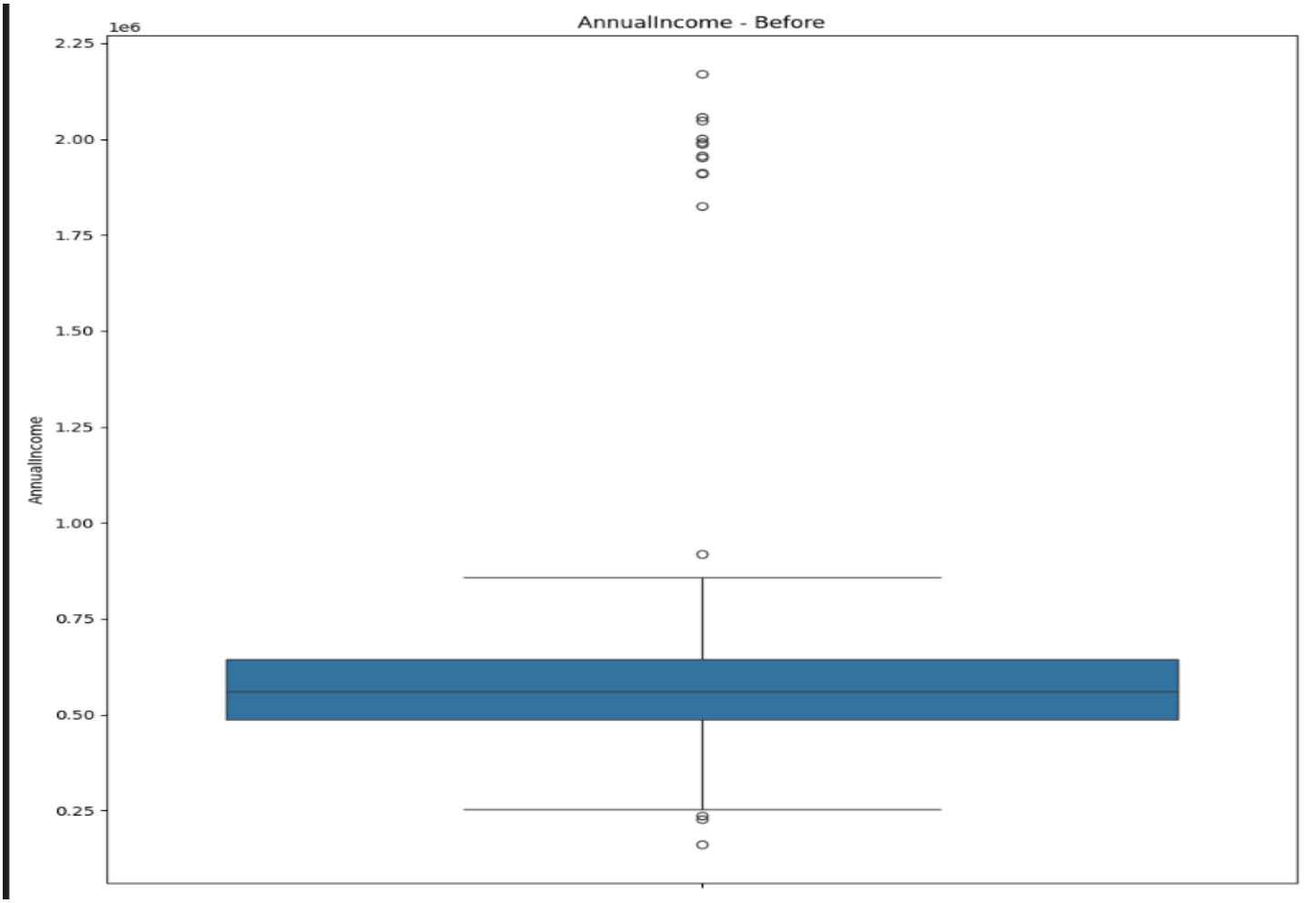

Before filling missing places
0
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
256
272
288
304
320
336
352
368
384
400
416
432
448
464
480
496
Gender
Age
CityTier
Income
Churned

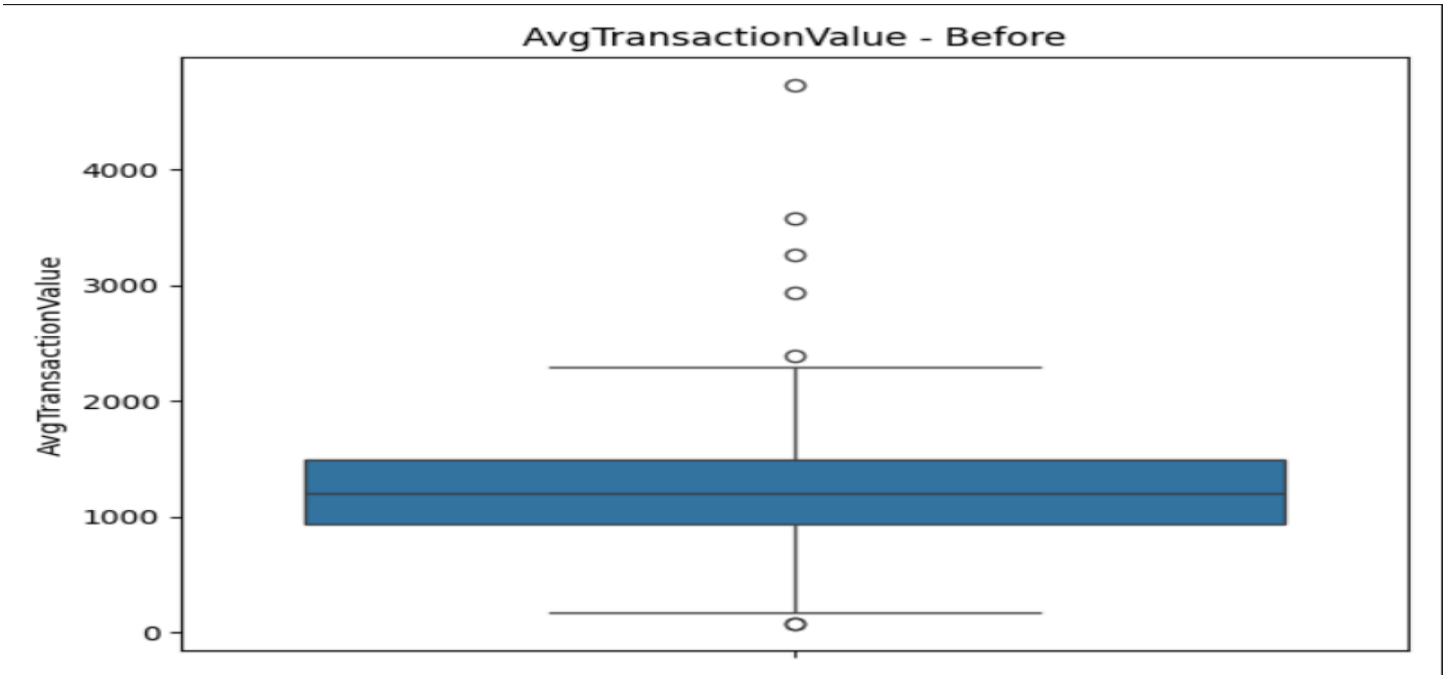

After filling missing places
0
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
256
272
288
304
320
336
352
368
384
400
416
432
448
464
480
496
Gender
Age
CityTier
Income
Churned

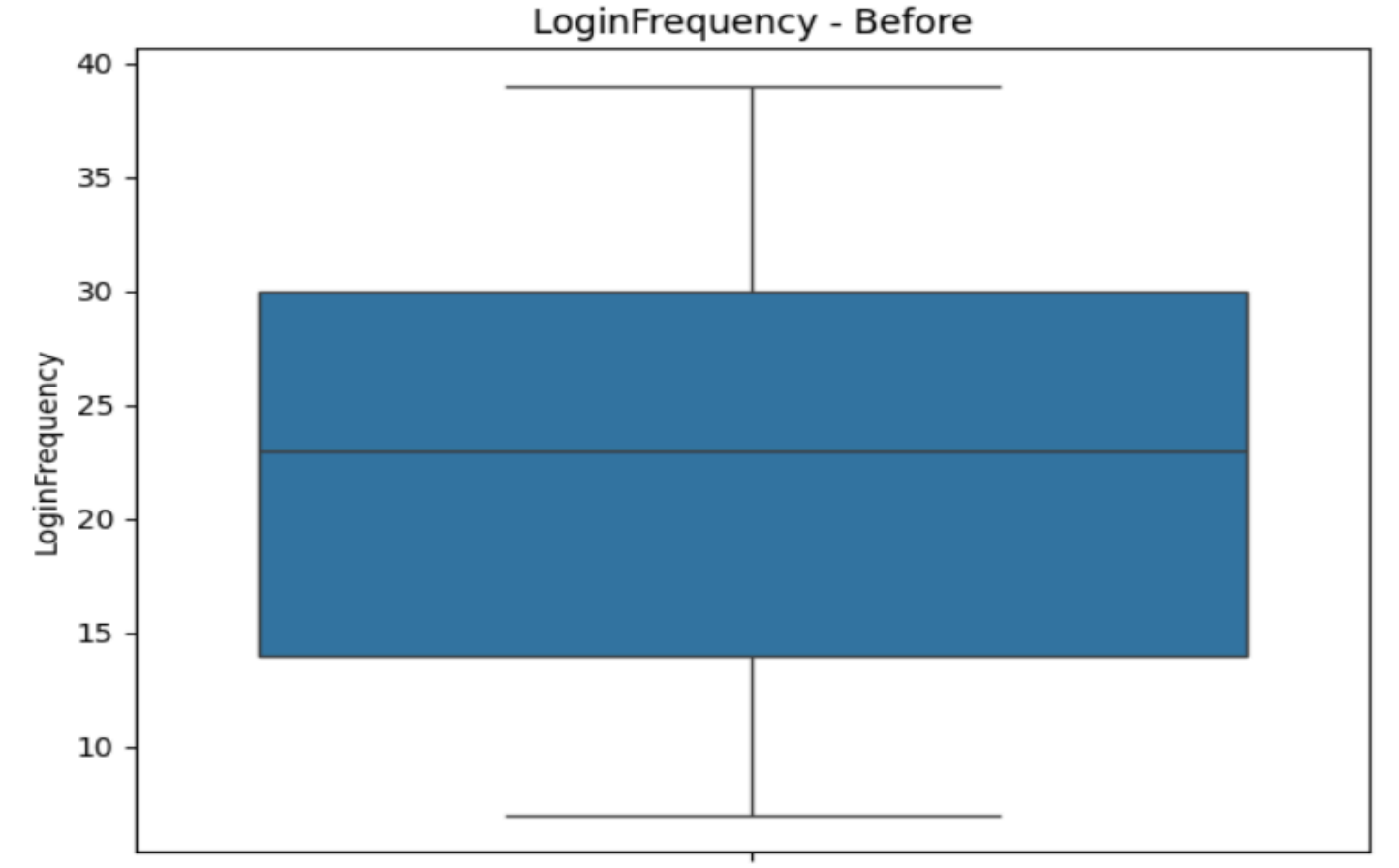
Section 2: Outlier Detection and Handling

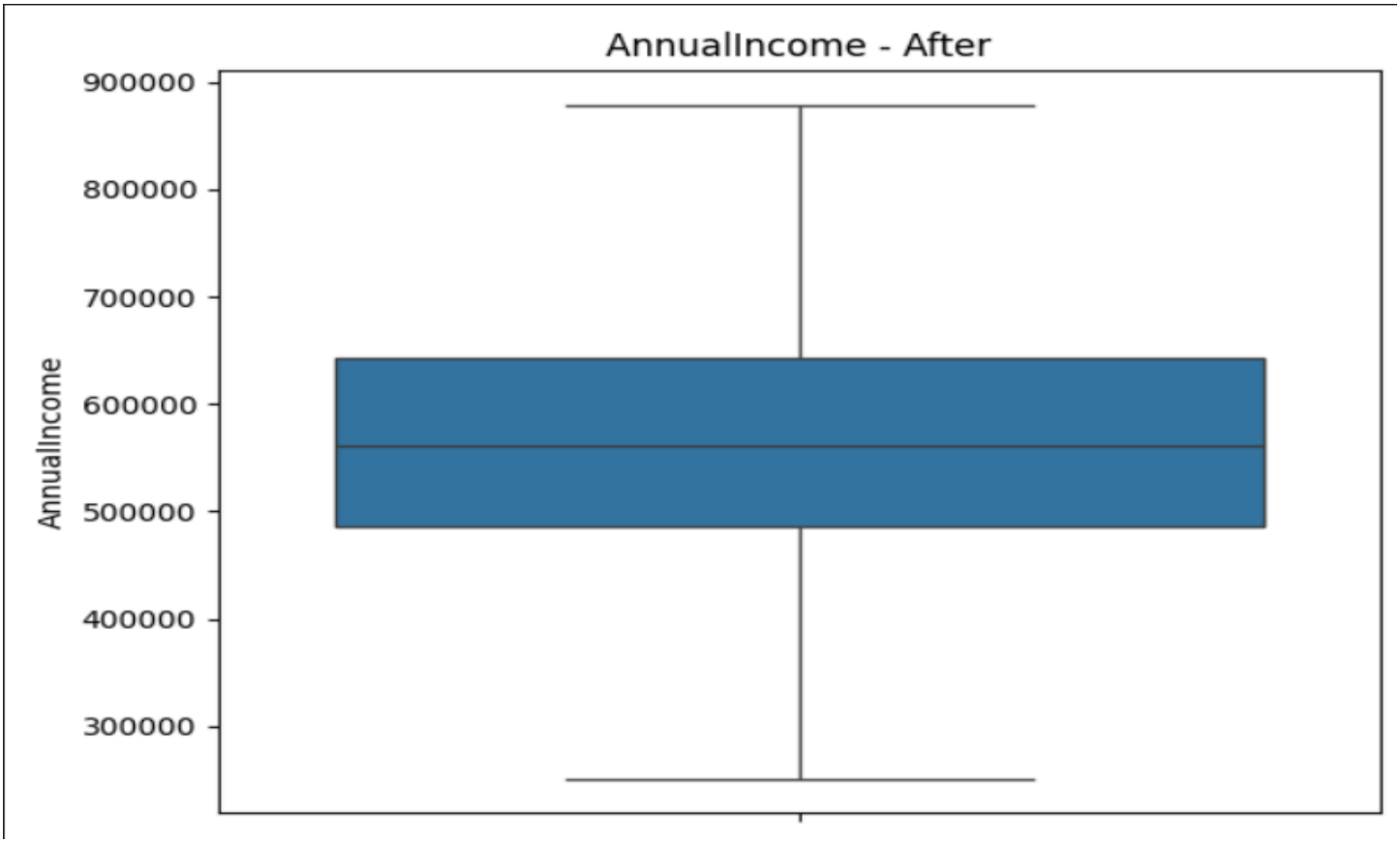
```python
import pandas as pd
from scipy.stats import zscore
import seaborn as sns
import matplotlib.pyplot as plt

def handle_outliers(df):  2 usages

    #taking the numerical cols
    numeric_cols = ['AnnualIncome', 'AvgTransactionValue', 'LoginFrequency']

    #Appling zscore
    z_score = df[numeric_cols].apply(zscore)

    threshold = 3
    outliers_zcore = (z_score.abs() > threshold)
    # print(outliers_zcore.sum())

    data_outliers_zscore = df[outliers_zcore.any(axis=1)]

    print('Outliers in data frame : \n', data_outliers_zscore)

    #Applying IQR

    iqr_bounds = {}
    for col in numeric_cols:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)

        iqr = q3 - q1

        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
```

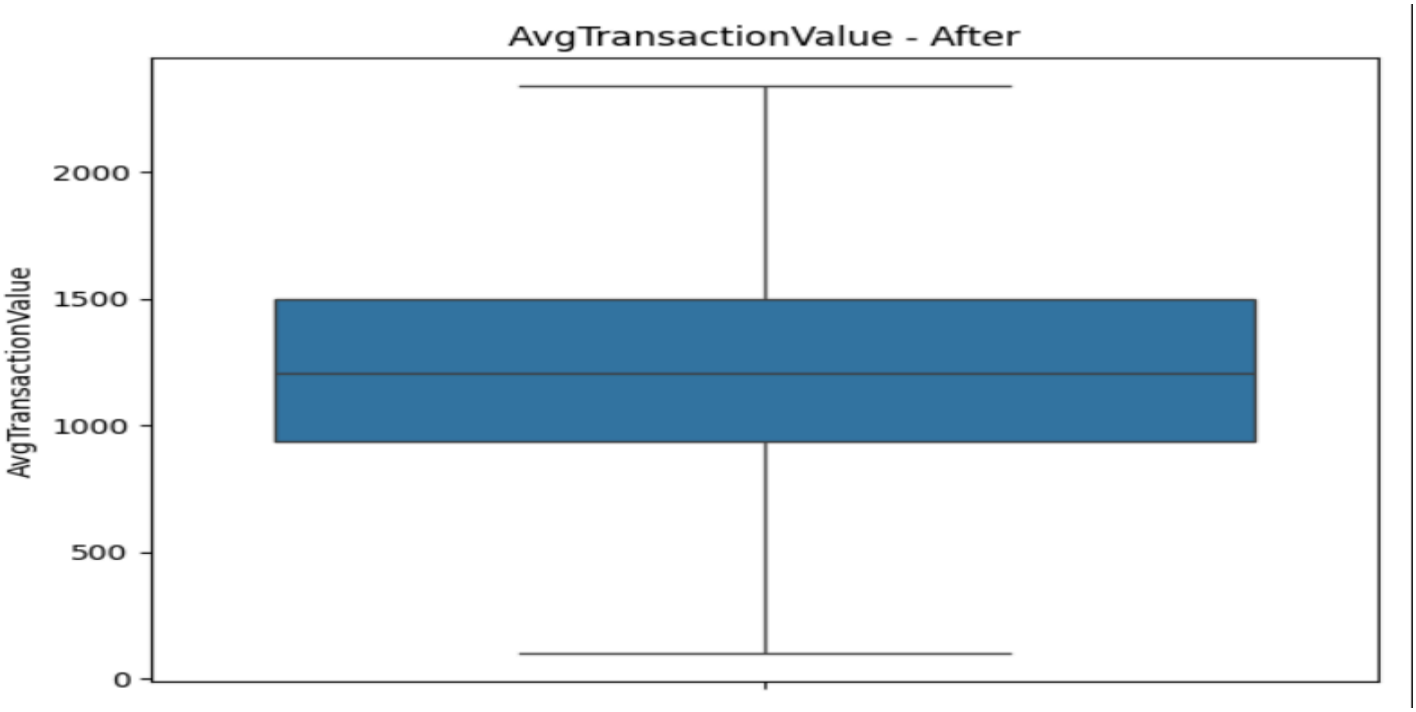
```python
        iqr_bounds[col] = (lower_bound, upper_bound)

        print(f'{col}: {lower_bound:.2f} - {upper_bound:.2f}')


    capping_data = df.copy()

    for col in numeric_cols:
        lower_bound, upper_bound = iqr_bounds[col]
        capping_data[col] = capping_data[col].apply(
            lambda x : lower_bound if x < lower_bound else upper_bound if x > upper_bound else x
        )


    #Before Outlier
    plt.figure(figsize = (10,10))
    for col in numeric_cols:
        plot(df,  title: f'{col} - Before', col)

    #After Outlier
    for col in numeric_cols:
        plot(capping_data,  title: f'{col} - After', col)

def plot(data, title, col):  2 usages
    sns.boxplot(
        data,
        y = col
    )
    plt.title(title)
    plt.tight_layout()
    plt.show()
```

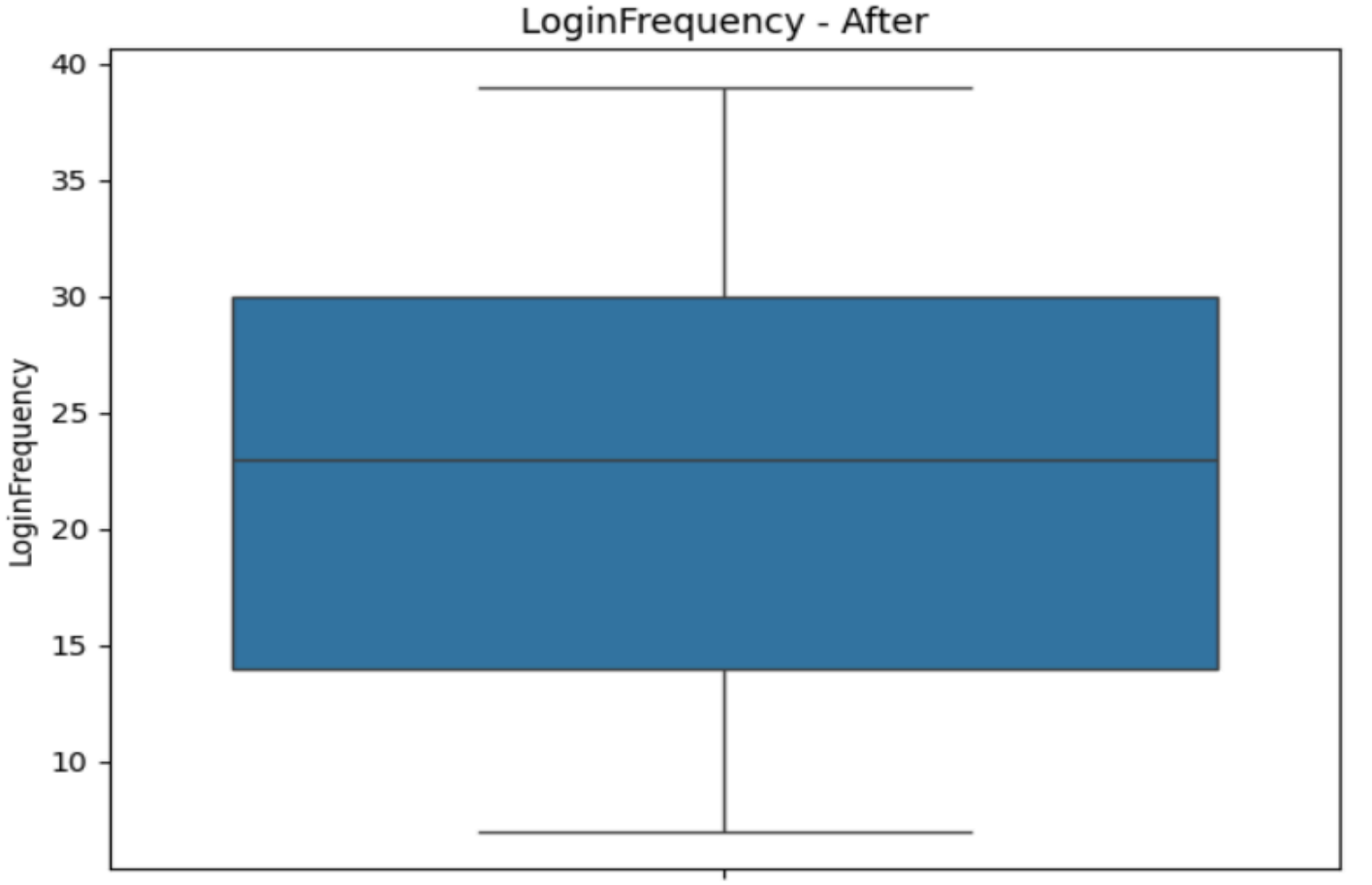

AnnualIncome - Before
1e6
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
AnnualIncome


AvgTransactionValue - Before
4000
3000
2000
1000
0
AvgTransactionValue


LoginFrequency - Before
LoginFrequency
40
35
30
25
20
15
10


AnnualIncome - After
AnnualIncome
900000
800000
700000
600000
500000
400000
300000

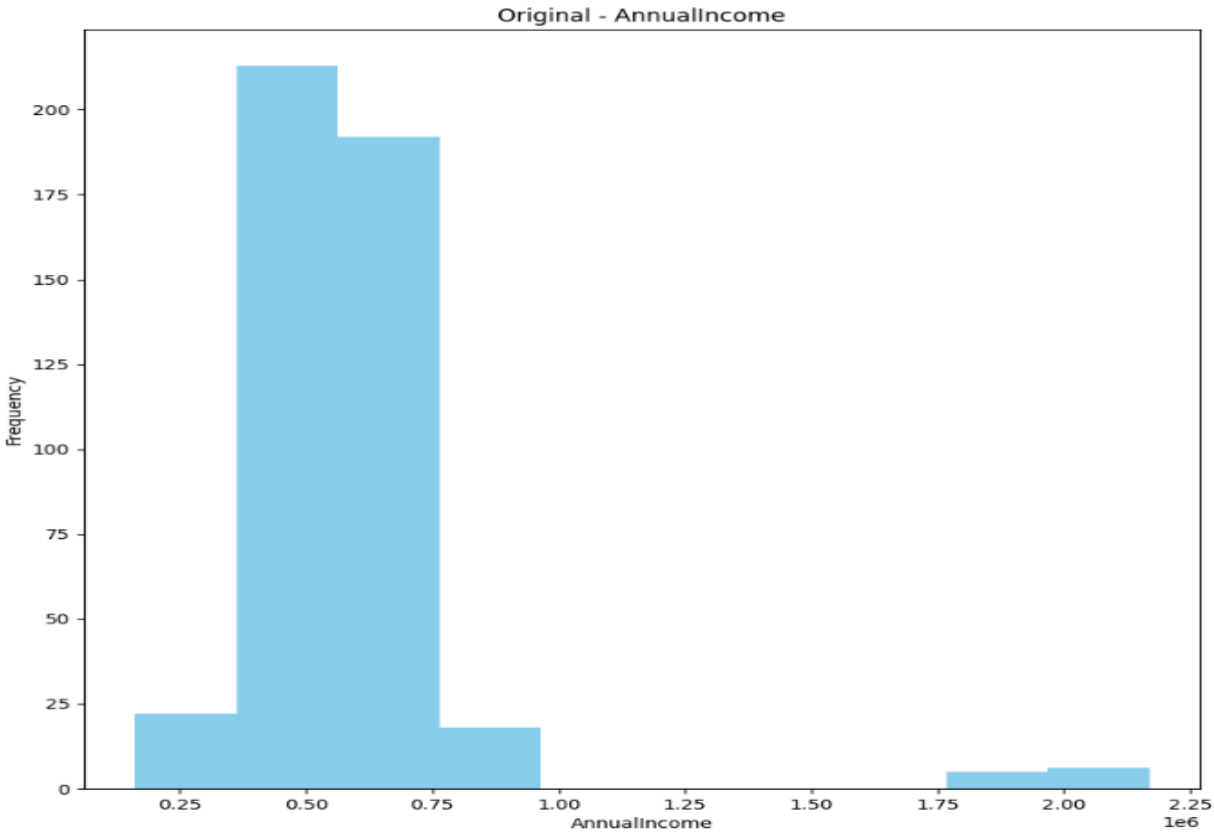

AvgTransactionValue - After
AvgTransactionValue
2000
1500
1000
500
0

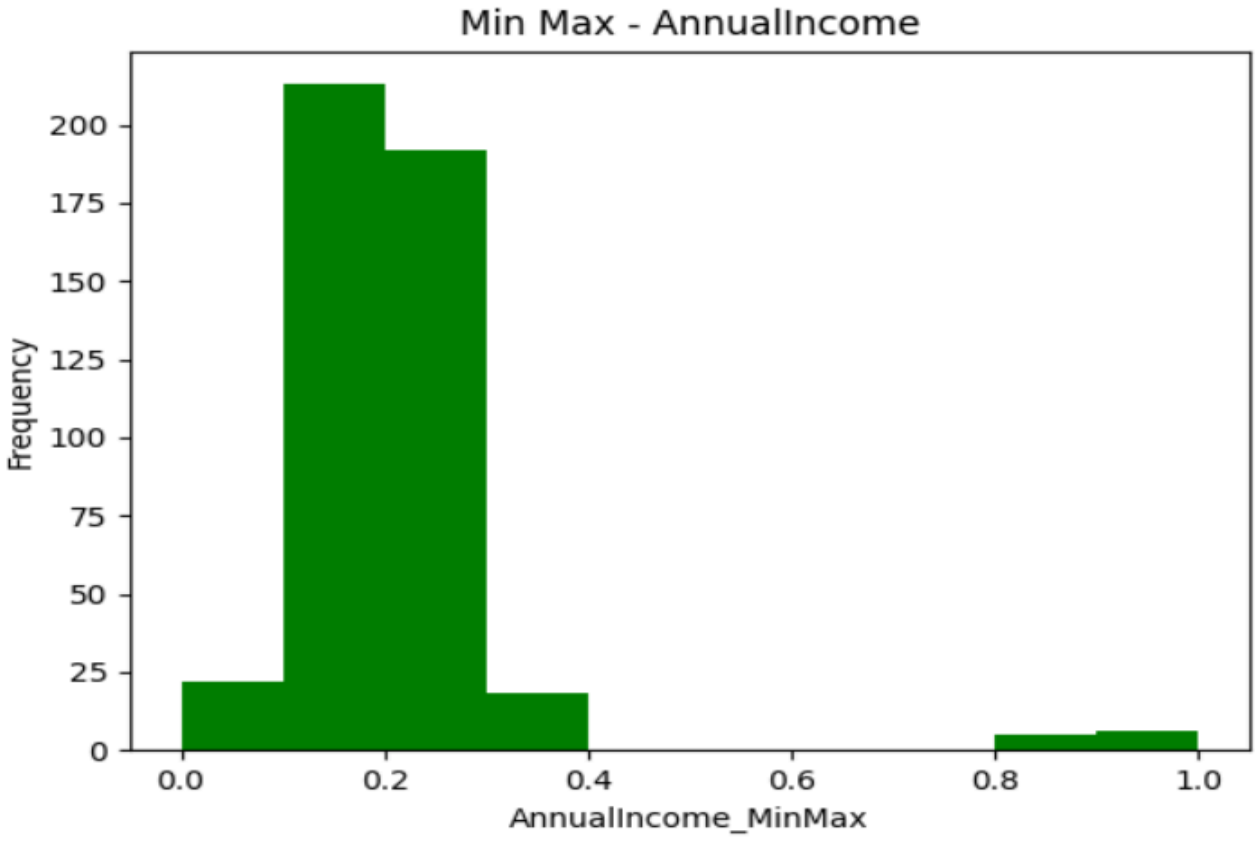

LoginFrequency - After
LoginFrequency
40
35
30
25
20
15
10

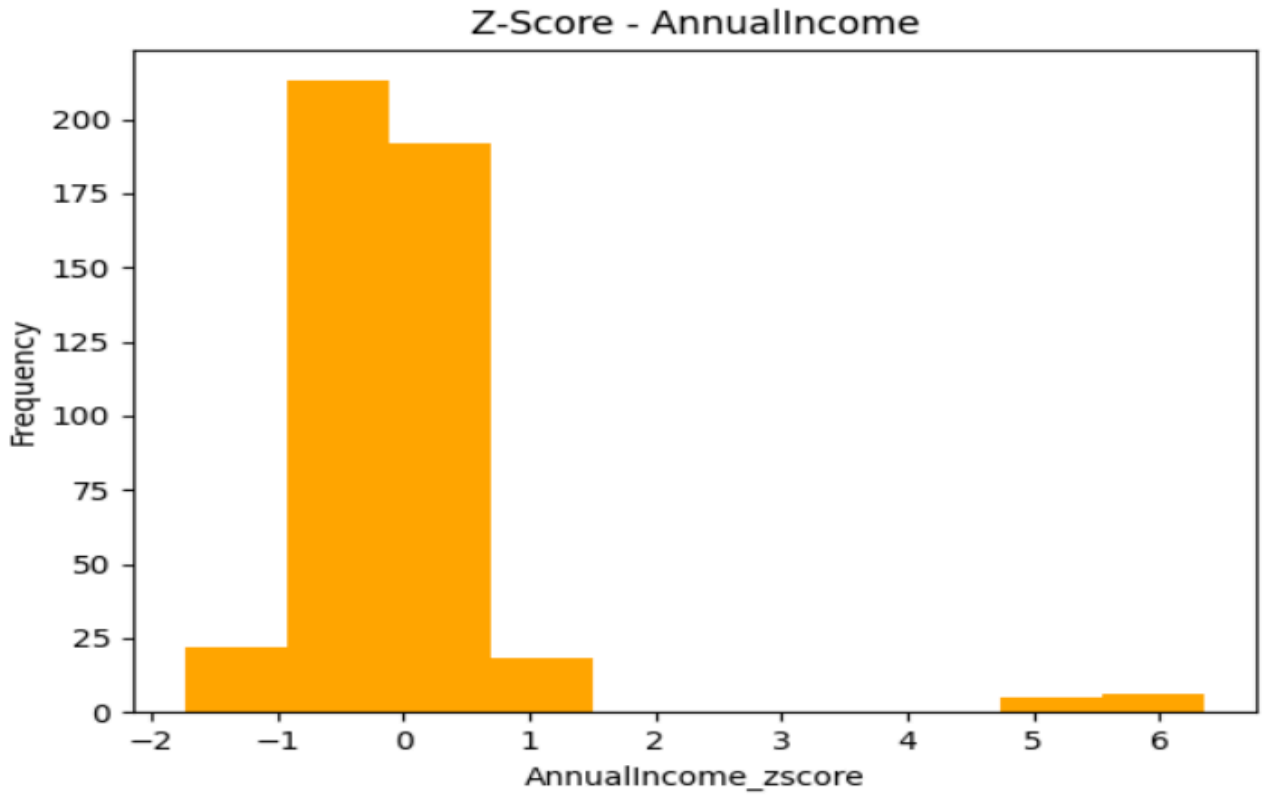
Section 3: Data Normalization

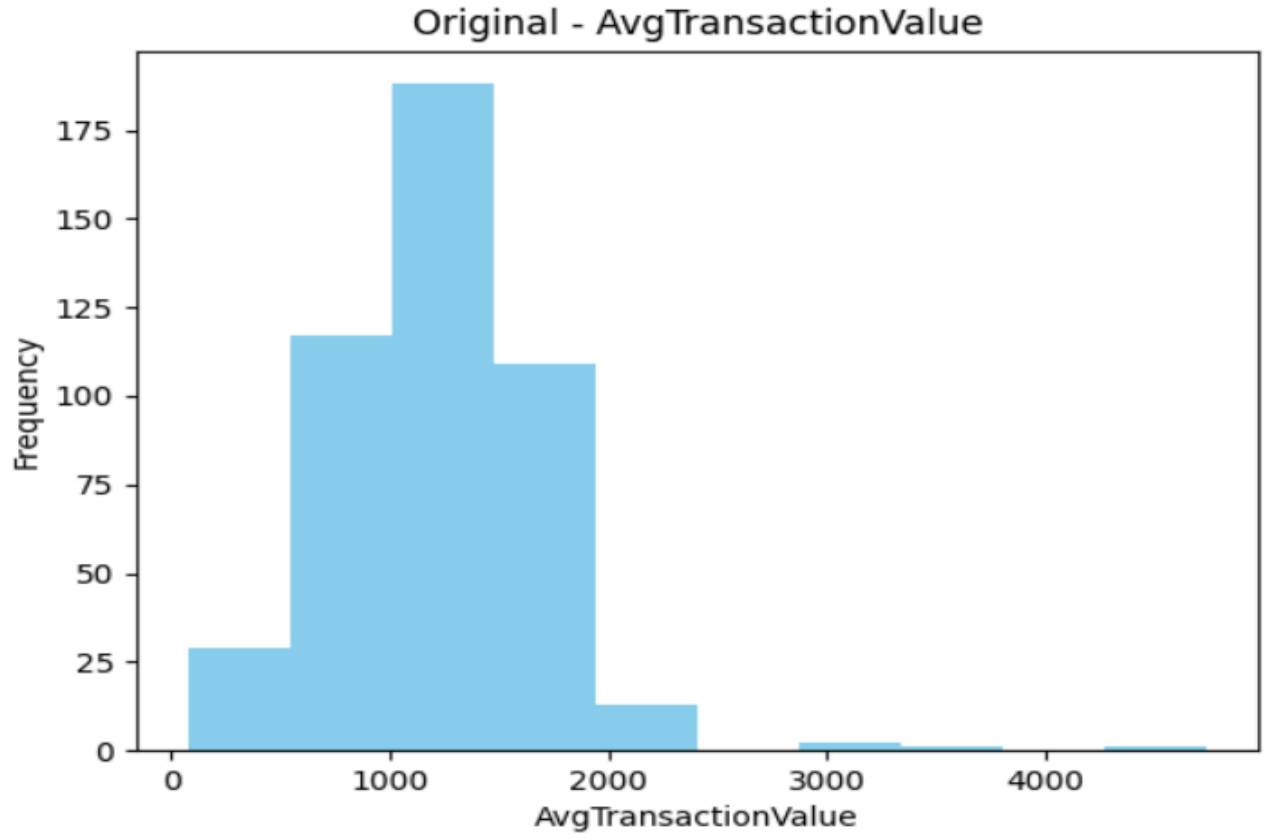
```python
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler

pd.set_option( pat: 'display.max_columns', val: None)

def handle_normalisation(df):  2 usages

    #Selecting the columns for normalisation
    cols = ['AnnualIncome', 'AvgTransactionValue', 'LoginFrequency']


    #Min-Max scaling
    minmax_scaler = MinMaxScaler()
    df_minmax = pd.DataFrame(
        minmax_scaler.fit_transform(df[cols]),
        columns = [f'{col}_MinMax' for col in cols]
    )

    #Z-Score standardisation
    zscore_standardisation = StandardScaler()
    df_zscore = pd.DataFrame(
        zscore_standardisation.fit_transform(df[cols]),
        columns=[f'{col}_zscore' for col in cols]
    )

    #making them into data set to push further to csv files
    df_minmax = pd.concat( objs: [df, df_minmax], axis=1)
    df_zscore = pd.concat( objs: [df, df_zscore], axis=1)

    #plotting histogram
    plt.figure(figsize = (10,10))
```

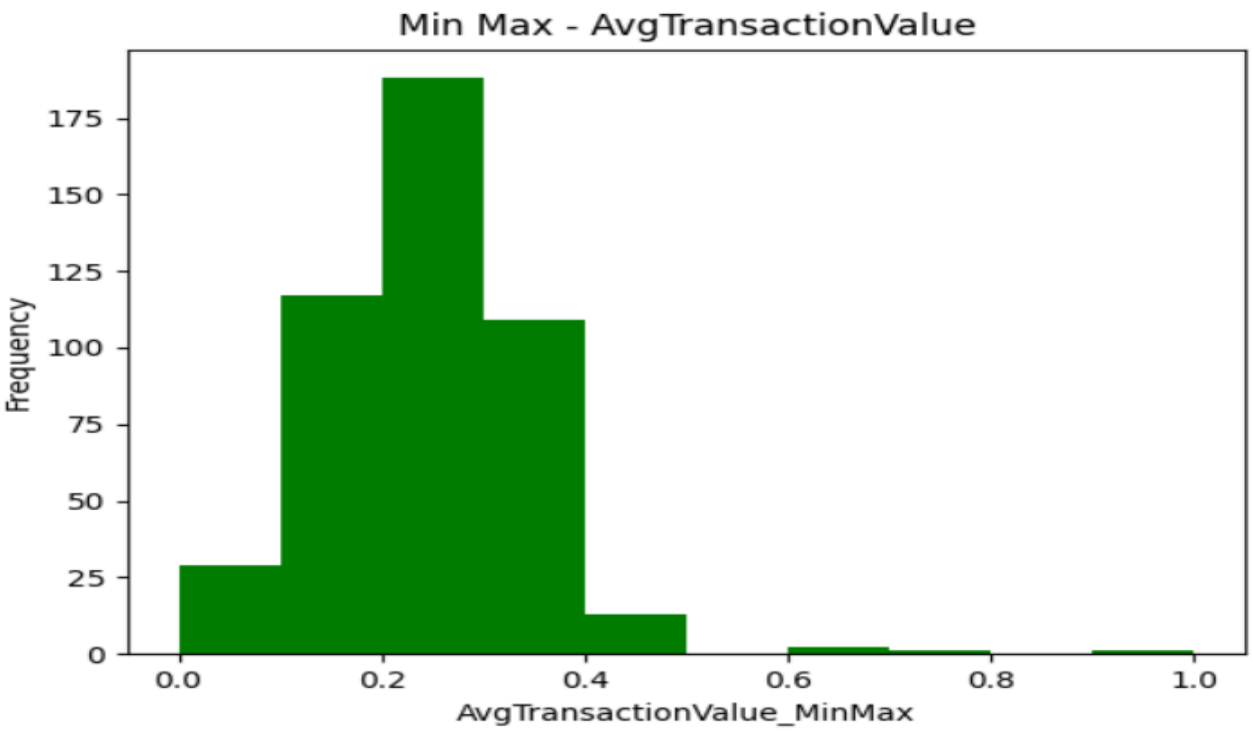
```python
    for col in cols:
        plot(df, col,  title: f'Original - {col}',  clr: 'skyblue')
        plot(df_minmax,  col: f'{col}_MinMax',  title: f'Min Max - {col}',  clr: 'green')
        plot(df_zscore,  col: f'{col}_zscore',  title: f'Z-Score - {col}',  clr: 'orange')


    #Making a new csv with the data frames
    df_minmax.to_csv( path_or_buf: 'customer_minmax_scaled.csv', index = False)
    df_zscore.to_csv( path_or_buf: 'customer_standard_scaled.csv', index = False)

def plot(df, col, title, clr):  3 usages
    plt.hist(
        df[col],
        bins = 10,
        color = clr
    )
    plt.title(title)
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```


Original - AnnualIncome
Frequency
AnnualIncome
1e6
0
25
50
75
100
125
150
175
200
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25

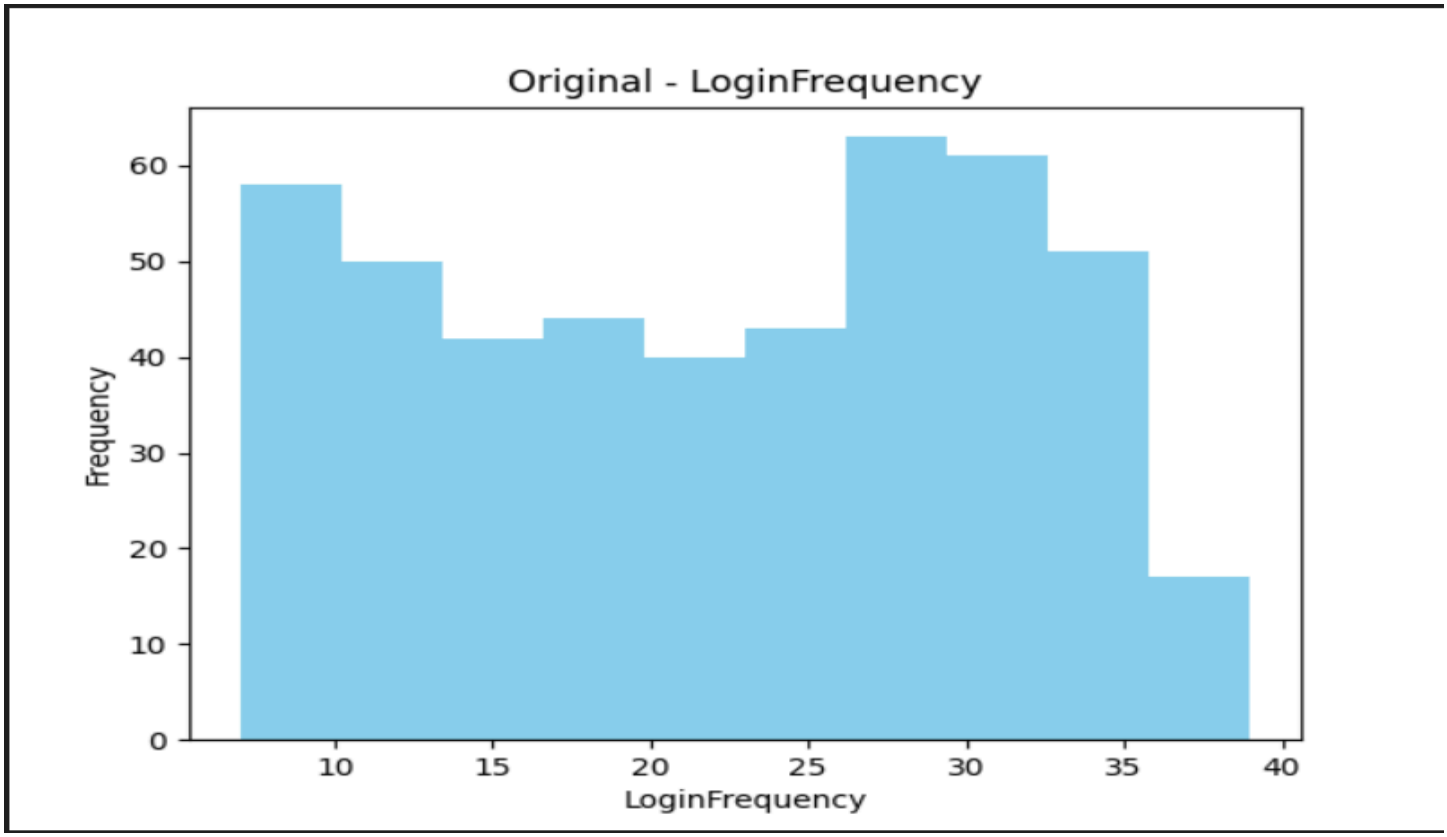

Min Max - AnnualIncome
Frequency
AnnualIncome_MinMax
0
25
50
75
100
125
150
175
200
0.0
0.2
0.4
0.6
0.8
1.0

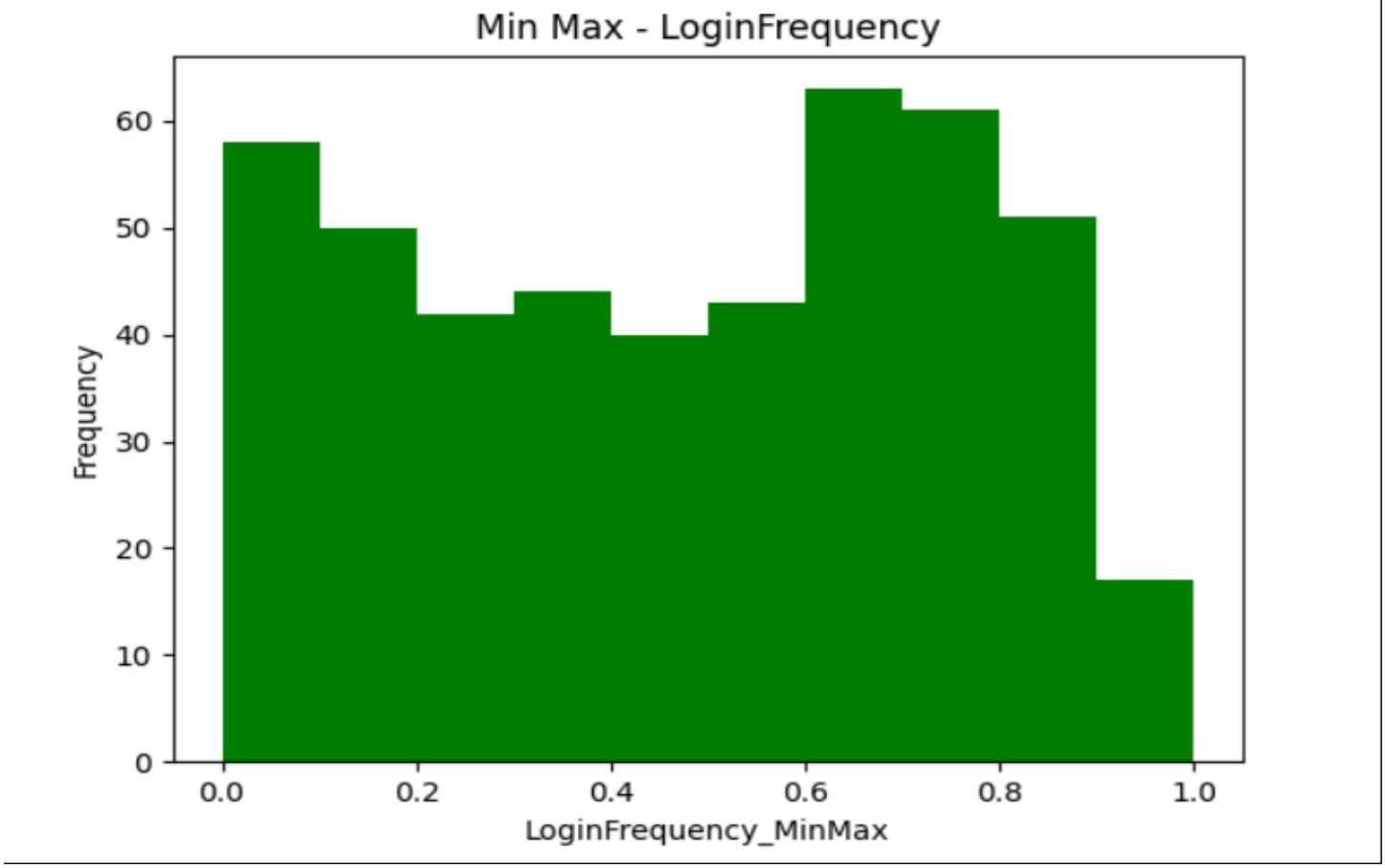

Z-Score - AnnualIncome
Frequency
200
175
150
125
100
75
50
25
0
−2
−1
0
1
2
3
4
5
6
AnnualIncome_zscore

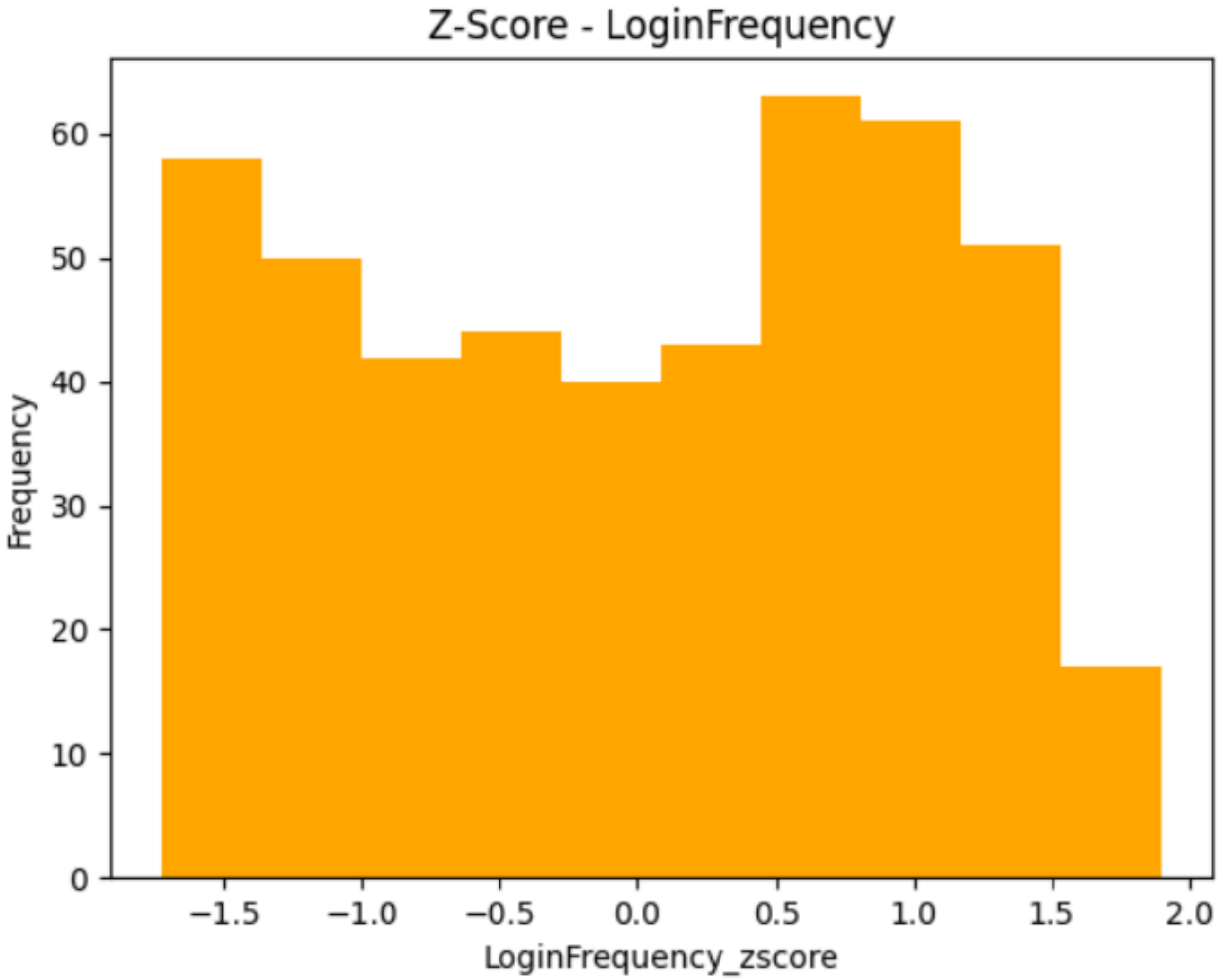

Original - AvgTransactionValue
Frequency
175
150
125
100
75
50
25
0
0
1000
2000
3000
4000
AvgTransactionValue


Min Max - AvgTransactionValue
Frequency
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
AvgTransactionValue_MinMax


Z-Score - AvgTransactionValue
Frequency
175
150
125
100
75
50
25
0
−2
0
2
4
6
8
AvgTransactionValue_zscore


Original - LoginFrequency
Frequency
LoginFrequency
0
10
20
30
40
50
60
10
15
20
25
30
35
40


Min Max - LoginFrequency
Frequency
LoginFrequency_MinMax
0
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
1.0


Z-Score - LoginFrequency
Frequency
0
10
20
30
40
50
60
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
LoginFrequency_zscore

Section 4: Feature Engineering

```python
import pandas as pd
pd.set_option( pat: 'display.max_columns', val: None)

def handle_feature(df):  2 usages

    #creating new feature
    df['TotalSpend'] = df['TotalTransactions'] * df['AvgTransactionValue']


    #creating age groups
    df['AgeGroup'] = df['Age'].apply(
        lambda x :'Child' if x < 18 else
        'Young' if x < 25 else 'Young Adults'
        if x < 36 else 'Mid-Age Adults' if x < 46 else 'Old'
    )

    #converting isChurned
    df['IsChurned'] = df['IsChurned'].apply(
        lambda x : 1 if x.lower() == 'yes' else 0
    )

    #average transaction frequency per month
    df['MonthlyTransactionRate'] = df['TotalTransactions'] / (df['LoginFrequency'] / 4)

    #Derive a new score
    df['CustomerScore'] = (df['TotalSpend'] * df['SatisfactionScore']) / df['CartAbandonRate']

    df.to_csv('customer_features_enriched.csv', index=False)
```