Step-by-Step Fee Schedule Process

Purpose:

Updating the HRP with the monthly Medi-Cal fee schedule requires a structured and systematic approach. This document provides a detailed overview of how to convert the medical fee rates file into the HRP format. It is aimed at informing providers about the reimbursement rates for different medical services under Medi-Cal.

For this monthly Medi-Cal fee rate update, our team has developed an automated Python script. This script fetches and downloads the required data file, then performs analyses based on different criteria to generate a set of output text files. These files are provided to the HRP for the necessary updates.

Source file Analysis:

The source for the Fee Schedule files is part of the Medi-Cal Rates section on the California Department of Health Care Services (DHCS) website. These rates are updated every 15 days and the data related to these updates can be downloaded from this page. The "Rates" tab on this site contains detailed information about the current medical fee schedules, including various billing codes, rate descriptions, effective dates, and possibly the applicable provider types or regions.

For an in-depth analysis of the columns and data in the file, let's examine the contents of the Excel file. Here's a breakdown of the columns in the file:

- **Proc Type**: Indicates the type of procedure, possibly identifying different categories or groups of medical services.
- **Proc Code**: The procedure code, likely corresponding to specific medical services or treatments.
- **Procedure Description**: Describes the medical service or treatment corresponding to the procedure code.
- **Unit Value**: The assigned value for the procedure, which might be used to calculate the final rate or payment.
- **Basic Rate**: The standard reimbursement rate for the procedure.
- **Child Rate**: Specific rate applicable to pediatric cases, if different from the basic rate.
- **ER Rate**: Emergency room rate, if applicable.
- **Conv Ind**: A code indicating specific conditions or additional indicators relevant to the procedure.
- **ER Ind**: Indicates whether the procedure is associated with emergency services.
- **Cutback Ind**: Could indicate adjustments or reductions applicable under certain conditions.
- **Prof** %: Percentage applicable to professional fees, if different from the basic rate.
- **Rental Rate**: Rate applicable if the procedure involves rental of equipment or facilities.
- **Non-Physn Med Prac Ind**: Indicates whether the procedure is applicable to non-physician medical practitioners.

These columns provide detailed information about the payment and billing guidelines for various medical procedures under the Medi-Cal program. The data would be critical for healthcare providers in understanding how different services are reimbursed and any special conditions that affect these rates.

Process Flow:

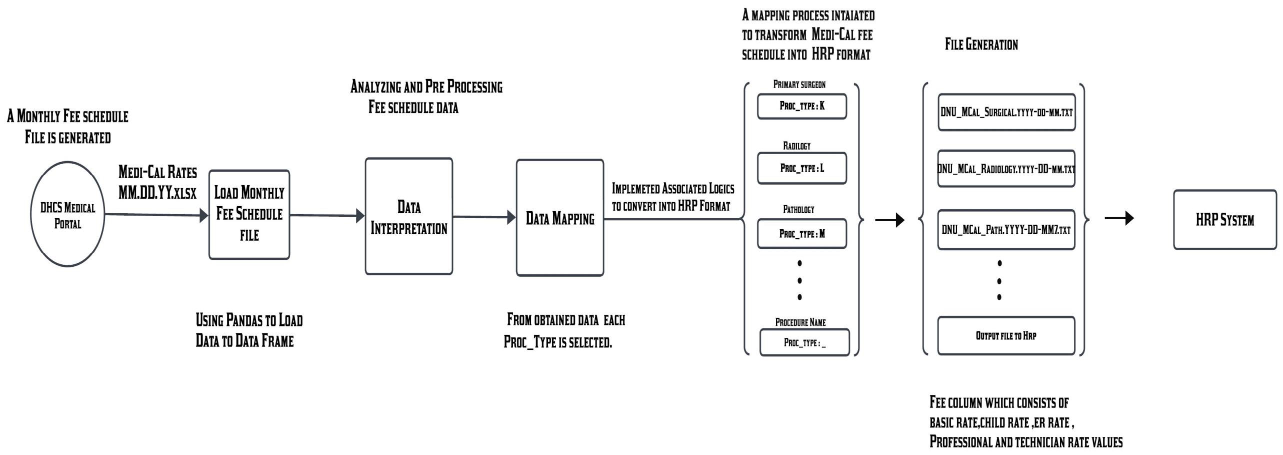
1. **Input Data Structure** (Medi-Cal Fee Schedule): The input consists of a comprehensive Excel spreadsheet with procedure codes and corresponding rates. These codes are categorized into "Proc Types" for classification. The fee schedule displays different rates for criteria such as Basic Rate, Child Rate, ER Rate, age, place of service, and procedure modifiers.

2. **Mapping to HRP Format**: The HRP system utilizes "Proc Types" for initial categorization. A mapping exercise has been undertaken to convert the Medi-Cal data into the format required by HRP, considering the specific criteria for each procedure type. This includes splitting multiple fees within a single Medi-Cal record into multiple rows for HRP.

3. **Output Data Structure** (HRP Config): The output files generated with mapping will be in the required format for HRP to load. The Fee Tables will align with the fee schedule and the Details will contain the code. The HRP system requires the details to accommodate the variance in fees as indicated in the Medi-Cal schedule.

4. **Process Automation**: The goal is to develop an automated and repeatable process. This involves loading the fee schedule into Python Code, interrogating the data based on the established criteria, generating data files compatible with HRP, and facilitating audits.

In practice, this process ensures that the Medi-Cal fee schedule data is accurately and efficiently ingested into the HRP system, with a keen focus on maintaining the integrity of data and supporting the subsequent updates for pricing claims. Detailed data manipulation and mapping are essential steps to translate the Medi-Cal schedule into the operational format for HRP, thereby enabling the application of appropriate fees for services rendered.


A Monthly Fee schedule File is generated
DHCS Medical Portal
Medi-Cal Rates MM.DD.YY.xlsx
Load Monthly Fee Schedule file
Using Pandas to Load Data to Data Frame
Analyzing and Pre Processing Fee schedule data
Data Interpretation
Data Mapping
From obtained data each Proc_Type is selected.
Implemeted Associated Logics to convert into HRP Format
A mapping process intaiated to transform Medi-Cal fee schedule into HRP format
Primary surgeon
Proc_type : K
Radilogy
Proc_type : L
Pathology
Proc_type : M
Procedure Name
Proc_type : _
File Generation
DNU_MCal_Surgical.yyyy-dd-mm.txt
DNU_MCal_Radiology.yyyy-DD-mm.txt
DNU_MCal_Path.YYYY-DD-MM7.txt
Output file to Hrp
Fee column which consists of basic rate,child rate ,er rate , Professional and technician rate values
HRP System

1. Overview of Files

Python Files:

fee_schedule_final.py: This is the main script of the project that processes healthcare-related data to adjust medical fee rates based on various conditions.

Configuration Files:
config.json: Contains settings such as logging configuration, data columns, and other parameters.
logging_config.ini: Configures logging settings for the Python script, such as log level and log file path.

2. Installation and Setup:

System Requirements:
Ensure Python is installed on your system. You can download it from official Python website.

Library Installation:
Use pip to install libraries not included in the Python Standard Library:
Ex: pip install **pandas**.

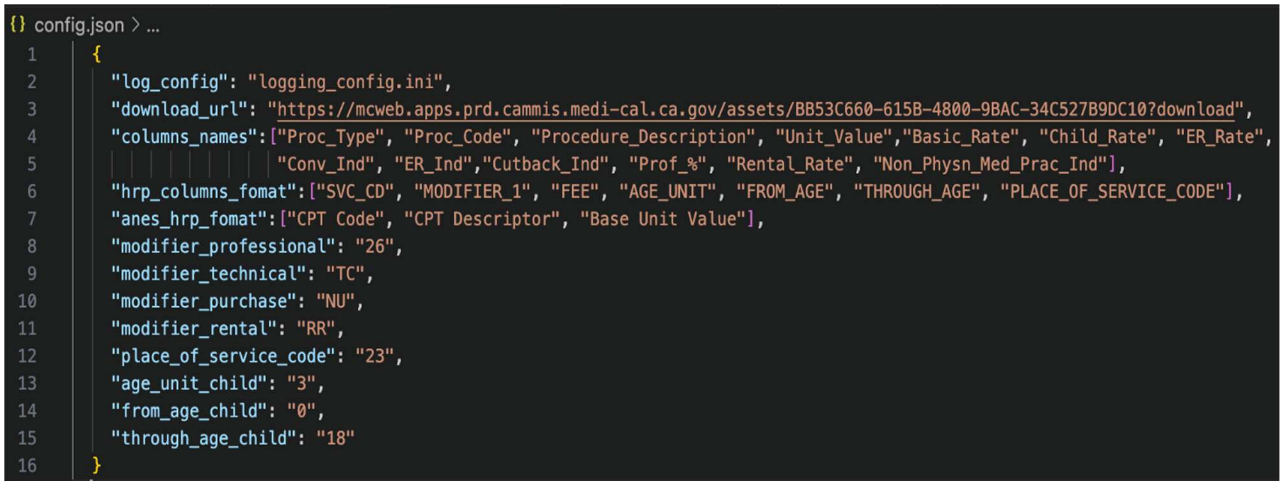
3. Required Libraries and Modules:

Module	Purpose	Installation Command
zipfile	Read and write ZIP archives	Included with Python
datetime	Manipulate dates and times	Included with Python
os	Interface with the operating system	Included with Python
requests	Send HTTP requests	**pip install requests**
pandas	Data manipulation and analysis	**pip install pandas**
logging	Logging capabilities	Included with Python
json	Parsing and outputting JSON	Included with Python
shutil	File operations like copying and archiving	Included with Python

4. Configuration Setup:

- config.json and logging_config.ini should be placed in the same directory as your Python script or their paths specified in the script.
- **config.json** file Contains settings for log configuration, data download URL, column names, data processing formats, and specific modifiers.
- **logging_config.ini** file Configures the logging system, including file paths, log levels, and formats.
- Adjust these settings to match the requirements of the data processing tasks.

5. Understanding of configuration files:

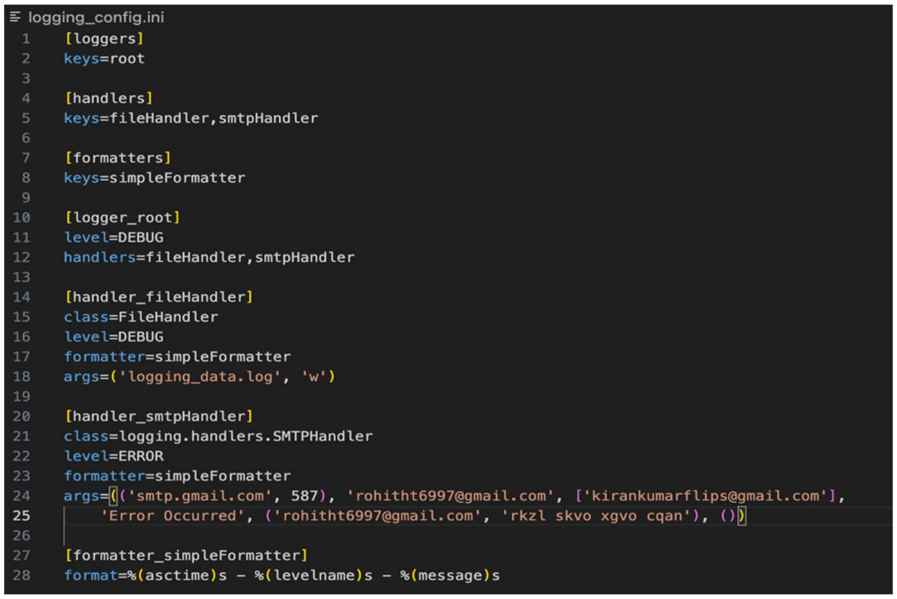


{} config.json > ...

```json
{
  "log_config": "logging_config.ini",
  "download_url": "https://mcweb.apps.prd.cammis.medi-cal.ca.gov/assets/BB53C660-615B-4800-9BAC-34C527B9DC10?download",
  "columns_names":["Proc_Type", "Proc_Code", "Procedure_Description", "Unit_Value","Basic_Rate", "Child_Rate", "ER_Rate",
                   "Conv_Ind", "ER_Ind","Cutback_Ind", "Prof_%", "Rental_Rate", "Non_Physn_Med_Prac_Ind"],
  "hrp_columns_fomat":["SVC_CD", "MODIFIER_1", "FEE", "AGE_UNIT", "FROM_AGE", "THROUGH_AGE", "PLACE_OF_SERVICE_CODE"],
  "anes_hrp_fomat":["CPT Code", "CPT Descriptor", "Base Unit Value"],
  "modifier_professional": "26",
  "modifier_technical": "TC",
  "modifier_purchase": "NU",
  "modifier_rental": "RR",
  "place_of_service_code": "23",
  "age_unit_child": "3",
  "from_age_child": "0",
  "through_age_child": "18"
}
```

a. The config.json file contains configuration data for the Code. Here's a detailed breakdown of its contents:
    - **log_config**: Specifies the logging configuration file (logging_config.ini).
    - **download_url**: URL for downloading Fee rates files from DHCS Website.
    - **columns_names**: Names of the columns for input Medical Rates file.
    - **hrp_columns_fomat**: Column names in a specific format related to HRP.
    - **anes_hrp_fomat**: Column names for anesthesia related to HRP.
    - **modifier_professional**: Code for professional services (26).
    - **modifier_technical**: Code for technical services (TC).
    - **modifier_purchase**: Code for purchased items (NU).
    - **modifier_rental**: Code for rental items (RR).
    - **place_of_service_code**: Code for place of service (23).
    - **age_unit_child**: Age unit for children (3).
    - **from_age_child**: Starting age for children (0).
    - **through_age_child**: Ending age for children (18).

b. The logging_config.ini file configures the logging system for the application, specifying how log messages are handled and formatted. This configuration is crucial for monitoring and debugging the application.



logging_config.ini

```
[loggers]
keys=root

[handlers]
keys=fileHandler,smtpHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=DEBUG
handlers=fileHandler,smtpHandler

[handler_fileHandler]
class=FileHandler
level=DEBUG
formatter=simpleFormatter
args=('logging_data.log', 'w')

[handler_smtpHandler]
class=logging.handlers.SMTPHandler
level=ERROR
formatter=simpleFormatter
args=(('smtp.gmail.com', 587), 'rohitht6997@gmail.com', ['kirankumarflips@gmail.com'],
    'Error Occurred', ('rohitht6997@gmail.com', 'rkzl skvo xgvo cqan'), ())

[formatter_simpleFormatter]
format=%(asctime)s - %(levelname)s - %(message)s
```

Loggers

- keys=root: Defines the root logger, which captures all log messages generated by the application.

Handlers

- keys=fileHandler,smtpHandler: Specifies two handlers:
- fileHandler: Logs messages to a file.
- smtpHandler: Sends log messages via email.

Formatters

- keys=simpleFormatter: Defines a single formatter for formatting log messages.

Logger Configuration

- root logger:
- level=DEBUG: Captures all messages at DEBUG level and higher.
- handlers=fileHandler,smtpHandler: Uses both handlers for logging.

Handler Details

- fileHandler:
- class=FileHandler
- level=DEBUG: Logs all messages at DEBUG level and higher.
- formatter=simpleFormatter
- args=('logging_data.log', 'w'): Writes to logging_data.log in write mode.

smtpHandler:

- class=logging.handlers.SMTPHandler
- level=ERROR: Sends emails for ERROR level messages and higher.
- formatter=simpleFormatter
- args=(('smtp.gmail.com', 587), '####@gmail.com', ['%%%%@gmail.com'], 'Error Occurred', ('####@gmail.com', 'rkzl skvo xgvo cqan'), ()): Configured to send emails via Gmail.

Formatter

- simpleFormatter:
- format=%(asctime)s - %(levelname)s - %(message)s: Formats log messages with timestamp, log level, and messag

6. Detailed Documentation for fee_schedule_final.py

Purpose:
Designed to process healthcare-related data, adjusting medical fee rates for different conditions and settings.

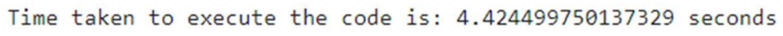
Core Logic:
The script processes different fee calculations based on provided data, applying specific conditions and modifiers for professional, technical, and rental rates.

Main Functionalities:
Downloading Data: Downloads datasets, handles errors, and extracts content.
Data Processing Functions:
Basic_Rate_Calculation, Child_Rate_Calculation, ER_Rate_Calculation: Calculate fees based on various conditions.
Purchase_Calculation, Rental_Calculation: Adjust fees for purchased or rented equipment/services.
anes_calculation: Organizes anesthesia service data.

File Output:

- The script creates several output files based on the type of service (e.g., Surgical, Radiology, Path, and Lab, etc.
- Each file is named according to the type and includes relevant data processed by the respective functions.
- Logs information about the process flow, ensuring that all steps are tracked for debugging and verification purposes.

Time Taken:



```
Time taken to execute the code is: 4.424499750137329 seconds
```

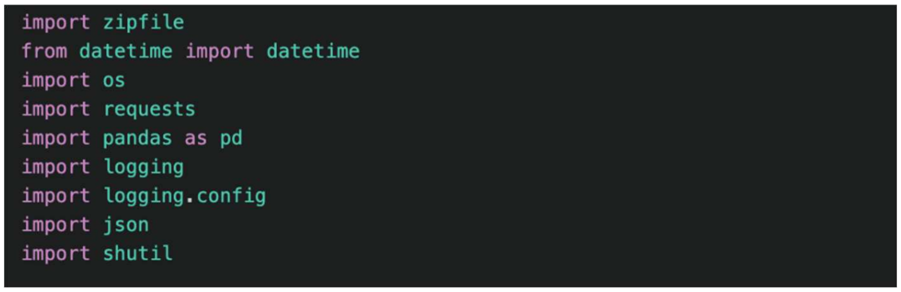
The total time taken to execute the code to produce output for all 15 fee schedule files (along with Augment code) is 4.42 sec.

Usage:
To execute the script, ensure all configurations are set in config.json, and execute the Python file. The script will process the data as per the functions defined and output the respective files in the specified directory.

Error Handling:
The script includes error handling for file download issues, data processing errors, and logging misconfigurations, which helps in maintaining robust execution flows.

7. Step by step Code Flow and Functionality:

a) Import Required Modules:

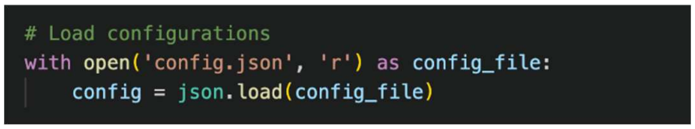


```python
import zipfile
from datetime import datetime
import os
import requests
import pandas as pd
import logging
import logging.config
import json
import shutil
```

zipfile	For working with ZIP files.
datetime	For handling date and time.
os	For interacting with the operating system.
requests	For making HTTP requests.
pandas	For data manipulation and analysis.
logging	For logging events and errors.
json	For parsing JSON files.
shutil	For high-level file operations.

These modules collectively enable the script to automate downloading, extracting, processing, and logging tasks efficiently.

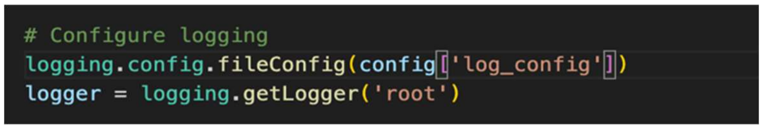
b) Load Configurations:



```python
# Load configurations
with open('config.json', 'r') as config_file:
    config = json.load(config_file)
```

Action: Opens and reads the config.json file, loading its contents into config dictionary.
Purpose: To retrieve configuration settings needed for script, such as the download URL, Module parameters and column names.

c) Configure Logging

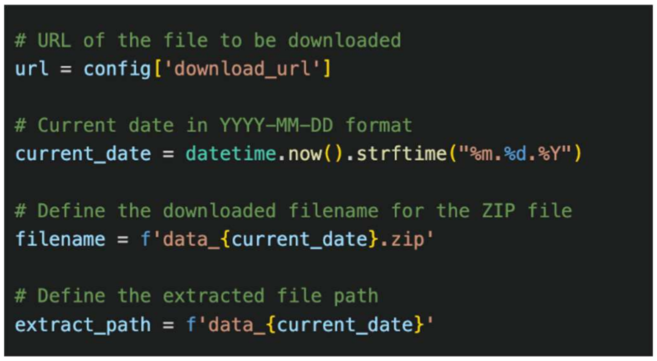


```python
# Configure logging
logging.config.fileConfig(config['log_config'])
logger = logging.getLogger('root')
```

Action: Configures logging using the settings from the logging configuration file specified in config['log_config'] and creates a logger named root.

Purpose: To set up logging for recording events and errors during the script's execution.

d) Define Variables:

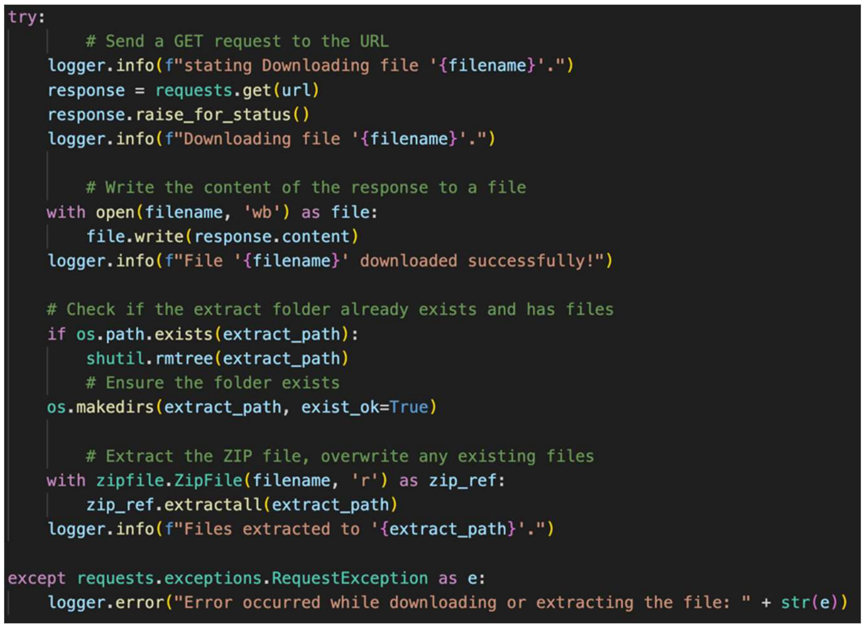


```
# URL of the file to be downloaded
url = config['download_url']

# Current date in YYYY-MM-DD format
current_date = datetime.now().strftime("%m.%d.%Y")

# Define the downloaded filename for the ZIP file
filename = f'data_{current_date}.zip'

# Define the extracted file path
extract_path = f'data_{current_date}'
```

Action: Defines variables for the URL, current date, download filename, and extraction path.

Purpose: To prepare for downloading and extracting the file with dynamically generated names based on the current date.

e) Download and Extract File:



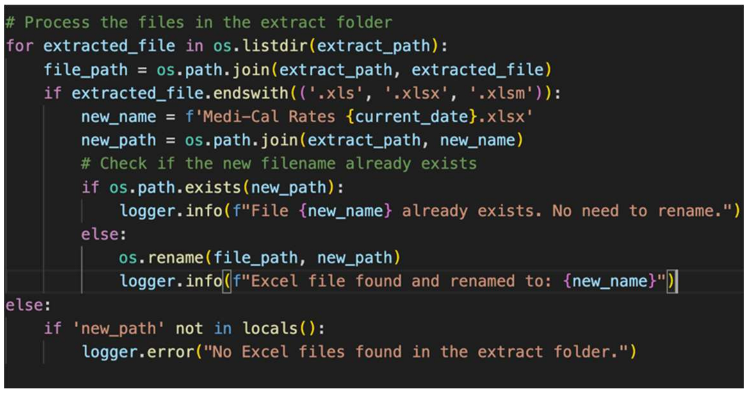
```
try:
        # Send a GET request to the URL
    logger.info(f"stating Downloading file '{filename}'.")
    response = requests.get(url)
    response.raise_for_status()
    logger.info(f"Downloading file '{filename}'.")

        # Write the content of the response to a file
    with open(filename, 'wb') as file:
        file.write(response.content)
    logger.info(f"File '{filename}' downloaded successfully!")

    # Check if the extract folder already exists and has files
    if os.path.exists(extract_path):
        shutil.rmtree(extract_path)
        # Ensure the folder exists
    os.makedirs(extract_path, exist_ok=True)

        # Extract the ZIP file, overwrite any existing files
    with zipfile.ZipFile(filename, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
    logger.info(f"Files extracted to '{extract_path}'.")

except requests.exceptions.RequestException as e:
    logger.error("Error occurred while downloading or extracting the file: " + str(e))
```

Action:

- Logs the start of the download process.
- Sends an HTTP GET request to the URL.
- Writes the downloaded content to a file.
- Logs the successful download.
- Removes existing extraction directory (for. zip extract) if it exists.
- Creates a new extraction directory.
- Extracts the downloaded ZIP file to the extraction directory.
- Logs the successful extraction.

Purpose: To download a ZIP file from a specified URL and extract its contents to a designated directory.

f) Process Extracted files:



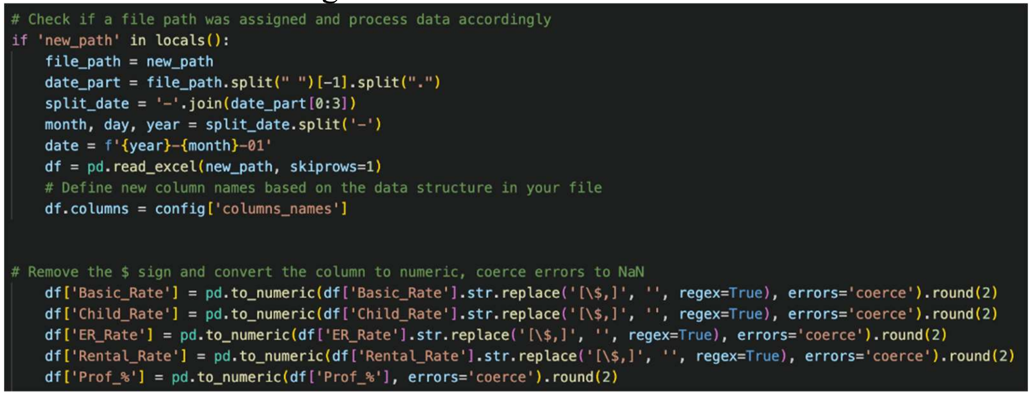
```python
# Process the files in the extract folder
for extracted_file in os.listdir(extract_path):
    file_path = os.path.join(extract_path, extracted_file)
    if extracted_file.endswith(('.xls', '.xlsx', '.xlsm')):
        new_name = f'Medi-Cal Rates {current_date}.xlsx'
        new_path = os.path.join(extract_path, new_name)
        # Check if the new filename already exists
        if os.path.exists(new_path):
            logger.info(f"File {new_name} already exists. No need to rename.")
        else:
            os.rename(file_path, new_path)
            logger.info(f"Excel file found and renamed to: {new_name}")
else:
    if 'new_path' not in locals():
        logger.error("No Excel files found in the extract folder.")
```

Action:

- Iterates over files in the extraction directory.
- Identifies Excel file and renames it to a standard format **('Medi-Cal Rates {current_date}.xlsx')**
- Logs the renaming or existence of the file.
- Logs an error if no Excel file is found.

Purpose: To process extracted Excel files by renaming them and preparing them for further data manipulation.

g) Further Data Processing:



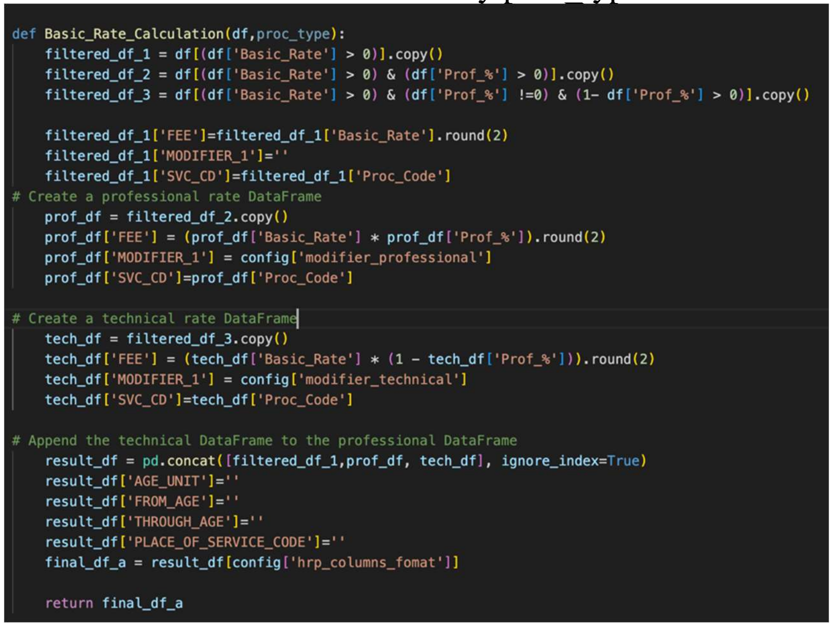
```python
# Check if a file path was assigned and process data accordingly
if 'new_path' in locals():
    file_path = new_path
    date_part = file_path.split(" ")[-1].split(".")
    split_date = '-'.join(date_part[0:3])
    month, day, year = split_date.split('-')
    date = f'{year}-{month}-01'
    df = pd.read_excel(new_path, skiprows=1)
    # Define new column names based on the data structure in your file
    df.columns = config['columns_names']


# Remove the $ sign and convert the column to numeric, coerce errors to NaN
    df['Basic_Rate'] = pd.to_numeric(df['Basic_Rate'].str.replace('[\$,]', '', regex=True), errors='coerce').round(2)
    df['Child_Rate'] = pd.to_numeric(df['Child_Rate'].str.replace('[\$,]', '', regex=True), errors='coerce').round(2)
    df['ER_Rate'] = pd.to_numeric(df['ER_Rate'].str.replace('[\$,]', '', regex=True), errors='coerce').round(2)
    df['Rental_Rate'] = pd.to_numeric(df['Rental_Rate'].str.replace('[\$,]', '', regex=True), errors='coerce').round(2)
    df['Prof_%'] = pd.to_numeric(df['Prof_%'], errors='coerce').round(2)
```

Action:

- Checks if a file path for the renamed Excel file exists.
- Parses the date from the filename.
- Reads the Excel file into a pandas DataFrame.
- Sets the DataFrame column names from the configuration file.
- Converts currency columns to numeric, handling errors and removing dollar signs.
- Rounds the numeric values to two decimal places for required columns.

Purpose: To ensure that the data in the Excel file is properly formatted and ready for analysis or further processing.

h) Basic rate calculation for necessary proc_types:



```python
def Basic_Rate_Calculation(df,proc_type):
    filtered_df_1 = df[(df['Basic_Rate'] > 0)].copy()
    filtered_df_2 = df[(df['Basic_Rate'] > 0) & (df['Prof_%'] > 0)].copy()
    filtered_df_3 = df[(df['Basic_Rate'] > 0) & (df['Prof_%'] !=0) & (1- df['Prof_%'] > 0)].copy()

    filtered_df_1['FEE']=filtered_df_1['Basic_Rate'].round(2)
    filtered_df_1['MODIFIER_1']=''
    filtered_df_1['SVC_CD']=filtered_df_1['Proc_Code']
# Create a professional rate DataFrame
    prof_df = filtered_df_2.copy()
    prof_df['FEE'] = (prof_df['Basic_Rate'] * prof_df['Prof_%']).round(2)
    prof_df['MODIFIER_1'] = config['modifier_professional']
    prof_df['SVC_CD']=prof_df['Proc_Code']

# Create a technical rate DataFrame
    tech_df = filtered_df_3.copy()
    tech_df['FEE'] = (tech_df['Basic_Rate'] * (1 - tech_df['Prof_%'])).round(2)
    tech_df['MODIFIER_1'] = config['modifier_technical']
    tech_df['SVC_CD']=tech_df['Proc_Code']

# Append the technical DataFrame to the professional DataFrame
    result_df = pd.concat([filtered_df_1,prof_df, tech_df], ignore_index=True)
    result_df['AGE_UNIT']=''
    result_df['FROM_AGE']=''
    result_df['THROUGH_AGE']=''
    result_df['PLACE_OF_SERVICE_CODE']=''
    final_df_a = result_df[config['hrp_columns_fomat']]

    return final_df_a
```

Action:

- Filters and creates a three DataFrame based on specific conditions from main data frame.
- Calculates FEE for the Basic, professional, and technical rates depend on conditions provided.
- Concatenates the results into a single DataFrame.
- Adds required columns like Modifier_1, AGE_UNIT, FROM_AGE, etc and assign its values based on conditions
- Returns the DataFrame with the specified columns in predefined format.

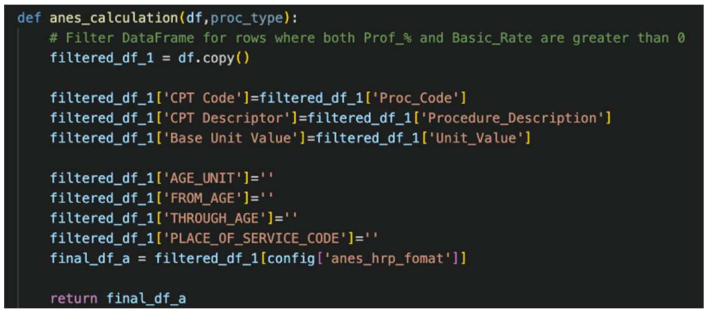
Purpose: Calculate the basic rate for procedures, including professional and technical rates.

Note:

- Similar to the Basic Rate Calculation function, we have two additional functions for calculating the Child Rate(**Child_Rate_Calculation**) and ER Rate(**ER_Rate_Calculation**) for the Fee component. However, the logic for filtering the DataFrame, and the values for Modifier and other columns will differ.

- Additionally, **Purchase_Calculation**() and **Rental_Calculation**() functions are specifically for proc_type='1' and are used to calculate the Fee component and other necessary modifiers and respective column fields

I) Anes Caluculation:



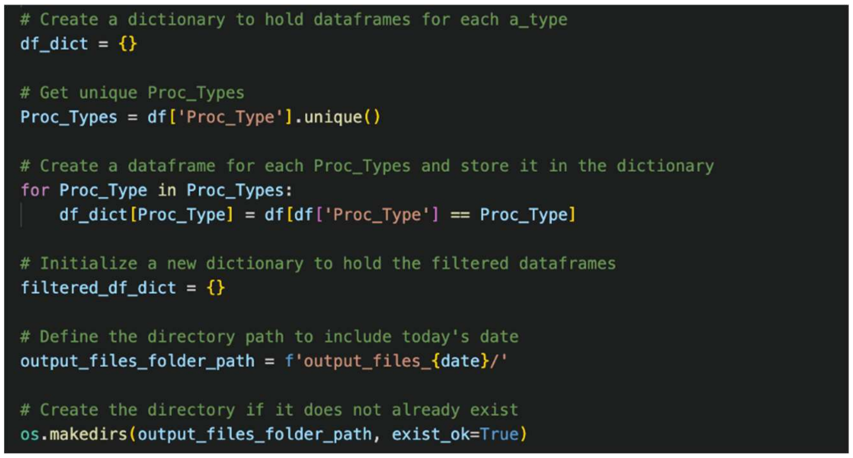
```python
def anes_calculation(df,proc_type):
    # Filter DataFrame for rows where both Prof_% and Basic_Rate are greater than 0
    filtered_df_1 = df.copy()

    filtered_df_1['CPT Code']=filtered_df_1['Proc_Code']
    filtered_df_1['CPT Descriptor']=filtered_df_1['Procedure_Description']
    filtered_df_1['Base Unit Value']=filtered_df_1['Unit_Value']

    filtered_df_1['AGE_UNIT']=''
    filtered_df_1['FROM_AGE']=''
    filtered_df_1['THROUGH_AGE']=''
    filtered_df_1['PLACE_OF_SERVICE_CODE']=''
    final_df_a = filtered_df_1[config['anes_hrp_fomat']]

    return final_df_a
```

Action:

- Copies the DataFrame and creates columns for anesthesia-specific data (CPT Code, CPT Descriptor, Base Unit Value).
- Adds required Modifiers (AGE_UNIT, FROM_AGE, etc.).
- Returns the DataFrame with the specified columns.

Purpose: Calculate anesthesia rates for the proc type “J” with respective logic.

J) DataFrame Initialization and Directory Setup:



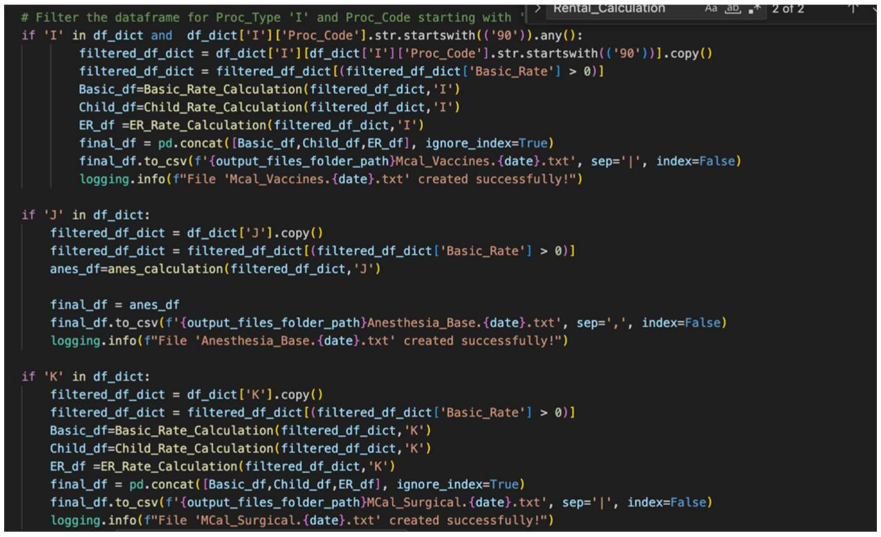
```python
# Create a dictionary to hold dataframes for each a_type
df_dict = {}

# Get unique Proc_Types
Proc_Types = df['Proc_Type'].unique()

# Create a dataframe for each Proc_Types and store it in the dictionary
for Proc_Type in Proc_Types:
    df_dict[Proc_Type] = df[df['Proc_Type'] == Proc_Type]

# Initialize a new dictionary to hold the filtered dataframes
filtered_df_dict = {}

# Define the directory path to include today's date
output_files_folder_path = f'output_files_{date}/'

# Create the directory if it does not already exist
os.makedirs(output_files_folder_path, exist_ok=True)
```

Action:

- Creates a dictionary to hold DataFrames for each unique Proc_Type.
- Initializes another dictionary to hold filtered DataFrames.
- Creates an output directory path based on the current date.
- Ensures the output directory exists.
-

Purpose: Sets up directories and initializes dictionaries for storing DataFrames.

k) Proc_type logic implementation:



```python
# Filter the dataframe for Proc_Type 'I' and Proc_Code starting with '
if 'I' in df_dict and  df_dict['I']['Proc_Code'].str.startswith(('90')).any():
        filtered_df_dict = df_dict['I'][df_dict['I']['Proc_Code'].str.startswith(('90'))].copy()
        filtered_df_dict = filtered_df_dict[(filtered_df_dict['Basic_Rate'] > 0)]
        Basic_df=Basic_Rate_Calculation(filtered_df_dict,'I')
        Child_df=Child_Rate_Calculation(filtered_df_dict,'I')
        ER_df =ER_Rate_Calculation(filtered_df_dict,'I')
        final_df = pd.concat([Basic_df,Child_df,ER_df], ignore_index=True)
        final_df.to_csv(f'{output_files_folder_path}Mcal_Vaccines.{date}.txt', sep='|', index=False)
        logging.info(f"File 'Mcal_Vaccines.{date}.txt' created successfully!")

if 'J' in df_dict:
    filtered_df_dict = df_dict['J'].copy()
    filtered_df_dict = filtered_df_dict[(filtered_df_dict['Basic_Rate'] > 0)]
    anes_df=anes_calculation(filtered_df_dict,'J')

    final_df = anes_df
    final_df.to_csv(f'{output_files_folder_path}Anesthesia_Base.{date}.txt', sep=',', index=False)
    logging.info(f"File 'Anesthesia_Base.{date}.txt' created successfully!")

if 'K' in df_dict:
    filtered_df_dict = df_dict['K'].copy()
    filtered_df_dict = filtered_df_dict[(filtered_df_dict['Basic_Rate'] > 0)]
    Basic_df=Basic_Rate_Calculation(filtered_df_dict,'K')
    Child_df=Child_Rate_Calculation(filtered_df_dict,'K')
    ER_df =ER_Rate_Calculation(filtered_df_dict,'K')
    final_df = pd.concat([Basic_df,Child_df,ER_df], ignore_index=True)
    final_df.to_csv(f'{output_files_folder_path}MCal_Surgical.{date}.txt', sep='|', index=False)
    logging.info(f"File 'MCal_Surgical.{date}.txt' created successfully!")
```

Initial Filtering:
For each Proc_Type ('I', 'J', 'K', 'L'….,etc), the code checks if it exists in the dictionary (df_dict).
Further filters are applied based on Proc_Code and Basic_Rate.

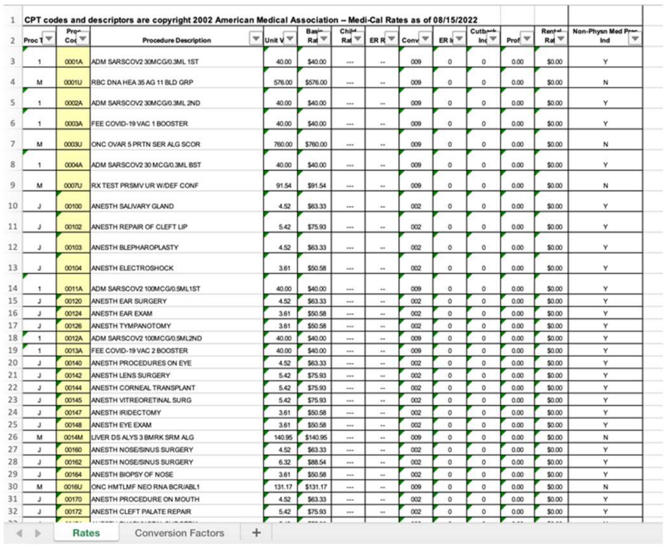
Calculations:
Functions (Basic_Rate_Calculation, Child_Rate_Calculation, ER_Rate_Calculation, anes_calculation) are called to calculate various rates.
Each function processes the DataFrame and returns the results.

Concatenation and Saving:
Results are concatenated into a final DataFrame.
The final DataFrame is saved as a text file with a specific naming convention based on Proc_Type.

Logging:
Logs are created to indicate the successful creation of each file.

Output file Overview:

As shown in the screenshot, the Medi-Cal fee schedule, available in an MS Excel format, contains thousands of rows detailing procedure codes and corresponding rates for pricing claims. The first Column of file consists of "Proc Types" for organizational and classification purposes. Additionally, the fee schedule includes different rates to define base, child, ER, and other applicable fees.



CPT codes and descriptors are copyright 2002 American Medical Association – Medi-Cal Rates as of 08/15/2022

Proc T	Proc Coc	Procedure Description	Unit V	Basic Rat	Child Rat	ER R	Conv	ER I	Cutback Inc	Prof	Rental Rat	Non-Physn Med Prac Ind
1	0001A	ADM SARSCOV2 30MCG/0.3ML 1ST	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
M	0001U	RBC DNA HEA 35 AG 11 BLD GRP	576.00	$576.00	---	--	009	0	0	0.00	$0.00	N
1	0002A	ADM SARSCOV2 30MCG/0.3ML 2ND	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
1	0003A	FEE COVID-19 VAC 1 BOOSTER	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
M	0003U	ONC OVAR 5 PRTN SER ALG SCOR	760.00	$760.00	---	--	009	0	0	0.00	$0.00	N
1	0004A	ADM SARSCOV2 30 MCG/0.3ML BST	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
M	0007U	RX TEST PRSMV UR W/DEF CONF	91.54	$91.54	---	--	009	0	0	0.00	$0.00	N
J	00100	ANESTH SALIVARY GLAND	4.52	$63.33	---	--	002	0	0	0.00	$0.00	Y
J	00102	ANESTH REPAIR OF CLEFT LIP	5.42	$75.93	---	--	002	0	0	0.00	$0.00	Y
J	00103	ANESTH BLEPHAROPLASTY	4.52	$63.33	---	--	002	0	0	0.00	$0.00	Y
J	00104	ANESTH ELECTROSHOCK	3.61	$50.58	---	--	002	0	0	0.00	$0.00	Y
1	0011A	ADM SARSCOV2 100MCG/0.5ML1ST	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
J	00120	ANESTH EAR SURGERY	4.52	$63.33	---	--	002	0	0	0.00	$0.00	Y
J	00124	ANESTH EAR EXAM	3.61	$50.58	---	--	002	0	0	0.00	$0.00	Y
J	00126	ANESTH TYMPANOTOMY	3.61	$50.58	---	--	002	0	0	0.00	$0.00	Y
1	0012A	ADM SARSCOV2 100MCG/0.5ML2ND	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
1	0013A	FEE COVID-19 VAC 2 BOOSTER	40.00	$40.00	---	--	009	0	0	0.00	$0.00	Y
J	00140	ANESTH PROCEDURES ON EYE	4.52	$63.33	---	--	002	0	0	0.00	$0.00	Y
J	00142	ANESTH LENS SURGERY	5.42	$75.93	---	--	002	0	0	0.00	$0.00	Y
J	00144	ANESTH CORNEAL TRANSPLANT	5.42	$75.93	---	--	002	0	0	0.00	$0.00	Y
J	00145	ANESTH VITREORETINAL SURG	5.42	$75.93	---	--	002	0	0	0.00	$0.00	Y
J	00147	ANESTH IRIDECTOMY	3.61	$50.58	---	--	002	0	0	0.00	$0.00	Y
J	00148	ANESTH EYE EXAM	3.61	$50.58	---	--	002	0	0	0.00	$0.00	Y
M	0014M	LIVER DS ALYS 3 BMRK SRM ALG	140.95	$140.95	---	--	009	0	0	0.00	$0.00	N
J	00160	ANESTH NOSE/SINUS SURGERY	4.52	$63.33	---	--	002	0	0	0.00	$0.00	Y
J	00162	ANESTH NOSE/SINUS SURGERY	6.32	$88.54	---	--	002	0	0	0.00	$0.00	Y
J	00164	ANESTH BIOPSY OF NOSE	3.61	$50.58	---	--	002	0	0	0.00	$0.00	Y
M	0016U	ONC HMTLMF NEO RNA BCR/ABL1	131.17	$131.17	---	--	009	0	0	0.00	$0.00	N
J	00170	ANESTH PROCEDURE ON MOUTH	4.52	$63.33	---	--	002	0	0	0.00	$0.00	Y
J	00172	ANESTH CLEFT PALATE REPAIR	5.42	$75.93	---	--	002	0	0	0.00	$0.00	Y

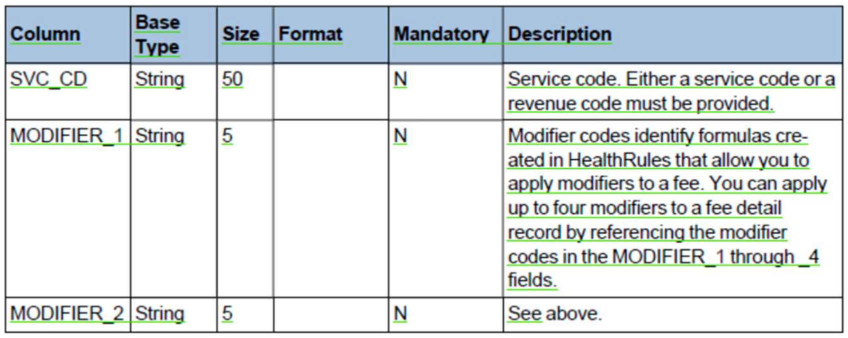
Rates | Conversion Factors | +

Sample record and Excepted Output:

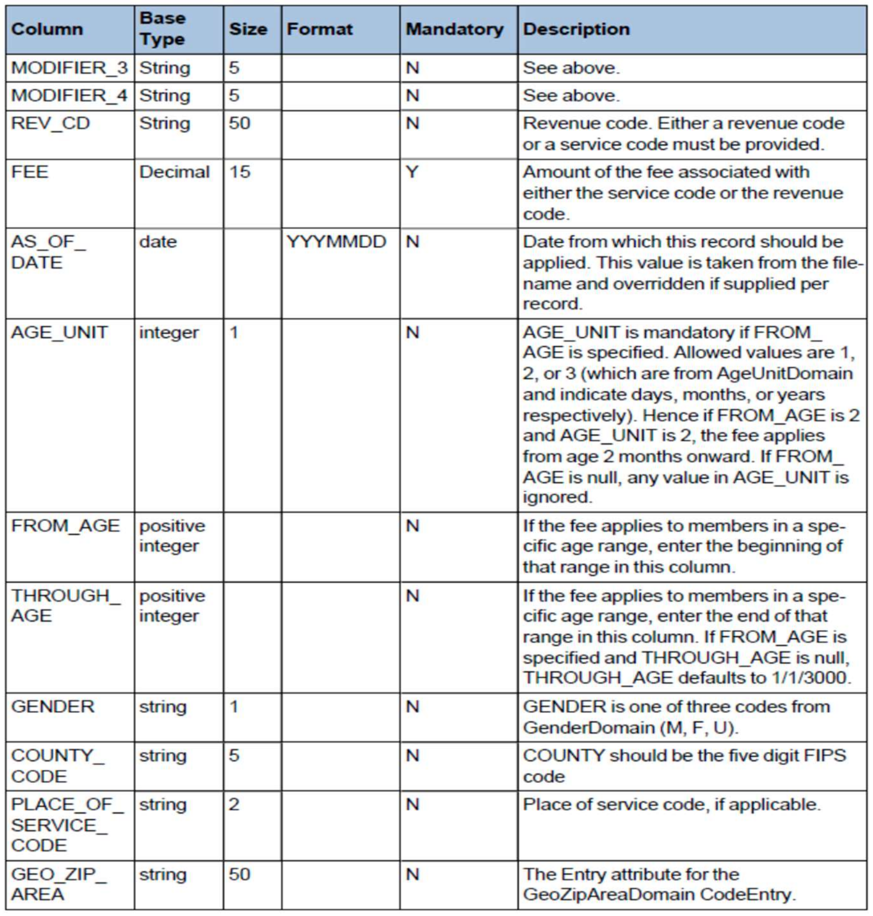
N	92587	EVOKED AUDITORY TEST LIMITED	4.10	$41.00	$44.73	$50.92	053	1	0	0.15	$0.00	Y

```
Basic         92587||41.0||||
Basic-Prof    92587|26|6.15||||
Basic- Tech   92587|TC|34.85||||

Child         92587||44.73|3|0|18|
Child-Prof    92587|26|6.71|3|0|18|
Child- Tech   92587|TC|38.02|3|0|18|

ER            92587||50.92||||23
ER - Prof     92587|26|7.64||||23
ER -Tech      92587|TC|43.28||||23
```

HRP Data format:



Column	Base Type	Size	Format	Mandatory	Description
SVC_CD	String	50		N	Service code. Either a service code or a revenue code must be provided.
MODIFIER_1	String	5		N	Modifier codes identify formulas created in HealthRules that allow you to apply modifiers to a fee. You can apply up to four modifiers to a fee detail record by referencing the modifier codes in the MODIFIER_1 through _4 fields.
MODIFIER_2	String	5		N	See above.



Column	Base Type	Size	Format	Mandatory	Description
MODIFIER_3	String	5		N	See above.
MODIFIER_4	String	5		N	See above.
REV_CD	String	50		N	Revenue code. Either a revenue code or a service code must be provided.
FEE	Decimal	15		Y	Amount of the fee associated with either the service code or the revenue code.
AS_OF_ DATE	date		YYYMMDD	N	Date from which this record should be applied. This value is taken from the file-name and overridden if supplied per record.
AGE_UNIT	integer	1		N	AGE_UNIT is mandatory if FROM_ AGE is specified. Allowed values are 1, 2, or 3 (which are from AgeUnitDomain and indicate days, months, or years respectively). Hence if FROM_AGE is 2 and AGE_UNIT is 2, the fee applies from age 2 months onward. If FROM_ AGE is null, any value in AGE_UNIT is ignored.
FROM_AGE	positive integer			N	If the fee applies to members in a specific age range, enter the beginning of that range in this column.
THROUGH_ AGE	positive integer			N	If the fee applies to members in a specific age range, enter the end of that range in this column. If FROM_AGE is specified and THROUGH_AGE is null, THROUGH_AGE defaults to 1/1/3000.
GENDER	string	1		N	GENDER is one of three codes from GenderDomain (M, F, U).
COUNTY_ CODE	string	5		N	COUNTY should be the five digit FIPS code
PLACE_OF_ SERVICE_ CODE	string	2		N	Place of service code, if applicable.
GEO_ZIP_ AREA	string	50		N	The Entry attribute for the GeoZipAreaDomain CodeEntry.

Overall code Summary:

- **Configuration Loading and Logging Setup**:
  - Loads configurations from config.json.
  - Sets up logging based on configurations.
- **Variable Initialization**:

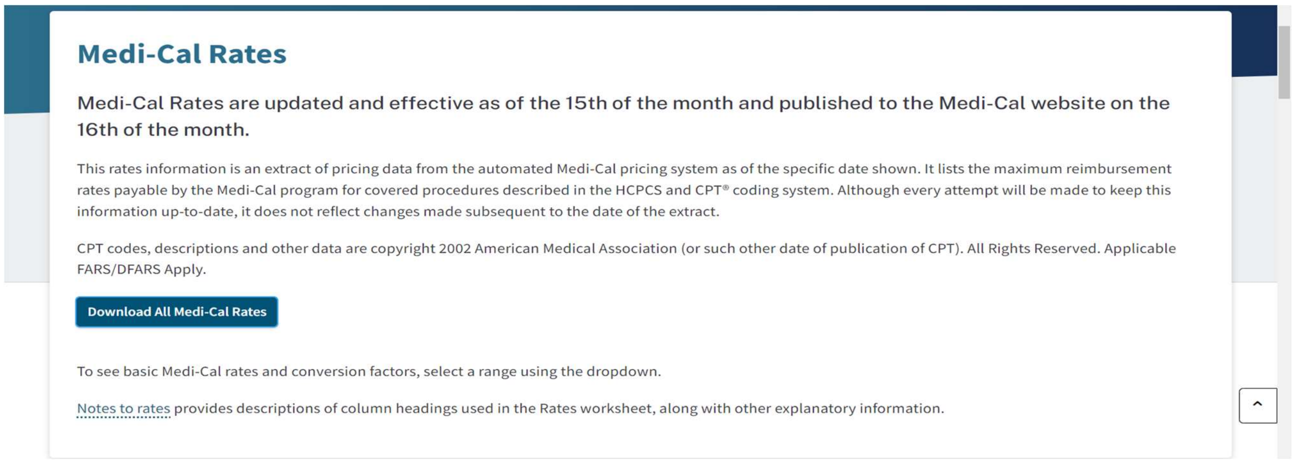
    - Initializes variables for URL, current date, filenames, and extraction paths.
- **File Download and Extraction**:
    - Downloads a ZIP file and extracts its contents.
- **Data Processing Functions**:
    - Defines functions for calculating various rates (Purchase_Calculation, Rental_Calculation, Basic_Rate_Calculation, Child_Rate_Calculation, ER_Rate_Calculation, anes_calculation).

- **DataFrame Initialization and Directory Setup**:
    - Initializes dictionaries to store DataFrames for each Proc_Type.
    - Creates directories for output files.
- **Filtering and Rate Calculation**:
    - Filters DataFrames based on Proc_Type and Proc_Code.
    - Calculates various rates and concatenates results.
    - Saves results to text files with specific naming conventions.
    - Logs the successful creation of files.

The code automates the process of downloading, extracting, processing, and saving medical procedure data into various formatted text files. It ensures thorough logging for monitoring and debugging purposes.

Assumptions:

- **URL Updates:** The URL used for the download button is subject to change in the future. Please ensure to verify and update the URL as necessary.



Medi-Cal Rates

Medi-Cal Rates are updated and effective as of the 15th of the month and published to the Medi-Cal website on the 16th of the month.

This rates information is an extract of pricing data from the automated Medi-Cal pricing system as of the specific date shown. It lists the maximum reimbursement rates payable by the Medi-Cal program for covered procedures described in the HCPCS and CPT® coding system. Although every attempt will be made to keep this information up-to-date, it does not reflect changes made subsequent to the date of the extract.

CPT codes, descriptions and other data are copyright 2002 American Medical Association (or such other date of publication of CPT). All Rights Reserved. Applicable FARS/DFARS Apply.

To see basic Medi-Cal rates and conversion factors, select a range using the dropdown.

Notes to rates provides descriptions of column headings used in the Rates worksheet, along with other explanatory information.

Right-Click on the (Download All Medi-Cal Rates) button and Select Copy Link Address for the URL. Paste the URL in Config.json.

- **Email Credentials:** The 'From' email, 'To' email, and the passkey must be updated with the credentials of the individual running the code to generate output files. Ensure these details are current and accurate.
- **Software Requirements:** It is essential to ensure that all the latest versions of Python and its corresponding libraries are installed and up to date. This guarantees compatibility and functionality of the scripts.