# Detailed Program Documentation: Bank Statement Extractor

## Overview

This document provides a comprehensive overview of the Bank Statement Extractor, a Python-based application designed to automate the extraction of financial transaction data from various PDF bank statement formats. The application consists of two main components: a robust set of PDF data extraction functions (`extractors.py`) and a user-friendly graphical user interface (GUI) (`gui.py`). The primary goal of this system is to streamline the process of converting unstructured PDF bank statement data into structured, analyzable formats such as CSV or Excel, thereby reducing manual effort and potential for errors.

## Scope of Work

The scope of this project encompasses the development and integration of specialized data extraction logic for multiple bank statement formats, alongside a intuitive GUI for seamless user interaction. The system is designed to handle variations in PDF layouts and to provide accurate and reliable data extraction. The key functionalities and components within this scope include:

### 1. PDF Data Extraction Module (`extractors.py`)

This module is the core of the data extraction process. It includes a collection of functions, each meticulously crafted to parse and extract transactional data from specific bank statement layouts. The module is designed with extensibility in mind, allowing for the addition of new bank formats as needed. The main responsibilities of this module are:

- **Bank-Specific Parsers:** Implementation of dedicated functions (e.g., `banorte0`, `citibanamex0`, `banorte1`, `citibanamex1`, `banbajio`, `bbva`) to handle the

unique structural characteristics of different bank statements.

- **Data Normalization:** Conversion of extracted data into a consistent, standardized format (e.g., `Fecha`, `Descripción`, `Depósitos`, `Retiros`, `Saldo`) suitable for analysis and reporting.
- **Robust Error Handling:** Mechanisms to gracefully manage potential issues during PDF parsing, such as malformed documents or unexpected data patterns.
- **Automated Extractor Selection:** Logic to intelligently identify the correct bank statement format and apply the appropriate extraction function, enhancing user convenience.

### 2. Graphical User Interface (GUI) Module (`gui.py`)

This module provides an interactive and user-friendly interface for the Bank Statement Extractor, built using Python's `tkinter` library. The GUI simplifies the process of selecting PDF files, initiating the extraction, and saving the results, making the application accessible to users without programming expertise. The key features of the GUI include:

- **File Selection:** An intuitive interface for browsing and selecting PDF bank statement files.
- **Extraction Control:** Buttons and indicators to initiate and monitor the data extraction process.
- **Output Format Selection:** Options to choose between CSV and Excel formats for saving the extracted data.
- **User Feedback:** Display of progress, success messages, and error notifications to keep the user informed.
- **Concurrency:** Implementation of threading to ensure the GUI remains responsive during potentially time-consuming extraction operations.

## Technologies and Libraries Utilized

The Bank Statement Extractor leverages several powerful Python libraries to achieve its functionality. These libraries provide robust tools for PDF parsing, data manipulation, and GUI development:

## `extractors.py` Libraries:

### `pandas`

- **Description:** pandas is an open-source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It is widely used for data manipulation, analysis, and visualization.
- **Use Cases in the Program:** In `extractors.py`, pandas is primarily used to create and manipulate `DataFrame` objects. The extraction functions return data in a structured `DataFrame` format, which is ideal for further processing and saving to various output formats (e.g., CSV, Excel).
- **Key Features:**
  - `DataFrame` object for data manipulation with integrated indexing.
  - Tools for reading and writing data between in-memory data structures and different file formats.
  - Data alignment and integrated handling of missing data.
  - Reshaping and pivoting of data sets.
  - Label-based slicing, fancy indexing, and subsetting of large data sets.
  - Group by engine allowing flexible split-apply-combine operations on data sets.
  - High performance merging and joining of data series.
  - Time series-functionality: Date range generation, frequency conversion, moving window statistics, date shifting and lagging.

### `camelot`

- **Description:** Camelot is a Python library that can help you extract tables from PDFs. It is particularly useful for extracting tabular data from PDFs accurately and efficiently, especially from scanned documents or those with complex layouts.
- **Use Cases in the Program:** In `extractors.py`, Camelot is extensively used by `banorte0`, `citibanamex0`, `banorte1`, and `bbva` functions to read and extract tables from PDF bank statements. It uses different \'flavors\' (e.g., \'stream\') to handle various table structures.

- **Key Features:**
  - **Two flavors of table extraction:** `Lattice` for PDFs with clear lines separating cells, and `Stream` for PDFs with more free-form tables using whitespace to separate cells.
  - **High accuracy:** Designed to extract tables with high precision, even from challenging PDFs.
  - **Output flexibility:** Can export extracted tables to various formats like CSV, JSON, Excel, HTML, and SQLite.
  - **Visual debugging:** Provides tools to visualize the table detection process, helping in fine-tuning extraction parameters.
  - **Handles merged cells and spanning rows/columns:** Capable of dealing with complex table structures.

**`pdfplumber`**

- **Description:** pdfplumber is a Python library for extracting text and tabular data from PDFs. It allows for detailed inspection of PDF layouts, including information about each text character, rectangle, and line. It\'s particularly effective for extracting text and data from machine-generated PDFs.
- **Use Cases in the Program:** The `citibanamex1` and `banbajio` functions in `extractors.py` utilize `pdfplumber` to open and extract text lines from PDF pages. This text is then parsed using regular expressions to identify and extract relevant financial data.
- **Key Features:**
  - **Detailed PDF inspection:** Provides access to detailed information about PDF objects like characters, rectangles, and lines.
  - **Text extraction:** Efficiently extracts text from PDF pages.
  - **Table extraction:** Includes functionality for table extraction, often working well with visually defined tables.
  - **Visual debugging:** Allows for visual debugging of the extraction process, similar to Camelot.
  - **Works well with machine-generated PDFs:** Optimized for PDFs where text and layout information is precisely defined.

### `re`

- **Description:** The `re` module in Python provides regular expression operations. Regular expressions are powerful tools for pattern matching and manipulation of strings.
- **Use Cases in the Program:** The `re` module is extensively used across all extractor functions in `extractors.py` for pattern matching. For example, `DATE_RE` is used to validate and extract dates, `NUM_RE` to clean numerical strings, and `AMT_RE`, `DEP_KEYS` for identifying and extracting specific financial amounts and transaction types.
- **Key Features:**
  - **Pattern Matching:** Functions like `re.search()`, `re.match()`, `re.findall()`, and `re.finditer()` for finding patterns in strings.
  - **Substitution:** `re.sub()` for replacing occurrences of a pattern.
  - **Splitting:** `re.split()` for splitting strings by a pattern.
  - **Compilation:** `re.compile()` for compiling regular expression patterns into regular expression objects, which can be used for more efficient pattern matching when the same expression is used multiple times.
  - **Metacharacters:** Support for a wide range of metacharacters and special sequences for complex pattern definitions.

### `sys`

- **Description:** The `sys` module provides access to system-specific parameters and functions. It allows interaction with the Python interpreter and its environment.
- **Use Cases in the Program:** In `extractors.py`, `sys.stderr` is used in the `auto_extract` function to print messages to the standard error stream, indicating which extractor is being used or if no extractor matched. This is useful for debugging and logging.
- **Key Features:**
  - **`sys.argv`:** List of command-line arguments passed to a Python script.
  - **`sys.path`:** List of strings that specifies the search path for modules.

- **`sys.stdin`, `sys.stdout`, `sys.stderr`:** File objects corresponding to the interpreter`s standard input, output, and error streams.
- **`sys.exit()`:** Function to exit the Python interpreter.
- **`sys.version`:** String containing the version number of the Python interpreter.

**`warnings`**

- **Description:** The `warnings` module is a standard Python library that provides a way to control how warnings are handled within a Python program. It allows developers to issue warning messages to alert users of certain conditions that are not severe enough to be errors but should still be brought to attention.
- **Use Cases in the Program:** In `extractors.py`, the `warnings` module is used to suppress `CryptographyDeprecationWarning` messages. This is done to prevent these warnings from cluttering the output, especially since they might be related to underlying dependencies (like `camelot` or `pdfplumber`) rather than direct issues in the provided code.
- **Key Features:**
  - **Issuing Warnings:** Functions like `warnings.warn()` to emit warning messages.
  - **Controlling Warning Behavior:** The `warnings.filterwarnings()` function allows configuring how warnings are treated (e.g., ignore, always show, turn into errors).
  - **Warning Categories:** Supports different categories of warnings (e.g., `DeprecationWarning`, `UserWarning`) for finer control.
  - **Context Management:** Can be used with `with` statements to temporarily change warning filters.

## `gui.py` Libraries:

**`os`**

- **Description:** The `os` module in Python provides a portable way of using operating system dependent functionality. It allows interaction with the

operating system, such as creating files and directories, managing file paths, and executing system commands.

- **Use Cases in the Program:** In `gui.py`, the `os` module is likely used for path manipulation, such as getting the directory name of a file (`os.path.dirname`) or checking if a file exists (`os.path.exists`). While not explicitly shown in the truncated code, it`s a common utility for file operations in GUI applications.
- **Key Features:**
  - **Path Manipulation:** Functions like `os.path.join()`, `os.path.basename()`, `os.path.dirname()`, `os.path.exists()` for handling file and directory paths.
  - **Directory Operations:** Functions for creating, deleting, and listing directories (`os.mkdir()`, `os.rmdir()`, `os.listdir()`).
  - **Environment Variables:** Accessing and modifying environment variables (`os.environ`).
  - **Process Management:** Functions for interacting with processes, though less common in simple GUI applications.

### `threading`

- **Description:** The `threading` module in Python provides a high-level interface for working with threads. Threads are separate flows of execution within the same program, allowing for concurrent execution of tasks.
- **Use Cases in the Program:** In `gui.py`, the `threading` module is used to run the PDF extraction process (`_on_extract` method) in a separate thread. This is crucial for GUI applications to prevent the user interface from becoming unresponsive (freezing) while a potentially long-running operation like PDF processing is in progress. By offloading the extraction to a separate thread, the main GUI thread remains free to handle user interactions.
- **Key Features:**
  - **`Thread` class:** Represents an activity that runs in a separate thread of control.
  - **Synchronization Primitives:** Provides mechanisms like `Lock`, `RLock`, `Condition`, `Semaphore`, and `Event` to coordinate access to shared resources and prevent race conditions.

    - **Thread-Local Data:** Allows for data that is specific to a particular thread.
    - **Daemon Threads:** Threads that run in the background and are automatically terminated when the main program exits.

`tkinter`

- **Description:** `tkinter` is Python`s standard built-in package for creating graphical user interfaces (GUIs). It provides a simple and fast way to create desktop applications.
- **Use Cases in the Program:** In `gui.py`, `tkinter` is the core library used to build the entire GUI application. It is used to create the main window (`tk.Tk`), various widgets like `LabelFrame`, `Entry`, `Button`, `Radiobutton`, and to manage their layout using `pack()`.
- **Key Features:**
    - **Widgets:** A rich set of pre-built widgets like buttons, labels, entry fields, text areas, checkboxes, radio buttons, and more.
    - **Layout Managers:** Provides `pack()`, `grid()`, and `place()` methods for arranging widgets within a window.
    - **Event Handling:** Mechanisms to respond to user interactions like button clicks, key presses, and window events.
    - **Standard Library:** Being part of Python`s standard library, it doesn`t require separate installation.
    - **Cross-Platform:** Applications built with `tkinter` can run on various operating systems (Windows, macOS, Linux).

`tkinter.ttk`

- **Description:** The `tkinter.ttk` module provides access to the Tk themed widget set, introduced in Tk 8.5. These widgets offer a more modern look and feel compared to the classic `tkinter` widgets, and they also provide additional benefits like anti-aliased font rendering.
- **Use Cases in the Program:** In `gui.py`, `tkinter.ttk` is used for creating themed widgets such as `ttk.LabelFrame`, `ttk.Entry`, `ttk.Button`, and `ttk.Radiobutton`. Using `ttk` widgets helps in creating a more visually appealing and consistent user interface.

- **Key Features:**
  - **Themed Widgets:** Provides a set of widgets that can be styled using themes, allowing for a more native look and feel across different operating systems.
  - **Improved Aesthetics:** Offers better visual appearance, including anti-aliased fonts and more modern control styles.
  - **Enhanced Functionality:** Some `ttk` widgets offer improved functionality or behavior compared to their classic `tkinter` counterparts.
  - **Consistency:** Helps in maintaining a consistent look and feel throughout the application.

### `tkinter.filedialog`

- **Description:** The `tkinter.filedialog` module provides classes and factory functions for creating file and directory selection windows. It allows GUI applications to interact with the user`s file system in a standard and platform-independent way.
- **Use Cases in the Program:** In `gui.py`, `filedialog.askopenfilename()` is used in the `_browse_pdf` method to allow the user to select a PDF file for extraction. Similarly, `filedialog.asksaveasfilename()` is used in the `_on_save` method to let the user choose a location and filename for saving the extracted data (CSV or Excel).
- **Key Features:**
  - **`askopenfilename()`:** Opens a dialog box that allows the user to select a single file for opening.
  - **`asksaveasfilename()`:** Opens a dialog box that allows the user to specify a filename and location for saving a file.
  - **`askdirectory()`:** Opens a dialog box that allows the user to select a directory.
  - **Customization:** Supports options for filtering file types, setting initial directories, and providing default filenames.
  - **Native Dialogs:** Uses the native file dialogs of the operating system, providing a familiar user experience.

### `tkinter.messagebox`

- **Description:** The `tkinter.messagebox` module provides functions to display various types of message boxes, such as information, warning, error, question, and confirmation dialogs. These are useful for providing feedback to the user or asking for user input in a standardized way.
- **Use Cases in the Program:** In `gui.py`, `messagebox.showerror()` is used to display error messages to the user, for example, if an invalid PDF is selected or if an extraction fails. `messagebox.showinfo()` is used to inform the user about successful operations, such as saving the extracted data.
- **Key Features:**
  - **`showinfo()`:** Displays an informational message.
  - **`showwarning()`:** Displays a warning message.
  - **`showerror()`:** Displays an error message.
  - **`askquestion()`:** Asks a yes/no question.
  - **`askokcancel()`:** Asks an OK/Cancel question.
  - **`askyesno()`:** Asks a Yes/No question.
  - **`askyesnocancel()`:** Asks a Yes/No/Cancel question.
  - **`askretrycancel()`:** Asks a Retry/Cancel question.