Kenyan Collective Investment Schemes Dataset

Sourcing, Cleaning, and Exploring Collective Investment Schemes in Kenya

Kabui, Charles

2024-10-04

This study presents a comprehensive dataset of Kenyan Collective Investment Schemes and their performance from 2014 to 2024. By leveraging web crawling techniques on Capital Markets Authority (CMA) and Cytonn Research reports, we compiled data on Effective Annual Rates for Money Market Funds (KES) and Assets Under Management for the schemes (Money Market Funds, Equity Funds, Fixed Income Funds, Balanced Funds, and Special Funds). The research process involved data sourcing, cleaning, and basic exploratory analysis, resulting in a standardized dataset suitable for further research. We provide two primary visualizations: plots of Annual Rates for oney Market Funds (KES) and Assets Under Management over time. This dataset aims to serve as a foundation for researchers, investors, and industry stakeholders to conduct more in-depth analyses of Kenya's collective investment landscape. By archiving and publishing this data, we contribute to the accessibility and transparency of financial information in the Kenyan market.

Table of Contents

Updates

Thursday, December 19, 2024

Safaricom launches Ziidi Money Market Fund to mimic Mali Money Market Fund. Safaricom blamed Genghis for the Mali Market Fund's delayed launch and halts customer onbording due compaints of unstable Genghis platform. Genghis calls safaricom a fraud for secretly redirecting Mali Money Market Fund customers for Ziidi Money Market Fund thereby breaching data laws. Safaricom, Genghis Capital fight over M-Pesa unit trusts

Dry Associates Limited (DAL) has converted its Balanced Fund into the DAL High Yield Special Fund, now focused on high-yield returns through Kenyan

and East African government securities and corporate fixed-income investments, https://www.cma.or.ke/cma-approves-two-new-funds-to-expand-investment-opportunities-for-sophisticated-investors/

Introduction

In recent years, Kenya's financial landscape has witnessed significant growth and diversification, with Collective Investment Schemes playing an increasingly prominent role. Among these, Money Market Funds (MMFs) have emerged as a particularly popular investment vehicle, offering a unique blend of benefits that appeal to a wide range of investors in the Kenyan market.

Money Market Funds operate by pooling capital from numerous investors, which professional fund managers then invest collectively in short-term, highly liquid financial instruments. This structure allows MMFs to offer several key advantages in the Kenyan context:

1. **Higher Returns:** MMFs typically provide superior interest rates compared to standard savings accounts, making them an attractive option for investors seeking to maximize their returns on short-term investments.
2. **Lower Entry Barriers:** With the ability to start investing with smaller amounts, MMFs have democratized access to professional fund management, opening up opportunities for a broader range of Kenyan investors.
3. **Compound Interest:** Unlike most traditional bank deposits that offer simple interest, MMFs generally provide compound interest. This feature can lead to accelerated wealth accumulation over time, particularly benefiting long-term investors.
4. **Enhanced Liquidity:** MMFs maintain high liquidity, allowing investors to access their funds quickly when needed, typically within one to two business days after a withdrawal request. This flexibility is crucial in a dynamic economy like Kenya's, where financial needs can change rapidly.
5. **Diversification:** By investing in a variety of short-term securities, MMFs offer a level of diversification that can be challenging for individual investors to achieve on their own, especially with limited capital.

The growing popularity of MMFs in Kenya reflects broader trends in the country's financial sector, including increased financial literacy, a growing middle class, and the expansion of digital financial services. However, despite their importance, comprehensive and accessible data on the performance and characteristics of these funds has been limited.

This study aims to address this gap by creating a robust, clean dataset of Kenyan Collective Investment Schemes, with a particular focus on Money Market Funds. Our objectives are:

1. To source and gather relevant data from authoritative sources, including the Capital Markets Authority (CMA) and published financial reports.
2. To clean and standardize the collected data, ensuring consistency and reliability for analytical purposes.
3. To archive and publish the resulting dataset, facilitating further research and analysis by academics, industry professionals, and policymakers.

By undertaking this data-centric approach, we aim to contribute to the broader understanding of Kenya's financial market dynamics, particularly in the realm of collective investments. This dataset will serve as a foundation for more in-depth analyses, potentially informing investment strategies, policy decisions, and academic research in the field of Kenyan finance.

In the following sections, we will detail our methodology for data collection and cleaning, and present the structure of the resulting dataset. Through this effort, we hope to not only shed light on the current state of Money Market Funds in Kenya but also to set a precedent for transparent, reproducible financial data curation in emerging markets.

Sourcing and Gathering Data

Before we begin, lets prepare our enviroment with some important python packages and reusable functions

```
import sys
import os
from pathlib import Path

# Add root directory as python path
root_dir = os.path.abspath(Path(sys.executable).parents[2])
sys.path.append(root_dir)

%reload_ext autoreload
%autoreload 2

# Other imports
import pandas as pd
from playwright.async_api import Page, Route
import asyncio
import io
from bs4 import BeautifulSoup, Tag
from urllib.request import urlopen
import json5 as json5
import json
from tqdm import tqdm
```

```
import re
from typing import Callable, Literal
from copy import copy
from datetime import datetime, timedelta, date
from calendar import monthrange, month_abbr
import plotly.express as px
from json2txttree import json2txttree
from python_utils.web_screenshot import web_screenshot_async
from python_utils.get_browser import get_browser_page_async
from typing import Any
from toolz import groupby
import numpy as np
import inspect
from PIL import Image
import plotly.io as pio
pio.renderers.default = "plotly_mimetype+notebook_connected"
```

```
collective_scheme_type = \
    dict[Literal['Scheme'], str] | dict[Literal['Funds'], list[str]]


def strip_start_end(s1: str, last_acceptable_characters = ')'):
    """
    Cleans a given string by removing specific patterns and non-alphabet
    characters at the start and end of a string.

    Args:
        s1 (str): The input string to be cleaned.
        last_acceptable_characters (str, optional):
            Characters that are acceptable at the end of the string. Defaults to ')'.

    Returns:
        str: The cleaned string.

    The function performs the following steps:
    1. Removes the phrase "comprising of" or "which comprises of".
    2. Removes the word "and" followed by any non-alphabet characters at the end of the stri
    3. Removes any non-alphabet characters from the start of the string.
    4. Removes any non-alphabet characters from the end of the string,
        except those specified in `last_acceptable_characters`.
    5. Replaces multiple spaces with a single space.
    6. Strips leading and trailing whitespace.
```

```
    7. Recursively applies the function if any of the patterns still match the string.
    8. Removes non-ASCII characters.

    Example:
        >>> strip_start_end("comprising of example and123")
        'example'
    """
    if type(s1) != str or s1 is None:
        return ''
    # Define a regex pattern to match 'and' followed by any non-alphabet
    # characters at the end of the string
    and_pattern = r'\band[^a-zA-Z]*$'
    # Define a regex pattern to match any non-alphabet characters at the start of the string
    non_alphabet_start = r'^[^a-zA-Z]+'
    # Define a regex pattern to match any non-alphabet characters at the end of the string
    non_alphabet_end = f'[^a-zA-Z{last_acceptable_characters}]+$'
    # Define a regex pattern to match the phrase "comprising of|which comprises of"
    comprising_of_pattern = r'comprising of|which comprises of'
    # Replace multiple spaces with a single space
    multiple_white_space = r'\s+'
    s2 = re.sub(comprising_of_pattern, '', s1)
    s3 = re.sub(and_pattern, '', s2)
    s4 = re.sub(non_alphabet_start, '', s3)
    s5 = re.sub(non_alphabet_end, '', s4)
    s6 = re.sub(multiple_white_space, ' ', s5)
    s7 = s6.strip()
    # Recursively apply the function if any of the patterns still match the string
    while any(re.match(p, s7) for p in [
        and_pattern, non_alphabet_start, non_alphabet_end, comprising_of_pattern]):
        return strip_start_end(s5)
    # remove non ASCII characters
    s8 = s7.encode('ascii', errors='ignore').decode()
    # Return the cleaned string
    return s8

def hacky_normalizer(val: str):
    """
    Normalizes a given string by performing the following operations:
    1. Strips leading and trailing whitespace.
    2. Converts the string to uppercase.
    3. Replaces special characters with underscores,
        (non-alphanumeric, non-percent, non-parentheses, non-underscore).
```

```
    4. Replaces multiple consecutive underscores with a single underscore.

    Args:
        val (str): The input string to be normalized.

    Returns:
        str: The normalized string.
    """
    val = val.strip().upper()
    # Replace special characters with underscore
    modified_string = re.sub(r'[^a-zA-Z0-9\%()_]', '_', val)
    # Replace multiple consecutive underscores with a single underscore
    modified_string = re.sub(r'_+', '_', modified_string)
    return modified_string

def dynamic_callback(callback, *args):
    """
    Dynamically calls a callback function with the appropriate number of arguments.
    This function inspects the signature of the provided callback function to determine
    the number of parameters it accepts. It then calls the callback with the corresponding
    number of arguments from the provided *args.
    Args:
        callback (Callable): The function to be called.
        *args: Variable length argument list to be passed to the callback.
    Returns:
        The result of the callback function call.
    Raises:
        TypeError: If the callback is not callable.
    """
    sig = inspect.signature(callback)
    param_count = len(sig.parameters)

    if param_count == 0:
        return callback()
    return callback(*args[:param_count])
```

Approved Collective Schemes

To get a comprehensive and up to date list of approved collective managers, we crawled Capital Markets Authrity (CMA). They have published a list of approved schemes https://www.cma.

or.ke/licensees-market-players/[1] and https://licensees.cma.or.ke/licenses/15/[2].

Screenshots of the pages

Lets start with some screenshots of the pages

www.cma.or.ke

```
async def collective_investment_schemes_click_fn(page: Page):
    await page.wait_for_selector('ul.module-accordion')
    elements = await page.query_selector_all('li .accordion-title')
    for element in elements:
        text_content = await element.text_content()
        if 'APPROVED COLLECTIVE INVESTMENT SCHEMES' in text_content:
            await element.click()
            accordion_element = await page.wait_for_selector('li.current.builder-accordion-a
            await page.evaluate("""
                document.querySelector('#headerwrap').style.display = 'none';
                document.querySelector('.pojo-a11y-toolbar-toggle').style.display = 'none';
            """)
            await asyncio.sleep(1)
            return accordion_element
    print('Element not found')

# Take a screenshot
await web_screenshot_async(
    "https://www.cma.or.ke/licensees-market-players/",
    action = collective_investment_schemes_click_fn,
    width = 1000,
    screenshot_options = None,
    crop_options = { 'bottom': 600, 'right': 600 })
```

[1] https://www.cma.or.ke/licensees-market-players/ - archive
[2] https://licensees.cma.or.ke/licenses/15/ - archive

↑ APPROVED COLLECTIVE INVESTMENT SCHEMES

NAME
1. African Alliance Kenya Unit Trust Scheme – comprising of: ◦ African Alliance Kenya Money Market Fund (Formerly African Alliance Kenya Shilling Fund); ◦ African Alliance Kenya Fixed Income Fund; ◦ African Alliance Kenya Managed Fund; ◦ African Alliance Kenya Equity Fund; and ◦ African Alliance Kenya Enhanced Yield Fund.
2. British-American Unit Trust Scheme, comprising of: ◦ British-American Money Market Fund; ◦ British-American Income Fund; ◦ British-American Balanced Fund; and ◦ British-American Equity Fund.

licensees.cma.or.ke

```
async def collective_investment_schemes_2(page: Page):
    return await page.query_selector('table')

# Take a screenshot
await web_screenshot_async(
    # Fund manager URL
```

```
    "https://licensees.cma.or.ke/licenses/15/",
    action = collective_investment_schemes_2,
    width = 2000,
    screenshot_options = None,
    crop_options = { 'bottom': 500, 'right': 700 },)
```

Name	Address	License Number	
African Alliance Kenya Unit Trust Scheme • African Alliance Kenya Money Market Fund (Formerl • African Alliance Kenya Fixed Income Fund; • African Alliance Kenya Managed Fund • African Alliance Kenya Equity Fund; • African Alliance Kenya Enhanced Yield Fund.			https://www.africanalliance.com/
British-American Unit Trust Scheme			https://ke.britam.com/save-and-i

Crawling

Next, let's try grab the schemes table into a dataframe that we can work with. Below is the list of all the certified schemes in Kenya by CMA. [3] [4]

```
def extract_collective_scheme_name(para: Tag):
    full_name = ' '.join([i.get_text(strip=True) for i in para.find_all('strong')])
    return strip_start_end(full_name)
```

[3] https://www.cma.or.ke/licensees-market-players/ - archive
[4] https://licensees.cma.or.ke/licenses/15/ - archive

```
def make_collective_unit_obj(tbody_tr_td: Tag) -> collective_scheme_type:
    return {
        'Scheme': extract_collective_scheme_name(tbody_tr_td.find('p') or tbody_tr_td),
        'Funds': [
            strip_start_end(i.get_text(separator=' ', strip=True))
            for i
            in tbody_tr_td.select('ul li')
        ]
    }

def fetch_collective_schemes_1():
    CMA_market_players_html: str = urlopen("https://www.cma.or.ke/licensees-market-players/")
    investment_schemes_table_html = BeautifulSoup(CMA_market_players_html, "html.parser")\
        .find('span', string="APPROVED COLLECTIVE INVESTMENT SCHEMES")\
        .find_parent('li')\
        .find('table')
    return [
        make_collective_unit_obj(tbody_tr_td)
        for tbody_tr_td
        in investment_schemes_table_html.select('tbody tr td')
    ]

def fetch_collective_schemes_2():
    CMA_market_players_html: str = urlopen("https://licensees.cma.or.ke/licenses/15/").read()
    investment_schemes_table_html = BeautifulSoup(CMA_market_players_html, "html.parser")\
        .find('table')
    return [
        make_collective_unit_obj(tbody_tr_td)
        for tbody_tr_td
        in investment_schemes_table_html.select('tbody tr > :first-child')
    ]

# For example:
#       Orient Umbrella Collective Investment Scheme (formerly Alphafrica Umbrella Fund) =>
#       Orient Umbrella Collective Investment Scheme
def remove_quoted_str(str1: str): return re.sub(r'\(.*?(?!\)).*?$', '', str1 or '').strip()
def remove_rendadant_words(str1: str):
    return re.sub(
        r'\b(scheme|schemes|trust|trusts|specialized|special|funds|fund|unit|units|collective
        '',
        str1 or '',
        flags=re.IGNORECASE).strip()
```

```
def remove_special_words(str1: str):
    return re.sub(
        r'\b(specialized|special)\b\s*',
        '',
        str1 or '',
        flags=re.IGNORECASE).strip()

def make_merge_key(str1: str): return hacky_normalizer(remove_rendadant_words(remove_quoted_s

def merge_collective_schemes(schemes_list: list[collective_scheme_type]) -> collective_scheme
    all_names: dict[str, list[str]] = groupby(
        make_merge_key, [unit_obj['Scheme'] for unit_obj in schemes_list])
    all_schemes: dict[str, list[str]] = groupby(
        make_merge_key, [scheme for unit_obj in schemes_list for scheme in unit_obj['Funds']]
    return {
        'Scheme': sorted(
            [name for values in all_names.values() for name in values],
            key = lambda x: len(remove_special_words(remove_quoted_str(x))),
            reverse=True
        )[0],
        'Funds': [
            sorted(schemes, key = lambda x: len(x), reverse=True)[0]
            for schemes
            in all_schemes.values()
        ]
    }

collective_schemes_1 = [] # fetch_collective_schemes_1()
collective_schemes_2 = fetch_collective_schemes_2()
collective_schemes_1_2 = collective_schemes_1 + collective_schemes_2
collective_schemes_grouped_by_name = groupby(
    lambda x: make_merge_key(x['Scheme']), collective_schemes_1_2)
collective_schemes = [
    merge_collective_schemes(collective_schemes)
    for collective_schemes
    in collective_schemes_grouped_by_name.values()
    if len(collective_schemes) > 0 and len(collective_schemes[0]['Scheme']) > 0]
collective_schemes_df = pd.DataFrame(collective_schemes)
collective_schemes_df
```

	Scheme	Funds
0	African Alliance Kenya Unit Trust Scheme	[African Alliance Kenya Money Market Fund (For
1	British-American Unit Trust Scheme	[Britam Money Market fund (USD), Britam Incor
2	NCBA Unit Trust Funds	[NCBA Fixed Income Fund, NCBA Equity Fund,
3	Zimele Unit Trust Scheme	[Zimele Balanced Fund, Zimele Money Market Fu
4	ICEA Unit Trust Scheme	[ICEA Money Market Fund, ICEA Equity Fund,
5	CIC Unit Trust Scheme	[CIC Money Market Fund, CIC Balanced Fund, C
6	Madison Unit Trust Fund	[Madison Money Market Fund, Madison Fixed Inc
7	Dyer and Blair Unit Trust Scheme	[Dyer and Blair Diversified Fund, Dyer and Bla...
8	Amana Unit Trust Funds Scheme	[Amana Money Market Fund, Amana Balanced F
9	Diaspora Unit Trust Scheme	[Diaspora Money Market Fund, Diaspora Bond Fu
10	First Ethical Opportunities Fund	[]
11	Genghis Unit Trust Funds	[GenCap Hazina Fund (Bond Fund), GenCap Ene
12	Mali Money Market Fund	[]
13	Sanlam Unit Trust Scheme	[Sanlam Money Market Fund, Sanlam Balanced F
14	Nabo Africa Funds	[Nabo Africa Money Market Fund, Nabo Africa B
15	Old Mutual Unit Trust Scheme	[Old Mutual Equity Fund, Old Mutual Money Ma
16	Equity Investment Bank Collective Investment S...	[Equity Investment Bank Money Market Fund, Eq
17	Dry Associates Unit Trust Scheme	[Dry Associates Money Market Fund (Kenya Shill
18	Co-op Trust Fund	[Co-op Balanced Fund, Co-op Equity Fund, Co-op
19	Apollo Unit Trust Scheme	[Apollo Money Market Fund, Apollo Balanced Fu
20	Cytonn Unit Trust Scheme	[Cytonn Money Market Fund (USD), Cytonn Bala
21	Orient Umbrella Collective Investment Scheme (...	[Orient Hifadhi Fixed Income Fund (formerly Al..
22	Wanafunzi Investment Unit Trust Fund	[]
23	Absa Unit Trust Funds	[Absa Balanced Fund, Absa Bond Fund, Absa Do
24	Jaza Unit Trust Fund	[Jaza Premier Money Market Fund]
25	Masaru Unit Trust Scheme	[Masaru Wealth Management Fund, Masaru Mone
26	ADAM Unit Trust Scheme	[ADAM Balanced Fund, ADAM Equities Fund, A
27	KCB Unit Trust Scheme (formerly Natbank Unit T...	[KCB Money Market Fund (USD) (formerly Natb
28	GenAfrica Unit Trust Scheme	[GenAfrica Money Market Fund, GenAfrica Equit
29	Amaka Unit Trust (Umbrella) Scheme	[Amaka HOSP Fixed Income Fund, Amaka Quali
30	Jubilee Unit Trust Collective Investment Scheme	[Jubilee Balanced Fund, Jubilee Equity Fund, J...
31	Enwealth Capital Unit Trust Scheme	[Enwealth Balanced Fund, Enwealth Equity Fund
32	Kuza Asset Management Unit Trust Scheme	[Kuza Fixed Income Fund, Kuza Money Market F
33	Etica Unit Trust Funds which has the following...	[Etica Money Market Fund(USD), Etica Fixed Inc
34	Lofty Corban Unit Trust Scheme	[Lofty Corban Equity Fund, Lofty Corban Specia.
35	Standard Investment Trust Funds	[Standard Investment Equity Growth Fund, Stand
36	Faida Unit Trust Funds	[Hazina Bond Fund, Angaza Money Market Fund
37	Taifa Unit Trust Funds	[Taifa Money Market Fund (KES), Taifa Miney M
38	Stanbic Unit Trust Funds	[Stanbic Money Market Fund, Stanbic Fixed Inco
39	Spearhead Africa Infrastructure (Special) Fund	[Spearhead Africa Infrastructure (Special) Fund]
40	Rencap Unit Trust Scheme	[Rencap Money Market Fund(KES), Rencap Bala

	Scheme	Funds
41	Mayfair umbrella Collective investment scheme	[Mayfair Money Market Fund, Mayfair Fixed Inco
42	Investcent Partners Trust Fund	[Investcent Multi Asset Special Fund(KES), Inv...
43	Investcent Alternative Investment Fund	[Investcent Hedge Fund (KES)]
44	ICEA LION Collective Investment Scheme	[ICEA LION Money Market Fund, ICEA LION E
45	GCIB Unit Trust Scheme	[GCIB Money Market Fund, GCIB Fixed Income
46	CPF Unit Trust Funds	[CPF Money Market Fund, CPF Bond Fund, CPI
47	Arvocap Unit Trust Scheme	[Arvocap Money Market Fund, Arvocap Ngao Fix
48	MyXENO Unit Trust Scheme	[Xeno Kenya Money Market Fund, Xeno Kenya B
49	VCG Offshore Opportunities Special Fund	[Offshore Equities Megatrends Special Fund (US..
50	Octagon Unit Trust Scheme	[Octagon Money Market Fund, Octagon Balanced

As of 2024-12-22, there are **49 unique** and approved unit trust schemes in Kenya, regulated by the Capital Markets Authority (CMA). The management of these schemes involves a complex ecosystem of financial institutions, each playing a distinct role:

1. **Approved Fund Managers**: These are entities specifically licensed by the CMA to manage collective investment schemes. They are responsible for making investment decisions and managing the day-to-day operations of the funds [5].

2. **Investment Banks**: Investment banks are not the traditional commercial banks, but rather CMA-approved institutions [6] that can engage in activities such as underwriting, market making, and fund management. For example, Genghis Capital Limited is listed by the CMA as an investment bank and manages its own unit trust fund, the Gencap Hela Imara Money Market Fund [7].

3. **Commercial Banks with Asset Management Arms**: Traditional banks may establish separate entities for asset management. For instance, KCB Bank has KCB Asset Management, which is approved by the CMA to manage unit trusts[8]. Please note that there is also KCB Investment Bank[9] [10].

4. **Non-Financial Companies with Investment Products**: Some companies outside the traditional financial sector have entered the investment market. A notable example is the Mali Money Market Fund [11] [12], owned wholly or in part by Safaricom PLC, Kenya's

[5] CMA Approved Fund Managers - archive
[6] CMA Approved Investment Banks - archive
[7] Genghis Capital Unit Trust Fund - archive
[8] CMA Approved Fund Managers - archive
[9] CMA Approved Investment Banks - archive
[10] KCB restructures investment units after buyout of NBK - archive
[11] Frequently Asked Questions / Mali - archive
[12] Safaricom to launch unit trust, new savings service - archive

largest telecommunications company [13]. While Safaricom is not a licensed fund manager, they have partnered with Genghis Capital Limited to administer the Mali MMF [14].

This complex landscape can sometimes lead to potential conflicts of interest [15]. For instance, when an investment bank like Genghis Capital manages both its own funds and third-party funds like the Mali MMF, it raises questions about prioritization and fair treatment of all clients. An additional layer of complexity arises when commercial banks (or their subsidiaries), such as KCB[16], offer unit trust investment options alongside traditional savings and fixed deposit accounts. This dual offering presents a potential conflict of interest. Banks typically earn higher profits from traditional deposit accounts compared to the fees generated from managing unit trusts. This raises questions about how banks advise their clients on savings options. While money market funds often provide better returns for savers, banks might have an inherent incentive to promote their own deposit products. This situation underscores the importance of financial literacy and independent advice for consumers navigating these choices. Investors should be aware of this potential conflict and critically evaluate the recommendations they receive, considering whether the advice aligns more with their own financial interests or those of the bank.

The investment landscape is constantly evolving, with fund managers occasionally modifying their product offerings. A notable example is Zimele Asset Management's decision to convert its Money Market Fund into a Fixed Income Fund [17]. This transition underscores the fluid nature of investment products in Kenya. Zimele's clients were not given the option to retain their investments in the original Money Market Fund structure. Some investors expressed dismay at discovering their Money Market Accounts had been altered without their knowledge [18], suggesting a lack of comprehensive public communication about the change. This situation emphasizes the importance of regularly monitoring one's investments to stay informed about any modifications that may affect them.

Given this complexity, potential investors should exercise caution and conduct thorough due diligence before committing their funds. This includes:

1. Verifying the regulatory status of the fund and its manager with the CMA
2. Understanding the fund's investment strategy and associated risks
3. Reviewing the fund's performance history and fee structure
4. Investigating any potential conflicts of interest
5. Seeking independent financial advice if necessary

[13] M-PESA / M-PESA Services / Wealth / Mali - archive
[14] Safaricom's Mali unit trust asset base hits Sh1.4bn - archive
[15] Safaricom, Genghis Capital fight over M-Pesa unit trusts
[16] KCB restructures investment units after buyout of NBK - archive
[17] Zimele Savings Plan Transition: From Money Market to Fixed Income Fund - archive
[18] What Happened to the Zimele Money Market Fund? - archive

> **Warning**
>
> Always approach investments with caution, especially when important information is missing, unclear, or overly complicated. Remember that higher returns often come with higher risks, and past performance does not guarantee future results.

Scheme Performance Data Collection

Challenges in Data Accessibility

The Kenyan financial regulatory environment mandates that unit trust schemes publish their daily yields in two national newspapers. However, this requirement presents several challenges for comprehensive data collection and analysis:

1. Limited Digital Presence: Many newspapers lack a substantial digital archive, necessitating physical access to print copies for data retrieval.
2. Cost Barriers: Accessing historical data often involves purchasing old newspaper records, making large-scale data collection financially prohibitive.
3. Time-Intensive Process: Manually gathering data from physical newspapers is a labor-intensive task, impractical for long-term, comprehensive analysis.
4. Inconsistent Reporting: Not all fund managers consistently publish their yields, leading to gaps in the data.

These factors collectively create a significant barrier to accessing and analyzing comprehensive, historical performance data for Kenyan unit trust schemes.

Cytonn Research: A Valuable Data Source

In light of these challenges, Cytonn Fund Managers' research publications have emerged as an invaluable resource. Since 2014, Cytonn has been conducting and freely publishing market research at https://cytonnreport.com/ [19]. Key aspects of this data source include:

- Comprehensive Coverage: Over 600 reports covering various aspects of the Kenyan financial market.
- Historical Data: Consistent reporting since 2014, providing a substantial historical dataset.
- Free Access: Public availability of the reports, removing financial barriers to data access.
- Aggregated Information: Cytonn's reports often include compiled data from multiple sources, offering a more comprehensive view of the market.

[19] https://cytonnreport.com/ - archive

While some fund managers publish current yield data on their websites[20], the lack of historical data limits the usefulness of these sources for trend analysis and comprehensive research.

Data Collection Methodology

Given the richness and accessibility of Cytonn's research, we adopted the following approach for data collection:

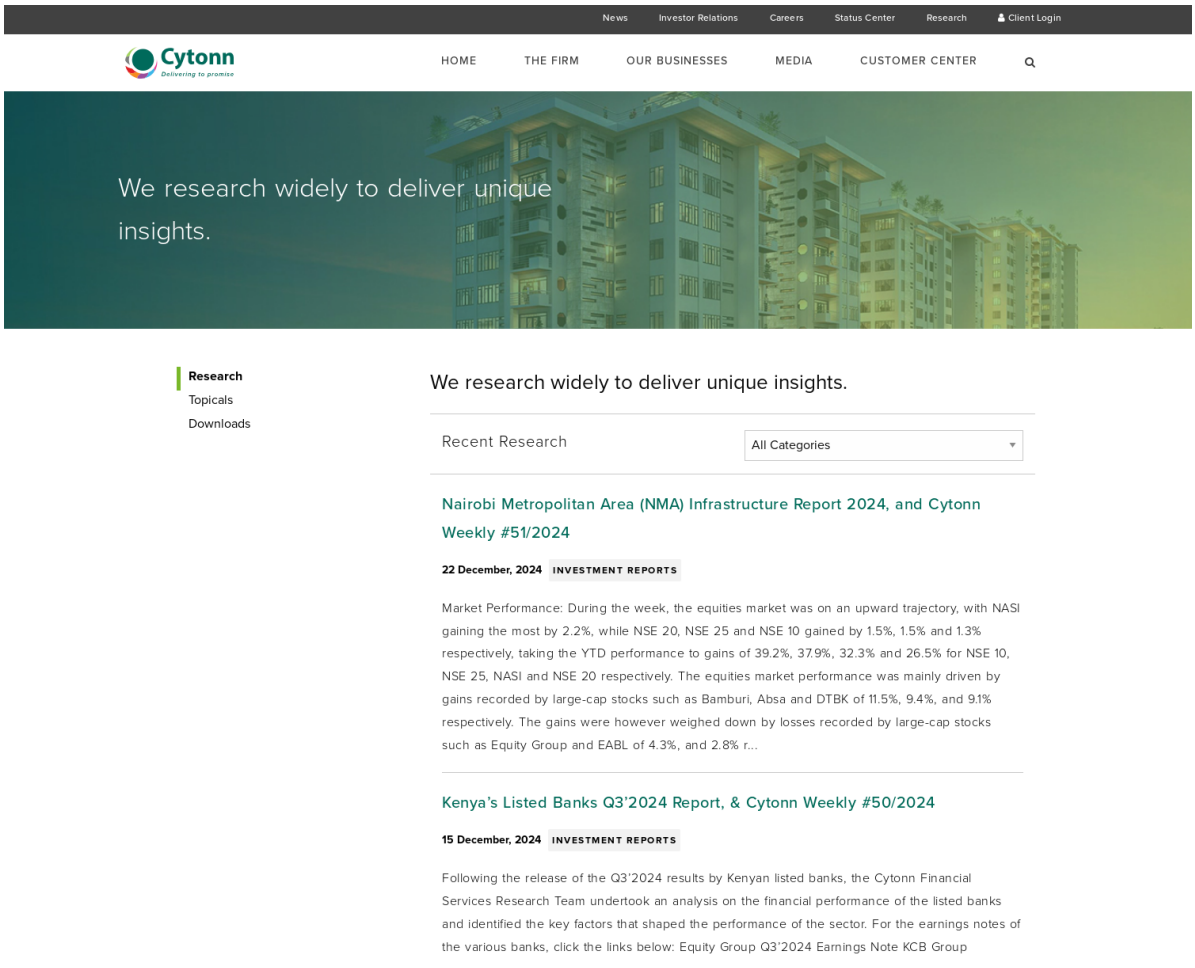
1. Web Crawling: We developed a crawling mechanism to systematically access reports from both https://cytonn.com/researches [21] and https://cytonnreport.com/research [22].
2. Ethical Considerations: Our crawling process was designed to respect Cytonn's server resources, avoiding any disruption to their services.
3. Data Extraction: We implemented a process to extract relevant tables and data points from each report.
4. Data Aggregation: The extracted information was compiled into a structured dataset suitable for analysis.
5. Compliance with Terms of Service: We carefully reviewed Cytonn's terms of service to ensure our use of the data aligns with their fair use policy[23].

Screenshots of Cytonn Reports

cytonn.com page

```
# Take a screenshot
await web_screenshot_async(
    "https://cytonn.com/researches",
    width = 1500,
    height = 1200)
```

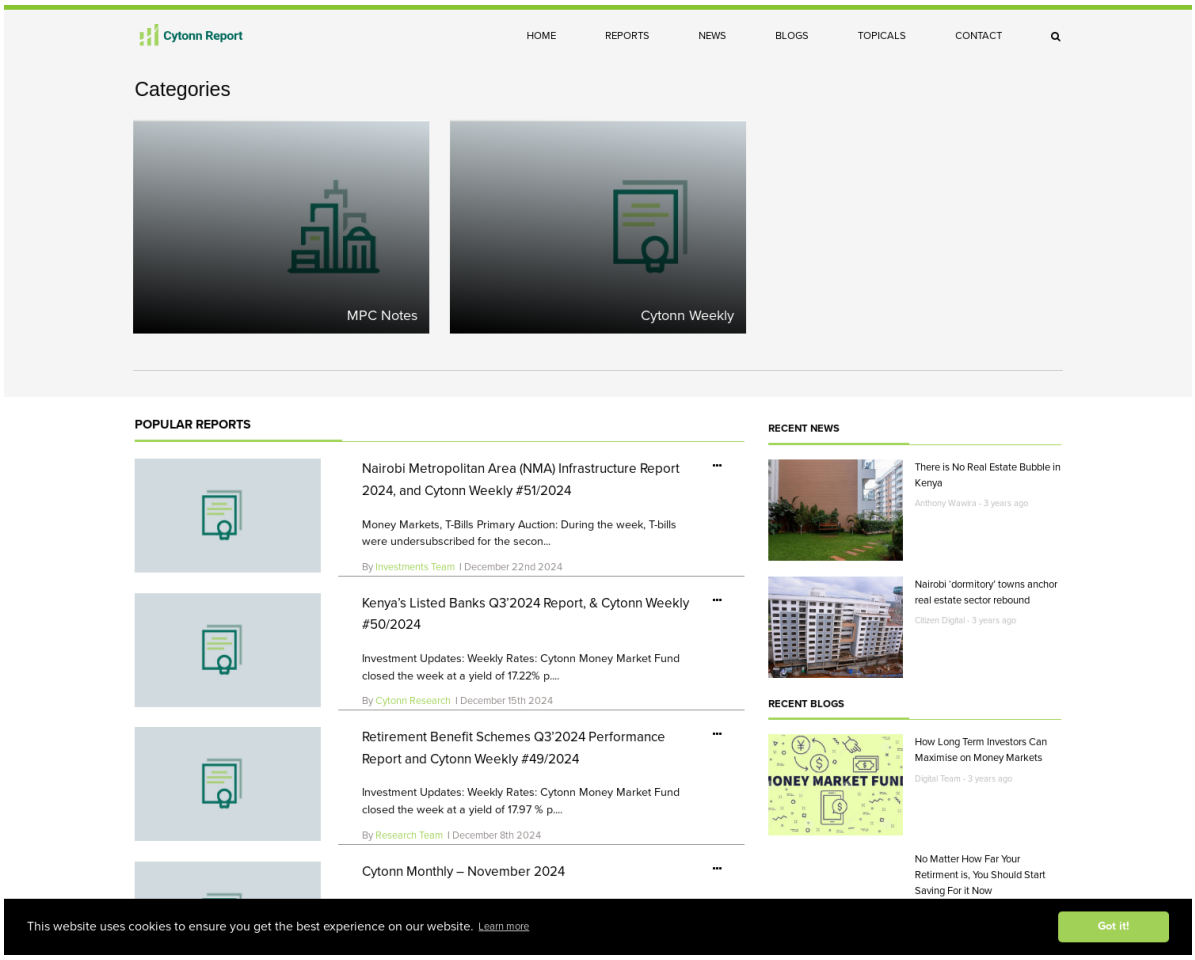
[20] https://ke.cicinsurancegroup.com/mmf/ - archive
[21] https://cytonn.com/researches - archive
[22] https://cytonnreport.com/research - archive
[23] Reproduction is prohibited other than in accordance with the copyright notice, which forms part of these terms and conditions. https://cytonn.com/terms-of-use - archive



cytonnreport.com page

```
async def cytonnreport_fn(page: Page):
    await page.wait_for_selector('.grid-x > .pagination')
    await asyncio.sleep(1)

# Take a screenshot
await web_screenshot_async(
    "https://cytonnreport.com/research",
    action = cytonnreport_fn,
    width = 1500,
    height = 1200)
```



Money Market Fund Yield Table

```
# Define a function that selects a table by its header text
def select_table_by_title(target_header_text: str):
    # Define a nested asynchronous function that takes a Page object as an argument
    async def fn(page: Page):
        # Wait for any table element to be present on the page
        await page.wait_for_selector('table')
        # Query and collect all table elements on the page
        table_elements = await page.query_selector_all('table')
        # Iterate through each table element
        for table_element in table_elements:
            # Query and collect all header cells in the current table
            table_headers = await table_element.query_selector_all('thead tr td')
```

```
            # Iterate through each header cell
            for table_header in table_headers:
                # Extract the text content of the current header cell
                header_text:str = await page.evaluate('(element) => element.textContent', tab
                # Check if the header text starts with the target text
                if header_text.startswith(target_header_text):
                    # If a match is found, return the current table element
                    await asyncio.sleep(1)
                    return table_element
    return fn

await web_screenshot_async(
    # URL to take a screenshot of
    "https://cytonnreport.com/research/cytonn-monthly-",
    # Action deciding WHAT (element) or WHEN (eg: click) to take the screenshot
    action = select_table_by_title('Cytonn Report: Money Market Fund Yield'),
    width = 500,
    crop_options = { 'bottom': 600 })
```

Cytonn Report: Money Market Fund Yield for Fund Managers as published on 02 August 2024		
Rank	Fund Manager	Effective Annual
1	Cytonn Money Market Fund *(Dial *809# or download the Cytonn app)*	18.3%
2	Lofty-Corban Money Market Fund	18.3%
3	Etica Money Market Fund	18.3%
4	Kuza Money Market fund	17.2%
5	Arvocap Money Market Fund	17.2%
6	GenAfrica Money Market Fund	16.9%
7	Nabo Africa Money Market Fund	16.6%
8	GenCap Hela Imara Money Market Fund	16.2%
9	Jubilee Money Market Fund	16.1%
10	Enwealth Money Market Fund	16.0%

Crawling

At https://cytonn.com/researches [24], we can crawl and parse HTML, but it could be very slow. We notice that https://cytonnreport.com/research [25], the exact same data is displayed, but using a background request, https://cytonnreport.com/get/allreports. We can use this to crawl multiple reports faster.

```
async def get_all_cytonn_reports(per_page_count: int = 10):
    """
    Retrieves all Cytonn reports from the Cytonn Report website.

    Args:
        per_page_count (int, optional): The number of reports to retrieve per page. Defaults

    Returns:
        list: A list of all the retrieved reports.
    """
    page, close_playwright = await get_browser_page_async()
    reports_url = "https://cytonnreport.com/get/allreports"
    reports_headers: dict = None
    reports_method: str = None

    async def handle_route(route: Route):
        nonlocal reports_headers
        nonlocal reports_method
        reports_headers = route.request.headers.copy()
        reports_method = route.request.method
        await route.continue_()

    async def get_cytonn_reports(current_page: int):
        javascript_fetch_fn = f'''
            async () => {{
                try {{
                    const response = await fetch(
                        "{reports_url}",
                        {{
                            "headers": {json.dumps(reports_headers)},
                            "method": "{reports_method}",
                            "body": {json.dumps(json.dumps(
                                {
                                    "pagination": {
```

[24] https://cytonn.com/researches - archive
[25] https://cytonnreport.com/research - archive

```
                                    "per_page": per_page_count,
                                    "current_page": current_page
                                }
                            }))},
                        "referrer": "https://cytonnreport.com/research",
                        "referrerPolicy": "no-referrer-when-downgrade",
                        "mode": "cors",
                        "credentials": "include"
                    }});
                if (!response.ok) {{
                    throw new Error(`HTTP error! status: ${{response.status}}`);
                }}
                const json = await response.json();
                return json;
            }} catch (error) {{
                console.error('Fetch error:', error);
                throw error; // Re-throw to allow calling code to handle it
            }}
        }}
    '''
    return await page.evaluate(javascript_fetch_fn)

await page.route(reports_url, handle_route)

# Navigate to the desired URL
await page.goto("https://cytonnreport.com/research")
while not reports_headers:
    await asyncio.sleep(1)
current_page = 1
all_reports = []
pbar: tqdm = None
while True:
    reports_response = await get_cytonn_reports(current_page)
    reports = reports_response['data'] if reports_response else []
    if len(reports) > 0:
        total = reports_response['total']
        pbar = pbar or tqdm(total=total)
        pbar.update(len(reports))
        all_reports.extend(reports)
        last_page = reports_response['last_page']
        if last_page == current_page:
            break
```

```
            current_page += 1
        else:
            break
        await asyncio.sleep(0.4)
    await close_playwright()
    if pbar:
        pbar.close()
    return all_reports

all_cytonn_reports = await get_all_cytonn_reports()
print(f'There are {len(all_cytonn_reports)} reports')
```

```
100%|      | 664/664 [03:03<00:00,  3.63it/s]

There are 664 reports
```

```
# converting the JSON into dataframe
all_cytonn_reports_df = pd.DataFrame(all_cytonn_reports)
all_cytonn_reports_df.head(3)
```

	id	slug	name
0	748	nairobi-metropolitan-area-9	Nairobi Metropolitan Area (NMA) Infrastructure
1	747	kenyas-listed-banks-q3-2024-report-and-cytonn-...	Kenya's Listed Banks Q3'2024 Report, & Cytonn
2	746	retirement-benefit-schemes	Retirement Benefit Schemes Q3'2024 Performanc

As can be observed, the dataset above is a bit complex and diffucult to uderstand or analyze. This is because alot of information is contained in the reports.

```
# This cofirms all the records have are unique
len(all_cytonn_reports_df), len(all_cytonn_reports), len(all_cytonn_reports_df['id'].unique()
```

```
(664, 664, 664)
```

Explore and Clean the Dataset

The goal here is to extract the details of Effective/Norminal/Annual Rate of Money Market Funds (KES) and Assets Under Management for the entire schemes

Preview the Columns

```
all_cytonn_reports_df.iloc[0]
```

```
id                                                            748
slug                                      nairobi-metropolitan-area-9
name                Nairobi Metropolitan Area (NMA) Infrastructure...
author                                                Investments Team
researchdate                                                2024-12-22
active                                                           1
created_by                                                      95
series_id                                                        1
campaign                                                      2671
sent                                                             1
thumbnail                                                     None
created_at                                          2024-12-22 12:07:19
meta_title                                                    None
meta_keywords                                                 None
meta_description                                              None
og_description                                                None
url                 https://cytonnreport.com/research/nairobi-metr...
og_image            https://cytonnreport.com/storage/research/tmpp...
updated_at                                          2024-12-22 13:20:13
deleted_at                                                    None
title               Nairobi Metropolitan Area (NMA) Infrastructure...
category                                             Investment Reports
summary             <p><strong>Money Markets, T-Bills Primary Auct...
body                <p><strong>Money Markets, T-Bills Primary Auct...
date_holder                                                     22
date                                                22 December, 2024
creator                                                Fredrick Maore
status                                                        Active
series              {'id': 1, 'name': 'Cytonn Weekly', 'descriptio...
topics              [{'id': 2528, 'name': 'Fixed Income', 'slug': ...
Name: 0, dtype: object
```

Below is a tree structure of one record, to visualize the objects and their inner properties

```
json_structure = json2txttree(all_cytonn_reports[:1])
min_topics = min(len(i.get('topics', [])) for i in all_cytonn_reports)
```

```
max_topics = max(len(i.get('topics', [])) for i in all_cytonn_reports)
json_structure = json_structure.replace('   (array)', f'   (array) [{len(all_cytonn_reports)}
json_structure = json_structure.replace('"topics" (array)', f'"topics" (array) [between {min_

print(json_structure)
```

```
   (array) [664 items]
      (object)
        "id" (number)
        "slug" (string)
        "name" (string)
        "author" (string)
        "researchdate" (string)
        "active" (number)
        "created_by" (number)
        "series_id" (number)
        "campaign" (string)
        "sent" (number)
        "thumbnail" (number)
        "created_at" (string)
        "meta_title" (number)
        "meta_keywords" (number)
        "meta_description" (number)
        "og_description" (number)
        "url" (string)
        "og_image" (string)
        "updated_at" (string)
        "deleted_at" (number)
        "title" (string)
        "category" (string)
        "summary" (string)
        "body" (string)
        "date_holder" (number)
        "date" (string)
        "creator" (string)
        "status" (string)
        "series" (object)
          "id" (number)
          "name" (string)
          "description" (string)
          "thumbnail" (string)
          "created_by" (number)
```

```
      "category_id" (number)
      "created_at" (string)
      "updated_at" (string)
      "category" (object)
         "id" (number)
         "slug" (string)
         "name" (string)
         "created_at" (string)
         "updated_at" (string)
   "topics" (array) [between 0 - 8 items]
       (object)
         "id" (number)
         "name" (string)
         "slug" (string)
         "title" (number)
         "summary" (string)
         "body" (string)
         "research_id" (number)
         "active" (number)
         "topical" (number)
         "type" (number)
         "weight" (number)
         "created_by" (number)
         "created_at" (string)
         "updated_at" (string)
```

A full report is formed by articles. Each `topics` is a subsection, with `title` being the header and `body` being the content. We will merge all bodies from the articles to form the entire report HTML, which we will parse to extract the Money Market Funds yields tables. In addition, we are also going to add the main `body` and main `summary` and topics `summary` to encure we capture any table we might miss.

```
CYTONN_RECORD_LITERALS = Literal['summary', 'body', 'topics', 'researchdate']
def get_report_HTML(report: dict[CYTONN_RECORD_LITERALS, Any]) -> str:
    summary_html = report['summary']
    body_html = report['body']
    topics_html = ''.join([f"{i['summary']} \n\n {i['body']}" for i in report['topics']])
    return f"{summary_html} \n {body_html} \n {topics_html}"

# from IPython.display import HTML
# HTML(get_report_HTML(all_cytonn_reports[0]))
```

Parsing Dates

There are some summary tables that have dates such as `Q1'2023`, `Q1'2023 (%)`, `FY'2023`, `FY'2023 (%)`, `Q1'2024`, `Q1'2024 (%)`

```
await web_screenshot_async(
    # URL to take a screenshot of
    "https://cytonnreport.com/research/q1-2024-unit-trust-funds-performance-note",
    # Action deciding WHAT (element) or WHEN (eg: click) to take the screenshot
    action = select_table_by_title('Cytonn Report: Assets Under Management (AUM) for the Appr
    width = 700,
    screenshot_options = None,
    crop_options = { 'bottom': 400 })
```

Cytonn Report: Assets Under Management (AUM) for the Approved Collective Investment Schemes

No.	Collective Investment Schemes	FY'2023 AUM (Kshs mn)	FY'2023 Market Share	Q1'2024 AUM (Kshs mn)	Q1'2024 Market Share	AUM Growth FY'2023 – Q1'2024
1	CIC Unit Trust Scheme	63,331.6	29.4%	61,916.2	27.5%	(2.2%)
2	NCBA Unit Trust Scheme	30,934.4	14.4%	31,287.8	13.9%	1.1%
3	British American Unit Trust Scheme	31,707.1	14.7%	30,029.5	13.3%	(5.3%)
4	Sanlam Unit Trust Scheme	25,126.7	11.7%	29,699.2	13.2%	18.2%
5	ICEA Unit Trust Scheme	15,953.3	7.4%	15,654.8	6.9%	(1.9%)
6	Old Mutual Unit Trust Scheme	10,661.8	5.0%	11,831.7	5.3%	11.0%

Below function will help parse such time ranges:

```
def get_date_range(month, year):
    # Convert month name to number
    month_num = [i.lower() for i in month_abbr].index(month.lower())\
        if len(month) > 0 and any([i.lower() == month.lower() for i in month_abbr])\
        else datetime.strptime(month, '%B').month
```

```
    # Get the last day of the month
    _, last_day = monthrange(int(year), month_num)
    # Create date objects for the first and last day of the month
    start_date = date(int(year), month_num, 1)
    end_date = date(int(year), month_num, last_day)
    return start_date.strftime('%Y-%m-%d'), end_date.strftime('%Y-%m-%d')
parse_date_pattern_months = (
    "JAN|JANUARY|FEB|FEBRUARY|MAR|MARCH|APR|APRIL|MAY|"
    "JUN|JUNE|JUL|JULY|AUG|AUGUST|SEP|SEPTEMBER|"
    "OCT|OCTOBER|NOV|NOVEMBER|DEC|DECEMBER"
)
parse_date_pattern = rf'(?:(\d{{2}})[_|\s|-]*)?({parse_date_pattern_months})[_|\s|-]*(\d{{4}}

def parse_fiscal_period_dates(date_string: str) -> (tuple[str, str] | None):
    """
    This function parses a date string representing a fiscal period
    (Fiscal/Financial Year, Quarter, or Half-year), year, or date and returns the correspondi
    start and end dates.
    """
    extracted_date = re.search('^' + parse_date_pattern + '$', date_string, re.IGNORECASE)
    if extracted_date:
        search_date, search_month, search_year = extracted_date.groups()
        if search_date:
            month_num = datetime.strptime(search_month, '%B').month
            return datetime(int(search_year), month_num, int(search_date)).strftime('%Y-%m-%
        else:
            return get_date_range(search_month, search_year)
    if re.match(r"^\d{4}$", date_string, re.IGNORECASE):
        date_string = f"FY'{date_string}"
    # Define a regex pattern to match fiscal periods (FY, Q1-Q4, H1-H2) followed by a year, o
    pattern = r"^(FY|Q[1-4]|H[1-2])['|_|\s]*(\d{4})$"
    # Try to match the input string against the pattern
    match = re.match(pattern, date_string, re.IGNORECASE)
    # If no match is found, return None
    if not match:
        return None
    # Extract the period type and year from the match
    period, year = match.groups()
    year = int(year)
    # Handle Fiscal Year (FY) case
    if period.upper() == 'FY':
        start_date = datetime(year, 1, 1)
```

```
        end_date = datetime(year, 12, 31)
    # Handle Quarter (Q1-Q4) cases
    elif period.upper().startswith('Q'):
        quarter = int(period[1])
        start_month = (quarter - 1) * 3 + 1
        start_date = datetime(year, start_month, 1)
        # Calculate end date of the quarter
        end_date = start_date.replace(month=start_month + 2) + timedelta(days=32)
        end_date = end_date.replace(day=1) - timedelta(days=1)
    # Handle Half-year (H1-H2) cases
    elif period.upper().startswith('H'):
        half = int(period[1])
        start_month = (half - 1) * 6 + 1
        start_date = datetime(year, start_month, 1)
        # Calculate end date of the half-year
        end_date = start_date.replace(month=start_month + 5) + timedelta(days=32)
        end_date = end_date.replace(day=1) - timedelta(days=1)
    # Return start and end dates formatted as strings
    return (start_date.strftime('%Y-%m-%d'), end_date.strftime('%Y-%m-%d'))

def TEST_parse_fiscal_period_dates():
    # Test the function
    test_dates = [
        "FY'2019", "Q1'2020", "H1'2019", "fy 2018", "q32021", "h2_2022", "2020", '2019',
        'JUNE_2020', '01_NOVEMBER_2017', "H3'2020"
    ]
    for expanding_value in test_dates:
        result = parse_fiscal_period_dates(expanding_value)
        if result:
            print(f"{expanding_value}: {result}")
        else:
            print(f"{expanding_value}: Invalid format")
TEST_parse_fiscal_period_dates()
```

```
FY'2019: ('2019-01-01', '2019-12-31')
Q1'2020: ('2020-01-01', '2020-03-31')
H1'2019: ('2019-01-01', '2019-06-30')
fy 2018: ('2018-01-01', '2018-12-31')
q32021: ('2021-07-01', '2021-09-30')
h2_2022: ('2022-07-01', '2022-12-31')
2020: ('2020-01-01', '2020-12-31')
2019: ('2019-01-01', '2019-12-31')
```

```
JUNE_2020: ('2020-06-01', '2020-06-30')
01_NOVEMBER_2017: 2017-11-01
H3'2020: Invalid format
```

Parsing a Effective Annual Rate(KES Money Market Fund) and Total Assets Under Management (Collective Investment Schemes)

The `Extracted_Scheme_Entry` class below represents and validates a financial record entry. It validates `record type` (*Assets Under Management* or *Effective Annual Rate*), `date`, `value`, and `scheme`. The class also maintains lists of non-existent schemes and invalid records. Assets under management (AUM) is the market value of the investments managed by the fund manager on behalf of clients, inluding MMF, FIXED, balanced, equity, etc.

```python
class Extracted_Scheme_Entry:
    """
    A class to represent and validate financial entry information.

    Class Attributes:
    INVALID_FUNDS (list[str]): Stores funds not found in the mapping.
    INVALID_DATES (list[str]): Stores entry dates not valid.
    INVALID_VALUES (list[str]): Stores entry values not valid.
    TYPE_ASSETS_UNDER_MANAGEMENT (str): Constant for Assets Under Management type.
    TYPE_EFFECTIVE_ANNUAL_RATE (str): Constant for Effective Annual Rate type.
    """
    INVALID_SCHEMES: list[str] = []
    INVALID_DATES: list[str] = []
    INVALID_VALUES: list[str] = []
    TYPE_ASSETS_UNDER_MANAGEMENT: str = 'ASSETS_UNDER_MANAGEMENT' # Assets Under Management
    TYPE_EFFECTIVE_ANNUAL_RATE: str = 'EFFECTIVE_ANNUAL_RATE' # Effective Annual Rate

    def __init__(self,
                 entry_type: Literal['ASSETS_UNDER_MANAGEMENT', 'EFFECTIVE_ANNUAL_RATE'],
                 entry_date: str,
                 entry_value: str,
                 entry_scheme: str,
                 scheme_filter_function: Callable[[str], list[str]]):
        """
        Initialize a RecordEntry instance with validated attributes.

        Args:
        entry_type (str): Type of the record (TYPE_ASSETS_UNDER_MANAGEMENT or TYPE_EFFECTIVE_
        entry_date (str): Date of the record (2024-03-01) or Financial period (H1'2024).
```

```
        entry_value (str): Value of the record.
        entry_scheme (str): Name of the MMF(KES) fund
        fund_manager_filter_predicate (Callable): A predicate to filter and return matched MF
        """
        self.entry_type = Extracted_Scheme_Entry.validate_type(entry_type)
        self.entry_date = Extracted_Scheme_Entry.validate_date(entry_date)
        self.entry_value = Extracted_Scheme_Entry.validate_value(entry_value)
        self.entry_scheme = Extracted_Scheme_Entry.validate_scheme(entry_scheme, scheme_filte
        if self.entry_date is None:
            Extracted_Scheme_Entry.INVALID_DATES.append(entry_date)
        if self.entry_value is None:
            Extracted_Scheme_Entry.INVALID_VALUES.append(entry_value)
        if self.entry_scheme is None:
            Extracted_Scheme_Entry.INVALID_SCHEMES.append(entry_scheme)

    def is_valid(self) -> bool:
        """
        Check if the record is valid (all attributes are non-empty).
        """
        is_valid = \
            bool(self.entry_type) \
            and bool(self.entry_date) \
            and bool(self.entry_value) \
            and bool(self.entry_scheme)
        return is_valid

    @staticmethod
    def validate_scheme(value: str, filter_predicate: Callable[[str], list[str]]) -> str|None
        """
        Validate and standardize the date or financial period.
        """
        try:
            value = str(value or '').lower()
            # These represent USD MMF's
            EXCLUDES = ['Dollar', 'USD']
            is_USD_MMF = any((exclude.lower() in value) for exclude in EXCLUDES)
            if not is_USD_MMF:
                names = filter_predicate(value)
                if len(names) == 1:
                    return names[0]
                if len(names) > 1:
                    print(f'"{value}" has more that two matches! {names}')
```

```
            return None
        except:
            return None

    @staticmethod
    def validate_date(value: str) -> str|tuple[str, str]|None:
        """
        Validate and standardize the date
        """
        try:
            return parse_fiscal_period_dates(value)\
                    or datetime.strptime(value, "%Y-%m-%d").strftime('%Y-%m-%d')\
                    or None
        except:
            return None

    @staticmethod
    def validate_value(value: str|float) -> str|None:
        """
        Validate and clean the entry value.
        """
        try:
            if type(value) == float:
                    return value
            # remove percentage sign
            value = value.rstrip('%')
            # remove comma and white space
            value = ''.join([i for i in value if i not in [' ', ',', '-']])
            return float(value) if len(value) > 0 else None
        except:
            return None

    @staticmethod
    def validate_type(value: str) -> Literal['ASSETS_UNDER_MANAGEMENT', 'EFFECTIVE_ANNUAL_RA
        """
        Validate the record type.

        Args:
        value (str): The record type to validate.

        Raises:
        TypeError exception.
```

```
        """
        value = (value or '').upper()
        if value in [Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT, Extracted_Scheme_E
            return value
        raise TypeError(f"{value} is not proper entry Type!")

def TEST_MoneyMarketFund_KES_RecordEntry():
    # Test the class
    test_cases = [
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
            "entry_date": "2024-03-01",
            "entry_value": "1,000,000",
            "entry_scheme": "britam",
        },
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
            "entry_date": "H1'2024",
            "entry_value": "5.5%",
            "entry_scheme": "old mutual",
        },
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
            "entry_date": "invalid-date",
            "entry_value": "1,000,000",
            "entry_scheme": "sanlam",
            "invalid": "invalid date"
        },
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
            "entry_date": "2024-03-01",
            "entry_value": "invalid-value",
            "entry_scheme": "britam",
            "invalid": "invalid value"
        },
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
            "entry_date": "2024-03-01",
            "entry_value": "1,000,000",
            "entry_scheme": "unknown scheme",
            "invalid": "unmapped scheme"
        },
```

```
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
            "entry_date": "2024-03-01",
            "entry_value": "1,000,000",
            "entry_scheme": "britam sanlam",
            "invalid": "2 funds matched"
        },
        {
            "entry_type": Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
            "entry_date": "2024-03-01",
            "entry_value": "1,000,000",
            "entry_scheme": "Britam USD Dollar Fund",
            "invalid": "USD MMF"
        },
    ]
    # Define the fund filter function
    test_fund_map = [
        (
            'Britam MMF(KES)',
            ['britam', 'british-american', 'british', 'american']
        ),
        (
            'UAP Old Mutual MMF(KES)',
            ['old mutual', 'uap old mutual', 'uap']
        ),
        (
            'Sanlam MMF(KES)',
            ['sanlam', 'sanlam investments']
        )
    ]

    def test_fund_filter(value: str):
        value = value.lower()
        names = [name for name, aliases in test_fund_map if any(alias in value for alias in a
        return names

    # Run tests
    for test_case in test_cases:
        entry = Extracted_Scheme_Entry(
            test_case["entry_type"],
            test_case["entry_date"],
            test_case["entry_value"],
```

```
            test_case["entry_scheme"],
            test_fund_filter
        )
        cases = [entry.entry_date, entry.entry_value, entry.entry_scheme]
        invalid = f" ({test_case.get('invalid')})" if test_case.get('invalid') else ''
        print(f"Valid: {entry.is_valid()}{invalid}, {cases}")

    # Print invalid entries
    print("\nInvalid Funds:", Extracted_Scheme_Entry.INVALID_SCHEMES)
    print("\nInvalid Dates:", Extracted_Scheme_Entry.INVALID_DATES)
    print("\nInvalid Values:", Extracted_Scheme_Entry.INVALID_VALUES)
    Extracted_Scheme_Entry.INVALID_SCHEMES = []
    Extracted_Scheme_Entry.INVALID_DATES = []
    Extracted_Scheme_Entry.INVALID_VALUES = []
TEST_MoneyMarketFund_KES_RecordEntry()
```

```
Valid: True, ['2024-03-01', 1000000.0, 'Britam MMF(KES)']
Valid: True, [('2024-01-01', '2024-06-30'), 5.5, 'UAP Old Mutual MMF(KES)']
Valid: False (invalid date), [None, 1000000.0, 'Sanlam MMF(KES)']
Valid: False (invalid value), ['2024-03-01', None, 'Britam MMF(KES)']
Valid: False (unmapped scheme), ['2024-03-01', 1000000.0, None]
"britam sanlam" has more that two matches! ['Britam MMF(KES)', 'Sanlam MMF(KES)']
Valid: False (2 funds matched), ['2024-03-01', 1000000.0, None]
Valid: False (USD MMF), ['2024-03-01', 1000000.0, None]

Invalid Funds: ['unknown scheme', 'britam sanlam', 'Britam USD Dollar Fund']

Invalid Dates: ['invalid-date']

Invalid Values: ['invalid-value']
```

We then create a fund collective schemes map with a tuple of `name` and `aliases` because the records don't have a simple or stardard naming in the Cytonn reports. As such, we need to use very unique and simple names that we can use to match abitrary Money market funds names from the crawled data.

```
SCHEME_NAME_ALIAS_MAP = [
    # The African Alliance (AA) Kenya Shillings Fund is a money market fund by
    # African Alliance Kenya Investment Bank Limited (the fund manager)
    # launched on 27th April 2015.
    # https://centwarrior.com/aa-kenya-shillings-fund/
```

```
    # https://www.linkedin.com/posts/centwarrior_aa-kenya-shillings-fund-explained-in-2024-a
    # https://cytonn.com/topicals/investment-risk-analysis
    (
        'African Alliance Kenya Unit Trust Scheme',
        ['african', 'alliance', 'aa kenya']
    ),
    (
        'British-American Unit Trust Scheme',
        ['britam', 'british-american', 'british', 'american']
    ),
    (
        'NCBA Unit Trust Funds',
        ['ncba', 'cba', 'commercial bank of africa']
    ),
    (
        'Zimele Unit Trust Scheme',
        ['zimele']
    ),
    (
        'ICEA Unit Trust Scheme',
        ['icea']
    ),
    (
        'Standard Investment Trust Funds',
        ['standard', 'mansa']
    ),
    (
        'CIC Unit Trust Scheme',
        ['cic']
    ),
    (
        'Madison Unit Trust Fund',
        ['Madison', 'madisson']
    ),
    (
        'Dyer and Blair Unit Trust Scheme',
        ['dyer', 'blair']
    ),
    (
        'Amana Unit Trust Funds Scheme',
        ['amana']
    ),
```

```
    (
        'Diaspora Unit Trust Scheme',
        ['diaspora']
    ),
    (
        'First Ethical Opportunities Fund',
        ['ethical',
         # 'first', 'opportunities'
        ]
    ),
    # https://www.cma.or.ke/licensees-market-players/
    # https://genghis-capital.com/asset-management/money-market-fund/
    (
        'Genghis Unit Trust Funds',
        ['hela','genghis', 'hazina', 'hisa', 'iman', 'gencap', 'compliant', 'eneza', 'genCap
    ),
    # https://www.businessdailyafrica.com/bd/markets/capital-markets/safaricom-s-mali-unit-t
    (
        'Mali Money Market Fund',
        ['mali']
    ),
    # https://www.safaricom.co.ke/main-mpesa/m-pesa-services/wealth/ziidi
    # https://www.cma.or.ke/cma-approves-ziidi-money-market-fundfrom-safaricom-plc/
    (
        'Ziidi Money Market Fund',
        ['ziidi']
    ),
    # https://www.cma.or.ke/cma-approves-establishment-of-new-unit-trust-sub-funds/
    (
        'Gulfcap Money Market Fund',
        ['gulfcap']
    ),
    (
        'Sanlam Unit Trust Scheme',
        ['sanlam']
    ),
    (
        'Nabo Africa Funds',
        ['nabo']
    ),
    (
        'Old Mutual Unit Trust Scheme',
```

```
        ['mutual', 'old', 'Faulu']
    ),
    # https://equitygroupholdings.com/ke/investor-relations/eib
    # https://www.cma.or.ke/licensees-market-players/
    (
        'Equity Investment Bank Collective Investment Scheme',
        ['equity']
    ),
    # https://www.cma.or.ke/licensees-market-players/
    (
        'Dry Associates Unit Trust Scheme',
        ['dry associates', 'dry', 'associates']
    ),
    (
        'Co-op Trust Fund',
        ['co-op', 'gratuity', 'Coop']
    ),
    (
        'Apollo Unit Trust Scheme',
        ['aggressive', 'apollo']
    ),
    (
        'Cytonn Unit Trust Scheme',
        ['cytonn']
    ),
    (
        'Orient Umbrella Collective Investment Scheme (formerly Alphafrica Umbrella Fund)',
        ['orient', 'kasha', 'alpha', 'alphafrica']
    ),
    (
        'Wanafunzi Investment Unit Trust Fund',
        ['wanafunzi']
    ),
    (
        'Absa Unit Trust Funds',
        ['absa']
    ),
    (
        'Jaza Unit Trust Fund',
        ['jaza']
    ),
    (
```

```
        'Masaru Unit Trust Scheme',
        ['masaru']
    ),
    (
        'ADAM Unit Trust Scheme',
        ['adam']
    ),
    (
        'KCB Unit Trust Scheme (formerly Natbank Unit Trust Scheme)',
        ['kcb', 'natbank']
    ),
    (
        'GenAfrica Unit Trust Scheme',
        ['genafrica']
    ),
    (
        'Amaka Unit Trust (Umbrella) Scheme',
        ['amaka']
    ),
    (
        'Jubilee Unit Trust Collective Investment Scheme',
        ['jubilee']
    ),
    # Previusly "Liberty Pension Services Limited"
    # https://enwealth.co.ke/about/#governance
    # https://www.linkedin.com/company/enwealth-kenya/?originalSubdomain=ke
    # https://enwealth.co.ke/capital/enwealth-money-market-fund/
    (
        'Enwealth Capital Unit Trust Scheme',
        ['enwealth']
    ),
    (
        'Kuza Asset Management Unit Trust Scheme',
        ['kuza', 'momentum']
    ),
    # https://www.linkedin.com/company/arvocap-asset-managers/
    # https://www.businessdailyafrica.com/bd/markets/avocarp-latest-to-enter-kenya-s-asset-ma
    (
        'Arvocap Unit Trust Scheme',
        ['arvocap']
    ),
    (
```

```
        'Etica Capital Limited',
        ['etica']
    ),
    # https://licensees.cma.or.ke/licenses/15/
    (
        'Mayfair umbrella Collective investment scheme',
        ['mayfair']
    ),
    (
        'Lofty Corban Unit Trust Scheme',
        ['lofty-corban', 'lofty', 'corban']
    ),
    (
        'CPF Unit Trust Funds',
        ['cpf', 'cpof']
    ),
    (
        'Stanbic Unit Trust Funds',
        ['stanbic']
    ),
    (
        'MyXENO Unit Trust Scheme',
        ['myxeno']
    ),
    ##############################################
    ##### UNVERIFIED COLLECTIVE INVESTMENTS #####
    ##############################################
    (
        'Metropolitan Canon Asset Managers Limited',
        ['metropolitan']
    ),
    (
        'FCB Capital Limited',
        ['fcb']
    ),
    (
        'Fusion Investment Management Limited',
        ['fusion']
    ),
    (
        'Altree Capital Kenya Limited',
        ['altree']
```

```
    ),
    (
        'CFS Asset Management Limited',
        ['cfs']
    ),
    (
        'I&M Capital Limited',
        ['i&m']
    ),
    (
        'Globetec Asset Managers Limited',
        ['globetec']
    ),
    (
        # https://cytonnreport.com/research/cytonn-q3-2024-markets-review
        # https://www.businessdailyafrica.com/bd/markets/capital-markets/fintech-start-up-nd
        'Waanzilishi Capital Limited',
        ['waanzilishi', 'ndovu']
    ),
    (
        'Star Capital Management Limited',
        ['star']
    ),
    # Unverified and NO online presense!
    (
        'Stanlib Kenya',
        ['stanlib']
    ),

]
SCHEME_NAME_ALIAS_MAP
```

```
def scheme_filter(value: str) -> list[str]:
    value = value.lower()
    names = [
        name
        for name, aliases
        in SCHEME_NAME_ALIAS_MAP if any((alias.lower() in value) for alias in aliases)
    ]
    return names

# Test
```

```
def TEST_scheme_filter(fund_manager: str):
    name = scheme_filter(fund_manager)
    print(f"{fund_manager} => {name}")
TEST_scheme_filter('KCB Fund Managers')
TEST_scheme_filter('Cytonn Fund Mangers')
TEST_scheme_filter('Nabo')
TEST_scheme_filter('madison')
```

One report can have more than one table, see: https://cytonnreport.com/research/unit-trust-fund-performance-q3-1. The list below contains a tuple of:

1. A list if table names in the records. For matching, the table names are normalized with the below function:

```
def normalize_and_compare_two_strs(str1: str, str2:str) -> bool:
    if not str1 or not str2:
        return False
    str1 = str1.strip().upper()
    str2 = str2.strip().upper()
    return str1 == str2 or hacky_normalizer(str1) == hacky_normalizer(str2)

# Test
def TEST_normalize_and_match_two_strs():
    str1_str2 = [
        ('q2’2020-aum(kshs-mns)', 'Q2 2020_AUM(KSHs_MNs)'),
        ('q2’2020-aum(kshs-mns)', "Q2_2020_AUM(KSHs_MNs)"),
        ('', ""),
        (None, None),
        ('no.', 'NO.')
    ]
    for str1, str2 in str1_str2:
        is_match = normalize_and_compare_two_strs(str1, str2)
        print(f'IS_MATCH={is_match}; {[str1, str2]}')

TEST_normalize_and_match_two_strs()
```

```
IS_MATCH=True; ['q2’2020-aum(kshs-mns)', 'Q2 2020_AUM(KSHs_MNs)']
IS_MATCH=True; ['q2’2020-aum(kshs-mns)', 'Q2_2020_AUM(KSHs_MNs)']
IS_MATCH=False; ['', '']
IS_MATCH=False; [None, None]
IS_MATCH=True; ['no.', 'NO.']
```

2. A list of functions that process the matched tables. Each function should essentially process a single column. The function receives three parameters: a dataframe row data (a table row entry), the entire record from which the table was extracted from, and any `other_params`. This first parameter is useful to capture the entry value, and the second is important to capture the date of the record if not provided in the table. The table row entry values are a dictionally named with the table names used to match the tables. Callback function returns a `Extracted_Scheme_Entry`

```
remove_KSH_MNS = lambda str1: re.sub(r'(?:_AUM)?_?\(KSHS?_MNS?\)', '', str1, flags=re.IGNOREC
remove_MONEY_MARKET_FUND_KSHS_MNS = lambda str1: re.sub(r'_?MONEY_MARKET_FUNDS?_?\(KSHS?_MNS
remove_EFFECTIVE_ANNUAL = lambda str1: re.sub(r'_AVERAGE_EFFECTIVE_ANNUAL_YIELD_P_A_|EFFECTI
remove_MONEY_MARKET_FUND_KSHS_MNS_2= lambda str1: re.sub(r'_?MONEY_MARKET_FUNDS?_AUM_?\(KSHS
EXTRACTION_MAP: list[
    tuple[
        list[str],
        list[Callable[[dict[str, Any], dict[CYTONN_RECORD_LITERALS, Any]], Extracted_Scheme_
    ]] = [
    (
        [
            'RANK',
            'FUND_MANAGER',
            'DAILY_YIELD',
            'EFFECTIVE_ANNUAL_RATE'
        ],
        [
            # Effective Annual Rate and Daily Yield
            # https://cytonnreport.com/research/cytonn-monthly-october-2021
            # https://cytonnreport.com/research/potential-effects-covid-19
            lambda row, record: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                record['researchdate'],
                row['EFFECTIVE_ANNUAL_RATE'],
                row['FUND_MANAGER'],
                scheme_filter),
            lambda row, record: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                record['researchdate'],
                row['DAILY_YIELD'],
                row['FUND_MANAGER'],
                scheme_filter),
        ]
    ),
    (
```

```
        [
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: normalized in ['RANK', hacky_normalizer('NO.
            },
            {
                'column_name': 'SCHEME',
                'predicate': lambda normalized: bool(re.match(r'MONEY_MARKET_FUNDS?|FUND_MANA
            },
            {
                'column_name': 'EFFECTIVE_ANNUAL',
                'predicate': lambda normalized: normalized in ['EFFECTIVE_ANNUAL', 'EFFECTIVE
            },
        ],
        [
            # https://cytonnreport.com/research/kenyas-fy2024-2025-budget
            # https://cytonnreport.com/research/nairobi-metropolitan-area-serviced-apartments
            # https://cytonnreport.com/research/cytonn-monthly-may-2024
            # https://cytonnreport.com/research/q12023-unit-trust-funds-performance-cytonn-mo
            # https://cytonnreport.com/research/investing-in-unit
            lambda row, record: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                record['researchdate'],
                row['EFFECTIVE_ANNUAL'],
                row['SCHEME'],
                scheme_filter)
        ]
    ),
    (
        [
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: normalized in ['RANK', hacky_normalizer('NO.
            },
            {
                'column_name': 'SCHEME',
                'predicate': lambda normalized: bool(re.match(r'MONEY_MARKET_FUNDS?|FUND_MANA
            },
            {
                'column_name': 'EFFECTIVE_ANNUAL',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_EFFECTI
                'other_params':
```

```
                {
                    'DATE_1': lambda normalized: remove_EFFECTIVE_ANNUAL(normalized)
                }
            },
        ],
        [
            # https://cytonnreport.com/research/unit-trust-fund-performance-q3-1
            # https://cytonnreport.com/research/fy2019-utf-performance
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                other_params['DATE_1'],
                row['EFFECTIVE_ANNUAL'],
                row['SCHEME'],
                scheme_filter)
        ]
    ),
    (
        [
            {
                'column_name': 'SCHEME',
                'predicate': lambda normalized: 'AVERAGE' in normalized,
            },
            {
                'column_name': 'DATE_1_EAR',
                'predicate': lambda normalized: bool(re.match(r"^\d{4}$", normalized)),
                'other_params':
                {
                    'DATE_1': lambda normalized: normalized.strip()
                }
            },
            {
                'column_name': 'DATE_2_EAR',
                'predicate': lambda normalized: bool(re.search(parse_date_pattern, normalized
                'other_params':
                {
                    'DATE_2': lambda normalized: re.search(parse_date_pattern, normalized).gr
                }
            },
            {
                'column_name': 'DATE_3_EAR',
                'predicate': lambda normalized: bool(re.search(parse_date_pattern, normalized
                'other_params':
```

```
                {
                    'DATE_3': lambda normalized: re.search(parse_date_pattern, normalized).gr
                }
            },
        ],
        [
            # https://cytonnreport.com/research/cmmf-fact-sheet-june-2020
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                other_params['DATE_1'],
                row['DATE_1_EAR'],
                row['SCHEME'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                other_params['DATE_2'],
                row['DATE_2_EAR'],
                row['SCHEME'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE,
                other_params['DATE_3'],
                row['DATE_3_EAR'],
                row['SCHEME'],
                scheme_filter),
        ]
    ),
    (
        [
            'NO.',
            'UNIT_TRUST_FUND_MANAGER',
            'AUM',
            '%_OF_MARKET_SHARE'
        ],
        [
            # https://cytonnreport.com/research/investment-options-in-kenyan-market
            lambda row, record: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                record['researchdate'],
                row['AUM'],
                row['UNIT_TRUST_FUND_MANAGER'],
                scheme_filter),
```

```
        ]
    ),
    (
        [
            'NO.',
            'FUND_MANAGERS',
            {
                'column_name': 'DATE_1_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(normalized.rep
                'other_params':
                {
                    'DATE_1': lambda normalized: normalized.replace('_AUM(KSHS_MNS)', '')
                }
            },
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: 'MARKET_SHARE' in normalized
            },
            {
                'column_name': 'DATE_2_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(normalized.rep
                'other_params':
                {
                    'DATE_2': lambda normalized: normalized.replace('_AUM(KSHS_MNS)', '')
                }
            },
            {
                'column_name': 'UNWANTED_2',
                'predicate': lambda normalized: 'MARKET_SHARE' in normalized
            },
            {
                'column_name': 'UNWANTED_3',
                'predicate': lambda normalized: 'AUM_GROWTH' in normalized
            },
        ],
        [
            # https://cytonnreport.com/research/unit-trust-funds-performance-q2-2020
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_1'],
                row['DATE_1_AUM'],
                row['FUND_MANAGERS'],
```

```
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_2'],
                row['DATE_2_AUM'],
                row['FUND_MANAGERS'],
                scheme_filter),
        ]
    ),
    (
        [
            'NO.',
            'FUND_MANAGERS',
            {
                'column_name': 'DATE_1_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_MONEY_
                'other_params':
                {
                    'DATE_1': lambda normalized: remove_MONEY_MARKET_FUND_KSHS_MNS(normalize
                }
            },
            {
                'column_name': 'DATE_2_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_MONEY_
                'other_params':
                {
                    'DATE_2': lambda normalized: remove_MONEY_MARKET_FUND_KSHS_MNS(normalize
                }
            },
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: '_MARKET_SHARE' in normalized
            },
            {
                'column_name': 'UNWANTED_2',
                'predicate': lambda normalized: '_MARKET_SHARE' in normalized,
                'optional': True
            },
            {
                'column_name': 'UNWANTED_3',
                'predicate': lambda normalized: 'VARIANCE' in normalized,
                'optional': True
```

```
            },
        ],
        [
            # https://cytonnreport.com/research/fy2019-utf-performance
            # https://cytonnreport.com/research/investing-in-unit
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_1'],
                row['DATE_1_AUM'],
                row['FUND_MANAGERS'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_2'],
                row['DATE_2_AUM'],
                row['FUND_MANAGERS'],
                scheme_filter),
        ]
    ),
    (
        [
            'NO.',
            {
                'column_name': 'SCHEME',
                'predicate': lambda normalized: normalized in ['FUND_MANAGERS', 'FUND_MANAGE
            },
            {
                'column_name': 'DATE_1_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_KSH_MN
                'other_params':
                {
                    'DATE_1': lambda normalized: remove_KSH_MNS(normalized)
                }
            },
            {
                'column_name': 'DATE_2_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_KSH_MN
                'other_params':
                {
                    'DATE_2': lambda normalized: remove_KSH_MNS(normalized)
                }
            },
```

```
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: 'GROWTH' in normalized
            },
        ],
        [
            # https://cytonnreport.com/research/unit-trust-funds-perfomance-q1-2020-cytonn-we
            # https://cytonnreport.com/research/unit-trust-funds-performance
            # https://cytonnreport.com/research/fy2019-utf-performance
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_2'],
                row['DATE_1_AUM'],
                row['SCHEME'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_2'],
                row['DATE_2_AUM'],
                row['SCHEME'],
                scheme_filter),
        ]
    ),
    (
        [
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: bool(normalized in [hacky_normalizer('#'), ha
            },
            'FUND_MANAGERS',
            {
                'column_name': 'DATE_1_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_MONEY_
                'other_params':
                {
                    'DATE_1': lambda normalized: remove_MONEY_MARKET_FUND_KSHS_MNS_2(normaliz
                }
            },
            {
                'column_name': 'DATE_2_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_MONEY_
                'other_params':
```

```
                {
                    'DATE_2': lambda normalized: remove_MONEY_MARKET_FUND_KSHS_MNS_2(normaliz
                }
            },
            {
                'column_name': 'DATE_3_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_MONEY_M
                'other_params':
                {
                    'DATE_3': lambda normalized: remove_MONEY_MARKET_FUND_KSHS_MNS_2(normaliz
                }
            },
            {
                'column_name': 'UNWANTED_2',
                'predicate': lambda normalized: bool('ANNUALIZED' in normalized)
            },
        ],
        [
            # https://cytonnreport.com/research/options-for-your-pension
            # https://cytonnreport.com/research/cytonn-monthly-august-2019
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_1'],
                row['DATE_1_AUM'],
                row['FUND_MANAGERS'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_1'],
                row['DATE_2_AUM'],
                row['FUND_MANAGERS'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_1'],
                row['DATE_3_AUM'],
                row['FUND_MANAGERS'],
                scheme_filter),
        ]
    ),
    (
        [
```

```
            {
                'column_name': 'UNWANTED_1',
                'predicate': lambda normalized: 'NO' in normalized,
            },
            {
                'column_name': 'SCHEME',
                'predicate': lambda normalized: normalized in ['COLLECTIVE_INVESTMENT_SCHEMES
            },
            {
                'column_name': 'DATE_1_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_KSH_MNS
                'other_params':
                {
                    'DATE_1': lambda normalized: remove_KSH_MNS(normalized)
                }

            },
            {
                'column_name': 'UNWANTED_2',
                'predicate': lambda normalized: bool(
                    parse_fiscal_period_dates(normalized.replace('MARKET_SHARE', ''))),

            },
            {
                'column_name': 'DATE_2_AUM',
                'predicate': lambda normalized: bool(parse_fiscal_period_dates(remove_KSH_MNS
                'other_params':
                {
                    'DATE_2': lambda normalized: remove_KSH_MNS(normalized)
                }

            },
            {
                'column_name': 'UNWANTED_3',
                'predicate': lambda normalized: bool(
                    parse_fiscal_period_dates(normalized.replace('MARKET_SHARE', ''))),

            },
            {
                'column_name': 'UNWANTED_4',
                'predicate': lambda normalized: 'AUM_GROWTH' in normalized,
            }
```

```
        ],
        [
            # https://cytonnreport.com/research/unit-trust-fund-performance-q3-1
            # https://cytonnreport.com/research/unit-trust-funds-performance-cytonn-monthly-j
            lambda row, _, other_params : Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_1'],
                row['DATE_1_AUM'],
                row['SCHEME'],
                scheme_filter),
            lambda row, _, other_params: Extracted_Scheme_Entry(
                Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT,
                other_params['DATE_2'],
                row['DATE_2_AUM'],
                row['SCHEME'],
                scheme_filter),
        ]
    )
]
```

```
def compare_table_and_given_column_names(
        given_table_columns,
        header_column_names,
        *,
        use_optional_columns = True):
    table_columns = [
        table_column
        for table_column
        in given_table_columns
        if type(table_column) == str
            or use_optional_columns
            or not table_column.get('optional', False)
    ]
    table_column_strs = [
        table_column if type(table_column) == str else table_column['column_name']
        for table_column
        in table_columns
    ]
    is_match_new = len(header_column_names) == len(table_columns) and \
        all([
                normalize_and_compare_two_strs(header_column_name, table_column)
                    if type(table_column) == str
```

```
                    else table_column['predicate'](hacky_normalizer(header_column_name))
                for header_column_name, table_column
                in zip(header_column_names, table_columns)
            ])
    if is_match_new or not use_optional_columns:
        other_params = {
            other_param_key: other_param_value(hacky_normalizer(header_column_name))
            for header_column_name, table_column
            in zip(header_column_names, table_columns)
            if type(table_column) != str and table_column.get('other_params', None)
            for other_param_key, other_param_value
            in table_column['other_params'].items()
        } if is_match_new else { }
        return is_match_new, table_column_strs, other_params
    return compare_table_and_given_column_names(
        given_table_columns,
        header_column_names,
        use_optional_columns = False)

def get_table(table: Tag, extraction_map):
    for tag in table.find_all(True):
        tag.attrs = {} # remove tags such as colspan and rowspan
    # Iterate through predefined extraction mappings
    for (given_table_columns, extractor_callbacks) in extraction_map:
        clean_up_tasks: list[Callable[[], None]] = []
        header_tr_s: list[Tag] = table.select('thead tr')
        is_match = False
        is_match_columns = []
        is_match_other_params = {}
        # Check if table headers match the expected columns
        for header_tr in header_tr_s:
            header_column_names: list[str] = [i.get_text(strip=True) for i in header_tr.find_
            is_match_new, pure_str_columns, other_params = compare_table_and_given_column_nam
                given_table_columns,
                header_column_names)
            if is_match_new:
                is_match_columns = pure_str_columns
                is_match_other_params = other_params or {}
            # If not a match, add to cleanup tasks.
            # We add cleap tasks here to delay deleting the table headers before we decide
            # that this table is matched. When the given columns are matched, the other colu
            # are deleted to ensure the dataframe has one column.
```

```
            if not is_match_new:
                clean_up_tasks.append(header_tr.extract)
            is_match = is_match or is_match_new
        # If a match is found, process the table
        if is_match:
            try:
                # Execute cleanup tasks
                [clean_up_task() for clean_up_task in clean_up_tasks]
                # Convert table to DataFrame
                table_df = pd.read_html(io.StringIO(str(table)))[0]
                table_df.columns = is_match_columns
                return (table_df, extractor_callbacks, is_match_other_params)
            except Exception as e:
                print('error', e, table)
                continue
    return (None, None, None)

def is_valid_dataframe(df: pd.DataFrame | None) -> bool:
    return df is not None and not df.empty

DEBUG_OPTIONS = dict[
    Literal[
        'log_unmatched_table',
        'log_invalid_columns',
        'log_extracted_valid',
        'log_extracted_invalid',
        'log_extractor_count',
    ],
    Callable[[str], None]
]

def get_tables(html: str, extraction_map, *, debug_options: DEBUG_OPTIONS = {}):
    log_unmatched_table = debug_options.get('log_unmatched_table')
    # Parse the HTML content using BeautifulSoup
    parsed_html = BeautifulSoup(html, "html.parser")
    # Find all <table> elements in the parsed HTML and store them in a list
    # remove duplicates
    tables: list[Tag] = list({ hacky_normalizer(str(table)): table for table in parsed_html.
    # Iterate over each table found in the HTML
    for table in tables:
        # Generate a DataFrame and a list of extractor callbacks for each table
        table_df, extractor_callbacks, other_params = get_table(copy(table), extraction_map)
```

```
        # Check if the DataFrame is valid and not None
        if is_valid_dataframe(table_df) and table_df is not None:
            # Yield the DataFrame and the associated callbacks
            yield (table_df, extractor_callbacks, other_params)
        else:
            if log_unmatched_table:
                log_unmatched_table(str(table))

def extract_frame_by_column_names(
        report: dict[CYTONN_RECORD_LITERALS, str],
        tables_html: str,
        extraction_map,
        *,
        debug_options: DEBUG_OPTIONS = {},
        ):
    log_invalid_columns = debug_options.get('log_invalid_columns')
    log_extractor_count = debug_options.get('log_extractor_count')
    log_extracted_valid = debug_options.get('log_extracted_valid')
    log_extracted_invalid = debug_options.get('log_extracted_invalid')
    # Generate tables and callbacks using the get_tables function
    table__callback__generator = get_tables(tables_html, extraction_map, debug_options=debug_
    # Iterate over each table DataFrame and its extractor callbacks
    for table_df, extractor_callbacks, other_params in table__callback__generator:
        if log_extractor_count:
            log_extractor_count(len(extractor_callbacks))
        if len(extractor_callbacks) > 0:
            # Apply each callback function to the rows of the table
            for extractor_callback in extractor_callbacks:
                table_rows = [
                    dynamic_callback(extractor_callback, raw_table_row.to_dict(), report, otl
                    for _, raw_table_row
                    in table_df.iterrows()
                ]
                # Convert the processed rows into a new DataFrame
                extracted_df = pd.DataFrame([vars(i) for i in table_rows if i.is_valid()])
                # Check if the extracted DataFrame is valid and yield it
                if is_valid_dataframe(extracted_df):
                    if log_invalid_columns:
                        __invalid_columns = [vars(i) for i in table_rows if not i.is_valid()]
                        if len(__invalid_columns) > 0:
                            log_invalid_columns(pd.DataFrame(__invalid_columns))
                    if log_extracted_valid:
```

```
                        log_extracted_valid(extracted_df)
                    yield extracted_df
                elif log_extracted_invalid:
                    log_extracted_invalid(table_df)

def extract_table_by_column_names(
        report: dict[CYTONN_RECORD_LITERALS, str],
        *,
        debug_options: DEBUG_OPTIONS = {}):
    # Get the HTML content of the report
    report_html = get_report_HTML(report)
    yield from extract_frame_by_column_names(
        report,
        report_html,
        EXTRACTION_MAP,
        debug_options = debug_options)
```

Validating the Parsing

Below code extracts and parses entries from all the records and stores various metrics for validating.

```
# Stores a tuple of the index of the record, the key for the log and the value logged
debug_log_store: list[tuple[str, str, Any]] = []
def extract_all_records():
    for index, report in tqdm(all_cytonn_reports_df.iterrows(), total=len(all_cytonn_reports_
        debug_options = {
            'log_unmatched_table':
                lambda value_str: debug_log_store.append((index, 'log_unmatched_table', value
            'log_invalid_columns':
                lambda value_df: debug_log_store.append((index, 'log_invalid_columns', value_
            'log_extracted_valid':
                lambda value_tuple_df: debug_log_store.append((index, 'log_extracted_valid',
            'log_extracted_invalid':
                lambda value_tuple_df: debug_log_store.append((index, 'log_extracted_invalid
            'log_extractor_count':
                lambda value_int: debug_log_store.append((index, 'log_extractor_count', value
        }
        yield from extract_table_by_column_names(report, debug_options = debug_options)
```

```
extracted_records_df = pd.concat(objs = extract_all_records(), ignore_index = True)

extracted_records_df
```

```
  0%|          | 0/664 [00:00<?, ?it/s]100%|     | 664/664 [01:16<00:00,  8.69it/s]
```

	entry_type	entry_date	entry_value	entry_scheme
0	EFFECTIVE_ANNUAL_RATE	2024-12-22	17.1	Cytonn Unit Trust Scheme
1	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.6	Lofty Corban Unit Trust S
2	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.3	Gulfcap Money Market Fu
3	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.2	Orient Umbrella Collective
4	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.1	Etica Capital Limited
...	...	...	...	...
5174	EFFECTIVE_ANNUAL_RATE	(2018-01-01, 2018-12-31)	11.5	Cytonn Unit Trust Scheme
5175	EFFECTIVE_ANNUAL_RATE	(2018-01-01, 2018-12-31)	10.2	Nabo Africa Funds
5176	EFFECTIVE_ANNUAL_RATE	(2018-01-01, 2018-12-31)	10.1	CIC Unit Trust Scheme
5177	EFFECTIVE_ANNUAL_RATE	(2018-01-01, 2018-12-31)	9.9	Madison Unit Trust Fund
5178	EFFECTIVE_ANNUAL_RATE	(2018-01-01, 2018-12-31)	9.9	Zimele Unit Trust Scheme

Validating Extraction Process

```
(len(Extracted_Scheme_Entry.INVALID_SCHEMES),
len(Extracted_Scheme_Entry.INVALID_DATES),
len(Extracted_Scheme_Entry.INVALID_VALUES))
```

```
(81, 0, 97)
```

INVALID_SCHEMES

```
np.unique(Extracted_Scheme_Entry.INVALID_SCHEMES)
```

```
array(['Absa Money Market Fund USD', 'Average',
       'Average of Top 5 Money Market Funds',
       'Benchmark ( Average 91 day T- Bill + 1.0% point)',
       'Benchmark (Average 182 day T- Bill + 5.0% points)',
       'CIC Dollar Fund', 'Cytonn Money Market Fund USD',
       'Dry Associates Money Market Fund USD', 'Industrial Average',
       'Industry average', 'KCB Money Market Fund USD',
```

```
       'Kuza Money Market Fund USD', 'Lofty-Corban Money Market Fund USD',
       'Nabo Africa Money Market Fund USD',
       'Old Mutual Dollar Money Market Fund', 'Sanlam Dollar Fund',
       'Total', 'Weighted Average Growth', 'nan'], dtype='<U49')
```

```
all(
    re.match(r".*(total|usd|average|dollar).*|nan", str(i), flags=re.IGNORECASE)
    for i
    in Extracted_Scheme_Entry.INVALID_SCHEMES)
```

```
True
```

INVALID_VALUES

```
np.unique(Extracted_Scheme_Entry.INVALID_VALUES).tolist()
```

```
['-']
```

Lets look at unmatched tables

```
grouped_by_key_id = groupby(lambda x: (x[0], x[1]), debug_log_store)
```

```
filter_exclude_names = [
    'bank', 'insurance', 'pension', 'Equities', 'Balance', 'ratios', 'Supermarket',
    'Budget', 'NSSF', 'Bamburi', 'Metropolitan', 'Land', 'REIT', 'Debt', 'Demand',
    'Retail', 'Sahara', 'USD', 'Residential', 'Apartment', 'Credit', 'Hospitality',
    'Thematic', 'Company', 'Commercial', 'Office', 'Business', 'Macro', 'Affordable',
    'Housing', 'Government', 'Industry', 'NPL', 'GDP', 'Urban', 'Nairobi', 'County',
    'Inflation', 'Commodity', 'Price', 'Academia', 'School', 'Fixed Income', 'EPS Growth',
    'Rental', 'Life Assurance', 'Loan', 'Education', 'Energy', 'Tourism', 'Restaurant',
    'Building', 'Estates', 'Kuramo', 'communications@cytonn.com', 'Fitch Rating',
    'Minority', 'Europe', 'Geothermal', 'Tanzania', 'Uganda', 'Occupancy', 'Savings Account'
    'Filimbi', 'Undisclosed', 'Sharehold', 'Foreign Direct Investment', 'IEBC',
    'Key Performance Indicator', 'hotels', 'Cytonn High Yield Fund', 'years old', 'West Afri
]
def table_has_extractable_schemes(tables_str_value: str):
    table_tag = BeautifulSoup(tables_str_value, "html.parser")
    tbody = str(table_tag.find('tbody')).lower()
    thead = str(table_tag.find('thead')).lower()
    exclude_names = [i.lower() for i in filter_exclude_names]
```

```
    if not any(i in thead or i in tbody for i in exclude_names):
        for _, aliases in SCHEME_NAME_ALIAS_MAP:
            for alias in aliases:
                if alias.lower() in tbody:
                    return True
    return False

unmatched_tables = [
    (index, table)
    for (index, key), tables
        in tqdm(grouped_by_key_id.items())
        if key == 'log_unmatched_table'
    for (_,_,table)
        in tables
        if table_has_extractable_schemes(table)
]
len(unmatched_tables)
```

```
100%|      | 1030/1030 [00:19<00:00, 53.44it/s]
```

```
0
```

We are able to extract all tables with schemes that are in KES money market funds, and assets under management.

Preview the data

Expand date range

```
def expand_date_column(df: pd.DataFrame, expand_column: str):
    for _, row in df.iterrows():
        expanding_values = row[expand_column]
        if type(expanding_values) in [list, tuple] :
            start_date = datetime.strptime(expanding_values[0], "%Y-%m-%d")
            end_date = datetime.strptime(expanding_values[1], "%Y-%m-%d")
            start_end_diff_days = (end_date - start_date).days
            day_list = [
                (start_date + timedelta(days=i)).strftime('%Y-%m-%d')
                for i
```

```
                in range(start_end_diff_days + 1)
            ]
            for day in day_list:
                yield { **row.to_dict(), expand_column: day }
        else:
            yield row.to_dict()

expanded_records_df = pd.DataFrame(expand_date_column(extracted_records_df, 'entry_date'))
expanded_records_df['entry_date'] = pd.to_datetime(expanded_records_df['entry_date'])
expanded_records_df
```

	entry_type	entry_date	entry_value	entry_scheme
0	EFFECTIVE_ANNUAL_RATE	2024-12-22	17.1	Cytonn Unit Trust Scheme
1	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.6	Lofty Corban Unit Trust Scheme
2	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.3	Gulfcap Money Market Fund
3	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.2	Orient Umbrella Collective Investmer
4	EFFECTIVE_ANNUAL_RATE	2024-12-22	16.1	Etica Capital Limited
...	...	...	...	...
132059	EFFECTIVE_ANNUAL_RATE	2018-12-27	9.9	Zimele Unit Trust Scheme
132060	EFFECTIVE_ANNUAL_RATE	2018-12-28	9.9	Zimele Unit Trust Scheme
132061	EFFECTIVE_ANNUAL_RATE	2018-12-29	9.9	Zimele Unit Trust Scheme
132062	EFFECTIVE_ANNUAL_RATE	2018-12-30	9.9	Zimele Unit Trust Scheme
132063	EFFECTIVE_ANNUAL_RATE	2018-12-31	9.9	Zimele Unit Trust Scheme

Check duplicates

```
expanded_records_df[expanded_records_df.duplicated()]
```

	entry_type	entry_date	entry_value	entry_scheme
1656	EFFECTIVE_ANNUAL_RATE	2024-06-16	16.4	Nabo Africa Funds
2456	EFFECTIVE_ANNUAL_RATE	2024-03-10	17.7	Etica Capital Limited
2509	EFFECTIVE_ANNUAL_RATE	2024-02-25	16.1	GenAfrica Unit Trust Scheme
2532	EFFECTIVE_ANNUAL_RATE	2024-02-18	17.0	Lofty Corban Unit Trust Scheme
2631	EFFECTIVE_ANNUAL_RATE	2024-01-21	17.3	Etica Capital Limited
...	...	...	...	...
130234	ASSETS_UNDER_MANAGEMENT	2019-07-28	20270.8	CIC Unit Trust Scheme
130235	ASSETS_UNDER_MANAGEMENT	2019-07-28	8841.6	British-American Unit Trust Sch

	entry_type	entry_date	entry_value	entry_scheme
130236	ASSETS_UNDER_MANAGEMENT	2019-07-28	6578.8	Old Mutual Unit Trust Scheme
130237	ASSETS_UNDER_MANAGEMENT	2019-07-28	6951.9	ICEA Unit Trust Scheme
130238	ASSETS_UNDER_MANAGEMENT	2019-07-28	5189.7	NCBA Unit Trust Funds

```
# Remove the duplicates
expanded_records_df = expanded_records_df.drop_duplicates()
```

Sort

```
expanded_records_df = expanded_records_df.sort_values(by='entry_date')
expanded_records_df
```

	entry_type	entry_date	entry_value	entry_scheme
83624	EFFECTIVE_ANNUAL_RATE	2017-11-01	11.1	Cytonn Unit Trust Scheme
125347	ASSETS_UNDER_MANAGEMENT	2018-01-01	256.6	Apollo Unit Trust Scheme
130969	EFFECTIVE_ANNUAL_RATE	2018-01-01	10.1	CIC Unit Trust Scheme
97161	ASSETS_UNDER_MANAGEMENT	2018-01-01	19756.7	CIC Unit Trust Scheme
124442	ASSETS_UNDER_MANAGEMENT	2018-01-01	5837.0	NCBA Unit Trust Funds
...	...	...	...	...
28	EFFECTIVE_ANNUAL_RATE	2024-12-22	12.2	Ziidi Money Market Fund
29	EFFECTIVE_ANNUAL_RATE	2024-12-22	11.8	Stanbic Unit Trust Funds
30	EFFECTIVE_ANNUAL_RATE	2024-12-22	8.5	Equity Investment Bank Collecti
15	EFFECTIVE_ANNUAL_RATE	2024-12-22	13.9	Dry Associates Unit Trust Schen
0	EFFECTIVE_ANNUAL_RATE	2024-12-22	17.1	Cytonn Unit Trust Scheme

Ploting

```
grouped_df = expanded_records_df.groupby(
    ['entry_type', 'entry_date', 'entry_scheme'])['entry_value'].mean().reset_index()
grouped_df
```

Effective Annual Rate, Money Market Funds (KES)

```
EAR_df = grouped_df[
    grouped_df['entry_type'] == Extracted_Scheme_Entry.TYPE_EFFECTIVE_ANNUAL_RATE
    ].drop(columns=['entry_type']).copy()
EAR_pivot = EAR_df.pivot(index='entry_date', columns='entry_scheme', values='entry_value')
EAR_pivot
```

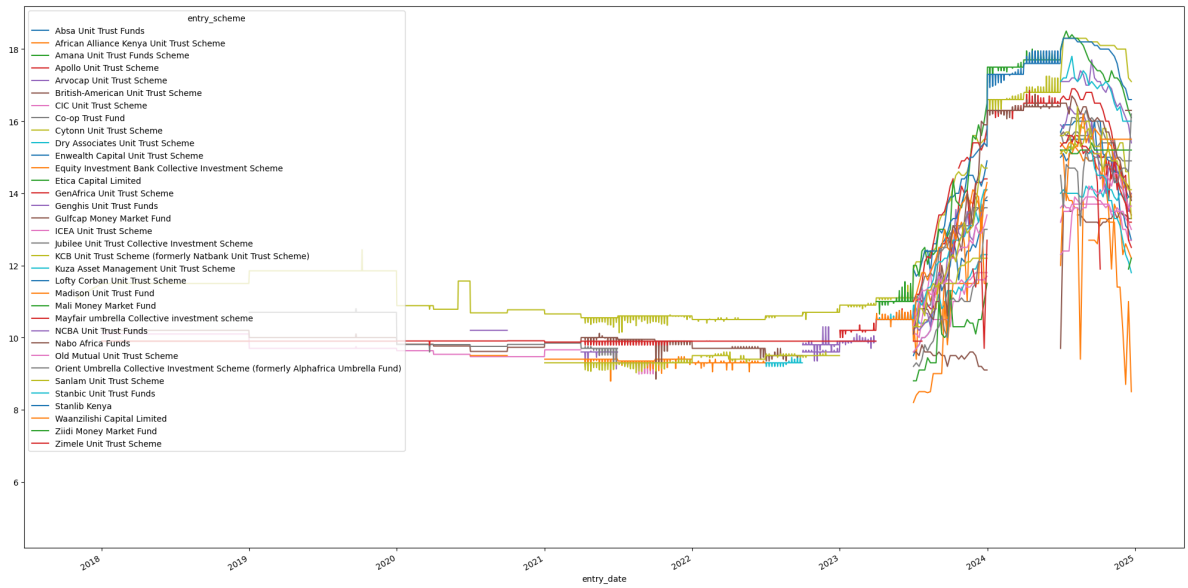
```
EAR_fig = px.line(EAR_pivot, x=EAR_pivot.index, y=EAR_pivot.columns)
EAR_fig.update_layout(
    height=800,
    margin=dict(t=100),
    title=dict(
        text="Effective Annual Rate (Percentage)",  # Your title here
        y=0.98,                    # Adjust the title's vertical position
        x=0.5,                     # Center the title
        xanchor='center',
        yanchor='top'
    ),
    xaxis=dict(
        side="top",    # This moves the x-axis to the top
        title="Date"   # This sets the title for the x-axis
    ),
    yaxis=dict(
        title="Effective Annual Rate"   # This sets the title for the x-axis
    ),

    legend=dict(
        orientation="h",  # horizontal orientation
        yanchor="bottom",
        y=-4.5,  # move the legend below the plot
        xanchor="center",
        x=0.5
    ))
EAR_fig.update_traces(
    hovertemplate="<br>".join([
        "scheme=%{fullData.name}",
        "date=%{x|%Y-%m-%d}",
        "annual_rate=%{y}%",
        # removes any additional trace information that Plotly might add by default.
        "<extra></extra>"
    ])
)
EAR_fig.show()
```

```
Unable to display output for mime type(s): text/html
```

Interactive plot of Effective Annual Rate (Percentage)

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```



Assets Under Management (Millions - KES)

```
AUM_df = grouped_df[
    grouped_df['entry_type'] == Extracted_Scheme_Entry.TYPE_ASSETS_UNDER_MANAGEMENT
    ].drop(columns=['entry_type']).copy()
AUM_pivot = AUM_df.pivot(index='entry_date', columns='entry_scheme', values='entry_value')
AUM_pivot
```

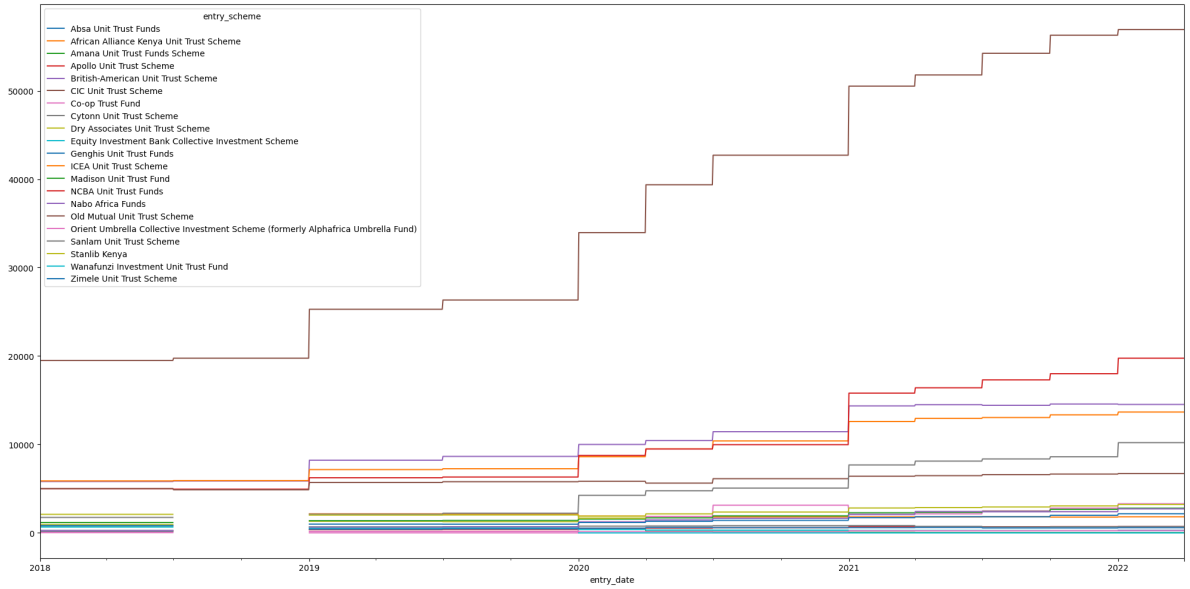
```
AUM_fig = px.line(AUM_pivot, x=AUM_pivot.index, y=AUM_pivot.columns)
AUM_fig.update_layout(
    height=800,
    margin=dict(t=100),
    title=dict(
        text="Assets Under Management (KSH Millions)",  # Your title here
        y=0.98,                  # Adjust the title's vertical position
        x=0.5,                   # Center the title
        xanchor='center',
```

```
        yanchor='top'
    ),
    xaxis=dict(
        side="top",    # This moves the x-axis to the top
        title="Date"   # This sets the title for the x-axis
    ),
    yaxis=dict(
        title="Assets Under Management"   # This sets the title for the x-axis
    ),

    legend=dict(
        orientation="h",  # horizontal orientation
        yanchor="bottom",
        y=-4.5,  # move the legend below the plot
        xanchor="center",
        x=0.5
    ))
AUM_fig.update_traces(
    hovertemplate="<br>".join([
        "scheme=%{fullData.name}",
        "date=%{x|%Y-%m-%d}",
        "AUM=%{y} (Mill-Kes)",
        # removes any additional trace information that Plotly might add by default.
        "<extra></extra>"
    ])
)
AUM_fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

Interactive plot of Assets Under Management (KSH Millions)



Archives - Data Preservation and Reproducibility

Importance of Data Preservation

In the rapidly evolving landscape of financial markets, preserving historical data is crucial for long-term analysis, policy-making, and understanding market trends. Our archiving efforts aim to create a lasting resource for researchers, analysts, and policymakers interested in Kenya's collective investment schemes.

Archiving Methodology

To ensure the perpetuity and reproducibility of this analysis, we have implemented a comprehensive archiving strategy:

1. **Raw Data Preservation**: We've archived the original, unaltered data crawled from various sources. This includes:
    - Raw Cytonn Reports: Over 600 market research reports from Cytonn, dating back to 2014.
    - Capital Markets Authority Approved Schemes: Official listings of CMA-approved collective investment schemes.
2. **Processed Data Archival**: We've preserved the cleaned and structured dataset resulting from our analysis:

- Processed Investment Schemes Records: This includes standardized data on fund performance, assets under management, and other key metrics.

3. **Comprehensive Documentation**: We've created detailed documentation covering:
   - Data collection methodologies
   - Data cleaning and processing techniques
   - Variable definitions and data structure explanations
   - Any assumptions or limitations in the data or analysis
4. **Open Access**: All archived data is made available through the Hugging Face platform, ensuring:
   - Easy accessibility for researchers globally
   - Version control and long-term preservation
   - Compliance with fair use principles and ethical data sharing practices

Accessing the Archives

The complete archive is hosted on Hugging Face at: https://huggingface.co/datasets/ToKnow-ai/Kenyan-Collective-Investment-Schemes-Dataset

Researchers can access this data through the Hugging Face interface or via API calls, facilitating easy integration into various research workflows.

Impact and Future Research

By preserving both raw and processed data, along with comprehensive documentation, we aim to:

1. Enable verification and replication of our analysis
2. Facilitate longitudinal studies on Kenya's collective investment schemes
3. Provide a foundation for comparative studies with other markets
4. Support evidence-based policy-making in Kenya's financial sector

Ethical Considerations and Usage Guidelines

While we encourage the use of this data for research and analysis, users should:

1. Adhere to the terms of use specified in the dataset documentation
2. Properly cite the dataset in any resulting publications or analyses
3. Be aware of the limitations and potential biases in the data, as outlined in our documentation

Conclusion

This archiving effort not only supports the reproducibility of our current analysis but also sets a precedent for transparent, ethical data preservation in financial research. We hope this resource will contribute to a deeper understanding of Kenya's financial markets and foster innovation in financial research methodologies.

Archived Links

- https://licensees.cma.or.ke/ - archive
- https://www.rba.go.ke/registered-fund-managers/ - archive
- https://centwarrior.com/aa-kenya-shillings-fund/ - archive
- https://www.linkedin.com/posts/centwarrior_aa-kenya-shillings-fund-explained-in-2024-activity-7169322082814705664-8nwu - archive
- https://cytonn.com/topicals/investment-risk-analysis - archive
- https://equitygroupholdings.com/ke/investor-relations/eib
- https://enwealth.co.ke/about/#governance - archive
- https://www.linkedin.com/company/enwealth-kenya/ - archive
- https://enwealth.co.ke/capital/enwealth-money-market-fund/ - archive
- https://www.linkedin.com/company/arvocap-asset-managers/ - archive
- https://www.businessdailyafrica.com/bd/markets/avocarp-latest-to-enter-kenya-s-asset-management-market-4644586 - archive
- https://www.businessdailyafrica.com/bd/markets/capital-markets/fintech-start-up-ndovu-gets-cma-nod-to-set-up-money-market-fund-4548970 - archive