**Health Benefits PDF Data Extraction Guide**

**Overview**

This document provides a complete guide for implementing an automated solution to extract health benefits data from PDF documents and load it into structured Excel templates. The solution uses Python and OpenAI's API to accurately identify and format benefit information according to specific requirements.

**Table of Contents**

1. [Technical Requirements](https://claude.ai/chat/de8ab646-2374-4b0b-8621-a8539e73449c#technical-requirements)
2. [Installation Instructions](https://claude.ai/chat/de8ab646-2374-4b0b-8621-a8539e73449c#installation-instructions)
3. [Usage Guide](https://claude.ai/chat/de8ab646-2374-4b0b-8621-a8539e73449c#usage-guide)
4. [Understanding the Code](https://claude.ai/chat/de8ab646-2374-4b0b-8621-a8539e73449c#understanding-the-code)
5. [Troubleshooting](https://claude.ai/chat/de8ab646-2374-4b0b-8621-a8539e73449c#troubleshooting)

**Technical Requirements**

* Python 3.7 or higher
* Required Python libraries:
  + pypdf
  + pandas
  + openai
* OpenAI API key (paid account required)
* Excel template file with proper formatting

**Installation Instructions**

1. **Install Python**:
   * Download and install Python from [python.org](https://python.org/)
   * Make sure to check "Add Python to PATH" during installation
2. **Install Required Libraries**:
3. pip install pypdf pandas openai
4. **Save the Python Script**:
   * Create a new file named benefits\_extraction.py
   * Copy and paste the full code from the "Python Script" section below
5. **Set Up OpenAI API**:
   * Create an account at [OpenAI](https://platform.openai.com/)
   * Generate an API key in your account settings
   * Keep this key secure as it will be used in the script

**Usage Guide**

1. **Prepare Your Files**:
   * Place all PDF files in a single directory
   * Have your Excel template ready (must have a sheet named 'HEALTH')
2. **Run the Script**:
3. python benefits\_extraction.py --directory /path/to/pdfs --template /path/to/template.xlsx --api\_key YOUR\_API\_KEY
4. **Output**:
   * For each PDF processed, a new Excel file will be created in the same directory
   * The naming convention will be benefits\_data\_1.xlsx, benefits\_data\_2.xlsx, etc.

**Understanding the Code**

The script performs the following operations:

1. **PDF Text Extraction**: Uses PyPDF2 to extract all text from PDF documents
2. **AI-Powered Data Extraction**: Sends the text to OpenAI's API with a detailed prompt that specifies all the fields to extract and formatting rules to follow
3. **Data Formatting**: Applies specific formatting rules based on plan type (HSA vs non-HSA) and benefit type
4. **Excel Integration**: Loads the extracted data into your Excel template at the correct cell positions
5. **Batch Processing**: Can process multiple PDF documents in sequence

**Troubleshooting**

* **PDF Text Extraction Issues**: If the PDFs are scanned documents or have complex layouts, the text extraction might be incomplete. Consider using OCR tools before processing.
* **API Rate Limits**: OpenAI has rate limits. If processing many documents, you might need to add delays between requests.
* **Excel Template Compatibility**: Ensure your Excel template has a 'HEALTH' sheet and the proper cell structure as described in your requirements.

**Python Script**

"""

Health Benefits Data Extraction Framework

This script extracts health benefits data from PDFs according to specific formatting rules

and loads them into a structured Excel template for mass upload.

"""

import os

import json

import pandas as pd

from pypdf import PdfReader

from openai import OpenAI

import re

def extract\_text\_from\_pdf(pdf\_path):

"""Extract all text from a PDF file."""

reader = PdfReader(pdf\_path)

text = ""

for page in reader.pages:

text += page.extract\_text() + "\n"

return text

def extract\_benefits\_data\_with\_ai(text, api\_key):

"""Use OpenAI to extract structured health benefits data based on specified fields."""

client = OpenAI(api\_key=api\_key)

# Define the extraction prompt with all the benefit fields we need

prompt = """

Extract the following health insurance benefits data from the provided document:

1. Carrier Name

2. Plan Name

3. Network Type (PPO, HMO, EPO, POS, etc.)

4. Network Name

Deductibles and Out-of-Pocket:

5. In-network individual deductible (number only)

6. Out-of-network individual deductible (number only)

7. In-network family deductible (number only)

8. Out-of-network family deductible (number only)

9. In-network coinsurance percentage (number only)

10. Out-of-network coinsurance percentage (number only)

11. In-network individual out-of-pocket maximum (number only)

12. Out-of-network individual out-of-pocket maximum (number only)

13. In-network family out-of-pocket maximum (number only)

14. Out-of-network family out-of-pocket maximum (number only)

15. Deductible type (Embedded or Aggregate)

Visits and Services:

16. In-network Primary Care Physician (PCP) visit cost

17. Out-of-network Primary Care Physician (PCP) visit cost

18. In-network Specialist visit cost

19. Out-of-network Specialist visit cost

20. In-network Urgent Care cost

21. Out-of-network Urgent Care cost

22. In-network Emergency Room cost

23. Out-of-network Emergency Room cost

24. In-network Preventive Services cost

25. Out-of-network Preventive Services cost

Procedures and Diagnostics:

26. In-network Outpatient Surgery cost (note any freestanding vs hospital differences)

27. Out-of-network Outpatient Surgery cost

28. In-network Inpatient Hospitalization/Surgery cost

29. Out-of-network Inpatient Hospitalization/Surgery cost

30. In-network CT Scan, PT Scan, MRI cost (note any freestanding vs hospital differences)

31. Out-of-network CT Scan, PT Scan, MRI cost

32. In-network Hospital Newborn Delivery cost

33. Out-of-network Hospital Newborn Delivery cost

Prescription Benefits:

34. In-network Prescription deductible

35. Out-of-network Prescription deductible

36. In-network Generic (Tier 1) cost

37. Out-of-network Generic (Tier 1) cost

38. In-network Brand Name (Tier 2) cost

39. Out-of-network Brand Name (Tier 2) cost

40. In-network Non-Preferred (Tier 3) cost

41. Out-of-network Non-Preferred (Tier 3) cost

42. In-network Specialty (Tier 4) cost

43. Out-of-network Specialty (Tier 4) cost

44. In-network Specialty (Tier 5) cost if available

45. Out-of-network Specialty (Tier 5) cost if available

46. In-network Mail Order (90-day supply) costs for all available tiers

47. Out-of-network Mail Order (90-day supply) costs for all available tiers

Contact Information:

48. Member Website URL

49. Customer Service Phone Number

Follow these rules when extracting the data:

- For costs that are percentage coinsurance, include "after deductible" after the percentage (e.g., "20% after deductible").

- For HSA plans (if "HSA" appears in the plan name), add "after deductible" after dollar amounts.

- For non-HSA plans with just a copay amount, list only the dollar amount without "after deductible".

- For outpatient surgery or major diagnostics with different costs for freestanding vs hospital settings, format as "Freestanding: X / Hospital: Y".

- For per occurrence deductibles, format as "$X, then Y% after deductible".

- For multiple tiers in prescription benefits, separate with "/".

- For specialty medications within a tier, format as "$X (Specialty: $Y)".

- In network preventive services should always be "0%".

- Emergency Room out-of-network value should match the in-network value.

- For designated network/in-network/out-of-network structures with 3 columns, format the in-network as "Designated: X / In-network: Y".

Return the data in a JSON format.

If a field is not found or not applicable, set its value to "N/A".

Text from document:

{text}

"""

response = client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "system", "content": "You are a health benefits data extraction specialist. Extract benefits data accurately following specific formatting rules."},

{"role": "user", "content": prompt}

],

response\_format={"type": "json\_object"}

)

# Parse the JSON response

return json.loads(response.choices[0].message.content)

def format\_benefit\_value(value, plan\_name, is\_hsa\_plan=None):

"""Apply formatting rules based on plan type and benefit."""

if value == "N/A":

return value

# Determine if plan is HSA if not explicitly provided

if is\_hsa\_plan is None:

is\_hsa\_plan = "HSA" in plan\_name if plan\_name else False

# Check if it's a simple dollar amount copay

if re.match(r'^\$\d+$', value):

if is\_hsa\_plan:

return f"{value} after deductible"

return value

return value

def load\_data\_to\_excel(data, template\_path, output\_path):

"""Load the extracted benefits data into the Excel template."""

# Load the template

try:

workbook = pd.ExcelFile(template\_path)

health\_df = pd.read\_excel(workbook, sheet\_name='HEALTH')

# Determine the next available column for the plan data

# Assuming we start with column D for the first plan

# Find the last non-empty column in row 4 (Carrier Name row)

last\_col = None

for col in health\_df.columns[3:]: # Start from column D (index 3)

if pd.isna(health\_df.loc[3, col]):

last\_col = col

break

if last\_col is None:

# If all columns are filled, add a new column

last\_col\_idx = len(health\_df.columns)

health\_df.insert(last\_col\_idx, f"New\_Col\_{last\_col\_idx}", "")

last\_col = f"New\_Col\_{last\_col\_idx}"

col\_idx = health\_df.columns.get\_loc(last\_col)

# Plan Info

health\_df.iloc[3, col\_idx] = data.get('Carrier Name', 'N/A')

health\_df.iloc[4, col\_idx] = data.get('Plan Name', 'N/A')

health\_df.iloc[5, col\_idx] = "Health Insurance"

health\_df.iloc[6, col\_idx] = "Health insurance provides financial protection against medical costs. It helps employees access necessary healthcare while minimizing out-of-pocket expenses."

# Determine if plan is HSA

is\_hsa\_plan = "HSA" in data.get('Plan Name', '')

# Deductibles

health\_df.iloc[8, col\_idx] = data.get('In-network individual deductible', 'N/A')

health\_df.iloc[8, col\_idx + 1] = data.get('Out-of-network individual deductible', 'N/A')

health\_df.iloc[9, col\_idx] = data.get('In-network family deductible', 'N/A')

health\_df.iloc[9, col\_idx + 1] = data.get('Out-of-network family deductible', 'N/A')

# Coinsurance

health\_df.iloc[11, col\_idx] = data.get('In-network coinsurance percentage', 'N/A')

health\_df.iloc[11, col\_idx + 1] = data.get('Out-of-network coinsurance percentage', 'N/A')

# Out of Pocket Maximum

health\_df.iloc[13, col\_idx] = data.get('In-network individual out-of-pocket maximum', 'N/A')

health\_df.iloc[13, col\_idx + 1] = data.get('Out-of-network individual out-of-pocket maximum', 'N/A')

health\_df.iloc[14, col\_idx] = data.get('In-network family out-of-pocket maximum', 'N/A')

health\_df.iloc[14, col\_idx + 1] = data.get('Out-of-network family out-of-pocket maximum', 'N/A')

# Doctor visits

health\_df.iloc[16, col\_idx] = format\_benefit\_value(data.get('In-network Primary Care Physician (PCP) visit cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[16, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Primary Care Physician (PCP) visit cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[17, col\_idx] = format\_benefit\_value(data.get('In-network Specialist visit cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[17, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Specialist visit cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[18, col\_idx] = format\_benefit\_value(data.get('In-network Urgent Care cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[18, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Urgent Care cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[19, col\_idx] = format\_benefit\_value(data.get('In-network Emergency Room cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[19, col\_idx + 1] = format\_benefit\_value(data.get('In-network Emergency Room cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan) # Same as in-network

# Preventive Care

health\_df.iloc[21, col\_idx] = "0%" # Always 0% for in-network

health\_df.iloc[21, col\_idx + 1] = data.get('Out-of-network Preventive Services cost', 'N/A')

# Surgeries and Hospital

health\_df.iloc[23, col\_idx] = data.get('In-network Outpatient Surgery cost', 'N/A')

health\_df.iloc[23, col\_idx + 1] = data.get('Out-of-network Outpatient Surgery cost', 'N/A')

health\_df.iloc[24, col\_idx] = data.get('In-network Inpatient Hospitalization/Surgery cost', 'N/A')

health\_df.iloc[24, col\_idx + 1] = data.get('Out-of-network Inpatient Hospitalization/Surgery cost', 'N/A')

health\_df.iloc[25, col\_idx] = data.get('In-network CT Scan, PT Scan, MRI cost', 'N/A')

health\_df.iloc[25, col\_idx + 1] = data.get('Out-of-network CT Scan, PT Scan, MRI cost', 'N/A')

# Newborn delivery (same as inpatient)

health\_df.iloc[26, col\_idx] = data.get('In-network Inpatient Hospitalization/Surgery cost', 'N/A')

health\_df.iloc[26, col\_idx + 1] = data.get('Out-of-network Inpatient Hospitalization/Surgery cost', 'N/A')

# Prescription Drugs

health\_df.iloc[28, col\_idx] = data.get('In-network Prescription deductible', 'N/A')

health\_df.iloc[28, col\_idx + 1] = data.get('Out-of-network Prescription deductible', 'N/A')

health\_df.iloc[29, col\_idx] = format\_benefit\_value(data.get('In-network Generic (Tier 1) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[29, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Generic (Tier 1) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[30, col\_idx] = format\_benefit\_value(data.get('In-network Brand Name (Tier 2) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[30, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Brand Name (Tier 2) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[31, col\_idx] = format\_benefit\_value(data.get('In-network Non-Preferred (Tier 3) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[31, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Non-Preferred (Tier 3) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[32, col\_idx] = format\_benefit\_value(data.get('In-network Specialty (Tier 4) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[32, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Specialty (Tier 4) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[33, col\_idx] = format\_benefit\_value(data.get('In-network Specialty (Tier 5) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[33, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Specialty (Tier 5) cost', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[34, col\_idx] = format\_benefit\_value(data.get('In-network Mail Order (90-day supply) costs', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

health\_df.iloc[34, col\_idx + 1] = format\_benefit\_value(data.get('Out-of-network Mail Order (90-day supply) costs', 'N/A'), data.get('Plan Name', ''), is\_hsa\_plan)

# Plan Year and Other Info

from datetime import datetime

current\_year = datetime.now().year

health\_df.iloc[36, col\_idx] = current\_year

health\_df.iloc[37, col\_idx] = "Calendar Year: January 1st -- December 31st"

health\_df.iloc[38, col\_idx] = data.get('Deductible type', 'N/A')

health\_df.iloc[39, col\_idx] = data.get('Network Type', 'N/A')

health\_df.iloc[40, col\_idx] = data.get('Network Name', 'N/A')

health\_df.iloc[41, col\_idx] = data.get('Member Website URL', 'N/A')

health\_df.iloc[42, col\_idx] = data.get('Customer Service Phone Number', 'N/A')

# Save to new file

health\_df.to\_excel(output\_path, sheet\_name='HEALTH', index=False)

return True

except Exception as e:

print(f"Error loading data to Excel: {e}")

return False

def process\_pdf\_benefits(pdf\_path, template\_path, output\_path, api\_key):

"""Process a PDF file to extract benefits data and load into Excel template."""

try:

print(f"Processing {pdf\_path}...")

# Extract text from PDF

text = extract\_text\_from\_pdf(pdf\_path)

# Use AI to extract structured benefits data

benefits\_data = extract\_benefits\_data\_with\_ai(text, api\_key)

# Load data into Excel template

success = load\_data\_to\_excel(benefits\_data, template\_path, output\_path)

if success:

print(f"Successfully processed {pdf\_path} and saved to {output\_path}")

return output\_path

else:

print(f"Failed to load data from {pdf\_path} into Excel")

return None

except Exception as e:

print(f"Error processing {pdf\_path}: {e}")

return None

def process\_directory(directory\_path, template\_path, api\_key):

"""Process all PDFs in a directory and output structured data."""

results = []

# Get list of PDF files

pdf\_files = [f for f in os.listdir(directory\_path) if f.lower().endswith('.pdf')]

if not pdf\_files:

print("No PDF files found in the directory.")

return

for i, filename in enumerate(pdf\_files):

pdf\_path = os.path.join(directory\_path, filename)

output\_filename = f"benefits\_data\_{i+1}.xlsx"

output\_path = os.path.join(directory\_path, output\_filename)

result = process\_pdf\_benefits(pdf\_path, template\_path, output\_path, api\_key)

if result:

results.append(result)

print(f"Processing complete. {len(results)} files successfully processed.")

return results

def main():

import argparse

parser = argparse.ArgumentParser(description='Extract health benefits data from PDFs using AI')

parser.add\_argument('--directory', required=True, help='Directory containing PDF files')

parser.add\_argument('--template', required=True, help='Path to Excel template file')

parser.add\_argument('--api\_key', required=True, help='OpenAI API key')

args = parser.parse\_args()

process\_directory(args.directory, args.template, args.api\_key)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Example Usage**

Here's how you would use the script to process PDFs:

# Basic usage

python benefits\_extraction.py --directory C:\benefits\pdfs --template C:\benefits\template.xlsx --api\_key sk-your-api-key-here

# For Mac/Linux users

python benefits\_extraction.py --directory /Users/username/benefits/pdfs --template /Users/username/benefits/template.xlsx --api\_key sk-your-api-key-here

**Alternative Solutions**

If you prefer not to use Python, consider these alternatives:

1. **Commercial OCR/Data Extraction Services**:
   * ABBYY FlexiCapture
   * Kofax
   * DocuWare
2. **Low-Code Automation Platforms**:
   * UiPath
   * Automation Anywhere
   * Microsoft Power Automate
3. **Document AI Platforms**:
   * Google Document AI
   * Amazon Textract + Comprehend
   * Microsoft Azure Form Recognizer