Pseudocode for `create_initial_table_from_csv`:

```
function create_initial_table_from_csv(csv_file, table_name):
    df ← read_csv(csv_file)  // Read CSV into DataFrame
    print("Creating initial table", table_name)
    create_table_in_database(df, table_name)  // Create table schema
from df
    print("Loading initial data into", table_name)
    load_data_into_table(df, table_name)  // Insert data into table
    print("Initial data successfully loaded into", table_name)
```

Pseudocode for `create_final_table_from_csv`:

```
function create_final_table_from_csv(csv_file, table_name):
    df ← read_csv(csv_file)  // Read CSV into DataFrame

    // Convert negative days to positive
    df['DAYS_EMPLOYED'] ← abs(df['DAYS_EMPLOYED'])
    df['DAYS_BIRTH'] ← abs(df['DAYS_BIRTH'])

    print("Missing values before imputation:")
    print(count_missing_values(df))  // Display missing values

    df ← impute_missing_values(df)  // Impute missing values

    print("Missing values after imputation:")
    print(count_missing_values(df))  // Display missing values after
imputation

    df ← identify_and_handle_outliers(df)  // Handle outliers

    compute_statistics(df)  // Compute statistical measures

    df ← perform_feature_transformations(df)  // Apply feature
transformations

    generate_plots(df)  // Generate data plots
```

```
    create_table_in_database(df, table_name)  // Create table schema
from df
    load_data_into_table(df, table_name)  // Insert data into table
    print("Data successfully loaded into", table_name)

    missing_info ← analyze_missing_data(df)  // Analyze missing data
post-imputation
    print("Missing Values Analysis:")
    print(missing_info)
```

Pseudocode for `impute_missing_values`:

```
function impute_missing_values(df):
    df['HOUSETYPE_MODE'] ← fill_na_with_mode(df['HOUSETYPE_MODE'])  //
Impute categorical column
    df['EXT_SOURCE_1'] ← fill_na_with_median(df['EXT_SOURCE_1'])  //
Impute numerical column with median
    df['EXT_SOURCE_2'] ← fill_na_with_median(df['EXT_SOURCE_2'])
    df['EXT_SOURCE_3'] ← fill_na_with_median(df['EXT_SOURCE_3'])
    df['TOTALAREA_MODE'] ← fill_na_with_median(df['TOTALAREA_MODE'])
    df['AMT_REQ_CREDIT_BUREAU_YEAR'] ←
fill_na_with_zero(df['AMT_REQ_CREDIT_BUREAU_YEAR'])  // Impute
specific column with 0
    return df
```

Pseudocode for `identify_and_handle_outliers`:

```
function identify_and_handle_outliers(df):
    numerical_cols ← ['CNT_CHILDREN', 'CNT_FAM_MEMBERS',
'AMT_INCOME_TOTAL', 'AMT_CREDIT', ...]

    for each col in numerical_cols:
        Q1, Q3 ← quantile(df[col], 0.25), quantile(df[col], 0.75)
        IQR ← Q3 - Q1
        lower_bound ← Q1 - 1.5 * IQR
        upper_bound ← Q3 + 1.5 * IQR

        outliers ← find_outliers(df[col], lower_bound, upper_bound)
```

```
        print(outliers)

        df[col] ← cap_values_to_bounds(df[col], lower_bound,
upper_bound)

    return df
```

Pseudocode for `compute_statistics`:

```
function compute_statistics(df):
    numerical_cols ← ['CNT_CHILDREN', 'CNT_FAM_MEMBERS',
'AMT_INCOME_TOTAL', 'AMT_CREDIT', ...]

    statistical_measures ← describe(df[numerical_cols])
    statistical_measures['skewness'] ← skew(df[numerical_cols])
    statistical_measures['kurtosis'] ← kurtosis(df[numerical_cols])

    print(statistical_measures)

    categorical_cols ← ['TARGET', 'CODE_GENDER', 'FLAG_OWN_CAR',
'FLAG_OWN_REALTY', ...]

    for each col in categorical_cols:
        mode ← mode(df[col])
        value_counts ← value_counts(df[col])
        print(mode, value_counts)
```

Pseudocode for `perform_feature_transformations`:

```
function perform_feature_transformations(df):
    target_column ← 'TARGET'

    target_encoding_columns ← ['CODE_GENDER', 'FLAG_OWN_CAR',
'FLAG_OWN_REALTY']

    for each col in target_encoding_columns:
        category_means ← groupby_and_calculate_mean(df, col,
target_column)
```

```
        df[col] ← map_to_category_means(df[col], category_means)

    print("Target-based label encoding applied.")

    one_hot_encoding_columns ← ['NAME_INCOME_TYPE', 'HOUSETYPE_MODE',
'NAME_EDUCATION_TYPE', ...]
    df ← apply_one_hot_encoding(df, one_hot_encoding_columns)

    print("One-hot encoding applied.")

    return df
```

Pseudocode for `generate_plots`:

```
function generate_plots(df):
    set_plot_style("whitegrid")

    numerical_cols ← ['AMT_INCOME_TOTAL', 'AMT_CREDIT',
'TOTALAREA_MODE', ...]

    for each col in numerical_cols:
        plot_boxplot(df[col], col)
        save_plot("Box Plot for " + col)

    plot_boxplot(df[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']],
"EXT_SOURCE Values")
    save_plot("Box Plot for EXT_SOURCE Values")

    plot_scatter(df['EXT_SOURCE_2'], df['EXT_SOURCE_3'], "EXT_SOURCE_2
vs EXT_SOURCE_3")
    save_plot("Scatter Plot of EXT_SOURCE_2 vs EXT_SOURCE_3")

    plot_scatter(df['AMT_INCOME_TOTAL'], df['AMT_CREDIT'], "Income vs
Credit Amount")
    save_plot("Income_vs_Credit_Amount.png")

    plot_scatter(df['DAYS_BIRTH'], df['AMT_CREDIT'], "Age vs Credit
Amount")
```

```
save_plot("Age_vs_Credit_Amount.png")
```