

Do California's CCPA ADMT rules cover employees using Claude?

What the CPPA's approved text says about human involvement, notices, opt-outs and request deadlines, and a DuckDB script that checks the records you keep.

Published 2026-10-31 · Online version and downloads: www.datameaning.com/guides/ccpa-admt-rules-and-claude-at-work/

California's rules ask whether a business uses technology's output to make an employment decision, such as hiring or pay, without human involvement. Human involvement requires a reviewer who knows how to interpret and use the output, reviews and analyzes it, and has the authority to make or change the decision. Whether your Claude use fits is for Legal to decide.

What you will need

- A register of your Claude uses in HR as a CSV with a header row: an id; the decision the use supports (hiring, promotion, pay or work assignment, discipline or termination, or neither); a yes, no or n/a for each of A, B and C; the date the use began; the date a Pre-use Notice was given; whether you offer an opt-out method; which exception from § 7221(b) you rely on, or none; whether its conditions are recorded as met; whether the notice names it; and the date the row was created.
- A log of opt-out, access and appeal requests as a CSV with a header row: an id, the register id of the use, the kind, the date received, the date confirmed, the date answered or, for an opt-out, the date processing ceased, the date of any extension notice, and the date the record was created. These are records about people: keep them under your own access controls. The script runs on your machine and sends none of your records to Data Meaning or anyone else; its only network call is the optional `--check-source` fetch of the regulation PDF.
- Python 3 and the `duckdb` module. The script reads each CSV with DuckDB's `read_csv` using a header row and `all_varchar` ([DuckDB: CSV overview](#) (read 2026-10-31)).

The rules the script reads

The California Privacy Protection Agency's regulations on automated decisionmaking technology (ADMT) were approved by the Office of Administrative Law on September 22, 2025, and are effective January 1, 2026 ([CPPA rulemaking page](#) (read 2026-10-31)). This guide quotes the agency's [approved regulation text](#) (read 2026-10-31), with section numbers. The table sets each rule beside the record the script reads to check it.

The rule, quoted from the regulation, and what the script reads

What the regulation says	The record the script reads	What it prints
ADMT is "any technology that processes personal information and uses computation to replace human decisionmaking or substantially replace human decisionmaking". To substantially replace it means a business "uses the technology's output to make a decision without human involvement" (§ 7001(e), (e)(1))	Register: whether the use supports an employment decision, and whether the register records A, B and C as yes	HUMAN INVOLVEMENT RECORDED, or FOR LEGAL TO CHECK naming the element recorded as no or not recorded
Human involvement requires the reviewer to (A) "Know how to interpret and use the technology's output to make the decision"; (B) "Review and analyze the output of the technology, and any other information that is relevant to make or change the decision"; and (C) "Have the authority to make or change the decision based on their analysis in subsection (B)" (§ 7001(e)(1))	Register: one yes or no for each of A, B and C	The element recorded as no, or not recorded
A significant decision includes "employment or independent contracting opportunities or compensation": hiring; allocation or assignment of work and compensation; promotion; and demotion, suspension and termination (§ 7001(ddd), (ddd)(4))	Register: the decision type, mapped to one of those four kinds or to neither	NOT AN EMPLOYMENT USE AS RECORDED, or a use that goes on to the next list
A business using ADMT for a significant decision before January 1, 2027 "must be in compliance with the requirements of this Article no later than January 1, 2027"; from that date, "any time it is using ADMT for a significant decision" (§ 7200(b))	Register: the date the use began	The notice's due date: the later of the first-use date and 2027-01-01
The Pre-use Notice must be "presented prominently and conspicuously to the consumer at or before the point when the business collects the consumer's personal information that the business plans to process using ADMT" (§ 7220(b)(2)). It must describe the opt-out right. For an exception other than the human appeal exception, it must "identify the specific exception it is relying upon" (§ 7220(c)(2)(B)). For the appeal exception, it must instead inform the consumer of their ability to appeal the decision and provide instructions (§ 7220(c)(2)(A))	Register: the date the notice was given, and whether it names the exception	NO PRE-USE NOTICE RECORDED, NOTICE DATED n DAYS AFTER ITS DUE DATE, NOTICE DOES NOT NAME THE EXCEPTION

What the regulation says	The record the script reads	What it prints
A business "must provide consumers with the ability to opt-out" of ADMT used for a significant decision, except in the circumstances of § 7221(b): (1) "The business provides the consumer with a method to appeal the decision to a human reviewer who has the authority to overturn the decision"; to qualify, it must designate a reviewer who "must know how to interpret and use the output of the ADMT that made the significant decision being appealed and must have the authority to change the decision based on their analysis" (§ 7221(b)(1)(A)) and must "Clearly describe to the consumer how to submit an appeal and enable the consumer to provide information to the human reviewer in support of their appeal" (§ 7221(b)(1)(B)); (2) for hiring, if "The business uses the ADMT solely for the business's assessment of the consumer's ability to perform at work or in an educational program to determine whether to admit, accept, or hire them" and "The ADMT works for the business's purpose and does not unlawfully discriminate based upon protected characteristics" (§ 7221(b)(2)(A), (B)); (3) for allocation/assignment of work and compensation, if "The business uses the ADMT solely for the business's allocation/assignment of work or compensation" and the ADMT "works for the business's purpose and does not unlawfully discriminate based upon protected characteristics" (§ 7221(b)(3)(A), (B)). The text sets further requirements for the appeal method in § 7221(b)(1)(A) and (B), which the script does not test (§ 7221(a), (b))	Register: whether an opt-out method exists; which one exception is relied on; whether its conditions are recorded as met	NO OPT-OUT METHOD AND NO EXCEPTION NAMED; an exception that does not match the decision type; conditions not recorded as met
After processing has begun, the business must cease it "as soon as feasibly possible, but no later than 15 business days from the date the business receives the request" (§ 7221(n)(1))	Request log: kind, date received, date processing ceased	NOT PROCESSED, 15 BUSINESS DAYS PASSED, or PROCESSED n DAYS AFTER RECEIPT
No later than 10 business days after receiving a request to access ADMT or to appeal ADMT, a business "shall confirm receipt of the request and provide information about how the business will process the request"; the description of "the business's verification process and when the consumer should expect a response" is not needed "in instances where the business has already granted or denied the request", but the confirmation still is (§ 7021(a)). A response "no later than 45 calendar days after receipt", with up to 45 more if the consumer is given "notice and an explanation", "for a maximum total of 90 calendar days" (§ 7021(a), (b))	Request log: received, confirmed, answered and extension- notice dates	NO CONFIRMATION, ANSWERED n DAYS AFTER RECEIPT (limit 45 or 90), EXTENSION NOTICE DATED AFTER DAY 45
Two limits on the table. The regulation says a consumer may ask to appeal or to access ADMT, and that a business "shall confirm receipt" within 10 business days; it does not define a business day in the text we read, so the script counts Monday to Friday and lets you list other days your counsel		

does not count. And the 15 business days in § 7221(n)(1) are written for a request made after processing began; the script applies them to every opt-out request in the log.

Do the rules reach employees? The regulation's own examples are about employers and employees: it says "an employer may provide a consolidated Pre-use Notice to an employee that addresses the employer's proposed use of productivity monitoring software to determine the employee's allocation/assignment of work and compensation, and to determine which employees will be demoted" (§ 7220(e)(1)). The rules speak of "consumers", a word the Civil Code defines in section 1798.140, which this guide does not quote. Whether a given worker, and a given use of Claude, falls inside is for Legal.

Every step in full

Step 1. Set what your files call things

The settings sit in one block at the top of the script. They hold the file names, the column each field lives in, the answers your files use and what each answer means, the as-of date and a list of days your counsel does not count as business days. The answer lists are ours, and you can disagree with them. Dates must be written YYYY-MM-DD and nothing else: DuckDB's own date cast reads 8/5/26 as the year 8 and accepts 2026/09/24, so the script matches the pattern first and only then parses the date. A date in any other form is counted and fails an input check. It is never dropped.

```
# ---- CHANGE THESE FOR YOUR OWN DATA -----
REGISTER_FILE = None      # your Claude use-case register (CSV with a header row); None = generate
REQUESTS_FILE = None      # your log of opt-out, access and appeal requests (CSV with a header
row); None = generate
# the column of YOUR files that holds each field. Dates must be written YYYY-MM-DD.
REG_COLS = {"use_case_id": "use_case_id", "decision": "decision", "human_a": "human_a", "human_b":
"human_b", "human_c": "human_c",
            "first_used_on": "first_used_on", "notice_on": "notice_on", "opt_out_link":
"opt_out_link", "exception": "exception",
            "exception_conditions": "exception_conditions", "notice_covers":
"notice_covers_exception", "created_on": "created_on"}
REQ_COLS = {"request_id": "request_id", "use_case_id": "use_case_id", "kind": "kind",
"received_on": "received_on",
            "confirmed_on": "confirmed_on", "answered_on": "answered_on", "extension_notice_on":
"extension_notice_on",
            "created_on": "created_on"}
# YOUR answers (lower case, no spaces at the ends) -> the meaning the script gives them. An answer
in your files that is in
# none of these lists is counted and fails an input check; it is never guessed.
DECISION_WORDS = {"hiring": "hiring", "promotion": "promotion", "pay": "work or pay", "work
assignment": "work or pay",
                  "discipline": "demotion, suspension or termination", "demotion": "demotion,
suspension or termination",
                  "suspension": "demotion, suspension or termination", "termination": "demotion,
suspension or termination",
                  "neither": "neither"}
```

```

YESNO_WORDS = {"yes": "yes", "no": "no", "n/a": "n/a"}          # human involvement A, B, C; opt-out
link; exception conditions; notice covers
EXCEPTION_WORDS = {"none": "none", "human appeal": "appeal", "hiring assessment": "hiring", "work
or pay allocation": "work or pay"}
KIND_WORDS = {"opt-out": "opt-out", "access": "access", "appeal": "appeal"}
AS_OF = None           # YYYY-MM-DD: the day the check is for; None = today
# "business day" is not defined in the regulation text we read. Monday to Friday count; list the
other days your counsel does not count.
BUSINESS_HOLIDAYS = []  # YYYY-MM-DD each, for example ["2026-11-26", "2026-12-25"]
# -----

```

Step 2. Sort each use by the three elements

For each register row the script takes the decision type. An empty or unknown one goes to Legal. A use that supports no employment decision is set aside as recorded. For an employment use it reads A, B and C: all three recorded yes is HUMAN INVOLVEMENT RECORDED (A, B, C); any element recorded as no, or empty, or in no list, is FOR LEGAL TO CHECK and names the element. The statuses say what the register records. They do not say what the regulation requires, and a reviewer recorded as having all three is only as good as the record.

```

decision = DECISION_WORDS.get(dec)
if decision is None:
    status, reason = "FOR LEGAL TO CHECK", "decision type not recorded or not recognized"
elif decision == "neither":
    status, reason = "NOT AN EMPLOYMENT USE AS RECORDED", ""
else:
    answers = {"A": YESNO_WORDS.get(a), "B": YESNO_WORDS.get(b), "C": YESNO_WORDS.get(c)}
    missing = [k for k, v in answers.items() if v not in ("yes", "no")]
    no = [k for k, v in answers.items() if v == "no"]
    if not missing and not no:
        status, reason = "HUMAN INVOLVEMENT RECORDED (A, B, C)", ""
    else:
        parts = ([f"{k} not recorded" for k in missing] + [f"{k} recorded as no" for k in
no])
        status, reason = "FOR LEGAL TO CHECK", "may substantially replace human
decisionmaking: " + ", ".join(sorted(parts))

```

Step 3. Check the notice and the opt-out, or the exception named

Only uses marked FOR LEGAL TO CHECK are reviewed. The notice is due on the later of the use's first-use date and January 1, 2027 (§ 7200(b)); the script uses the first-use date as its stand-in for the point at which personal information is collected, which is our proxy. A missing notice is a finding once the as-of date is past the due date and shown as not yet due before it. An opt-out method must exist, or exactly one exception must be named. The exception must match the decision type (the hiring exception is for hiring, the work-or-pay exception for allocation or assignment of work and compensation; the appeal exception fits any), its conditions must be recorded as met, and the notice must name it or, for an appeal, say how to appeal (§ 7220(c)(2)). The script records those conditions as yes or no. It cannot test them.

```

        meaning, optw, condw, coverw = EXCEPTION_WORDS.get(exc), YESNO_WORDS.get(opt),
YESNO_WORDS.get(cond), YESNO_WORDS.get(cover)
        if meaning is None:
            hard.append("exception answer not recognized")
        elif meaning == "none":
            if optw == "no":
                hard.append("NO OPT-OUT METHOD AND NO EXCEPTION NAMED")
            elif optw != "yes":
                hard.append("opt-out method answer not recorded or not recognized")
        else:
            section = {"appeal": "7221(b)(1)", "hiring": "7221(b)(2)", "work or pay": "7221(b)(3)"}
[meaning]
            if meaning == "hiring" and decision != "hiring" or meaning == "work or pay" and
decision != "work or pay":
                hard.append(f"EXCEPTION NAMED ({exc}, {section}) DOES NOT MATCH THE DECISION TYPE
({dec})")
            if condw != "yes":
                hard.append(f"EXCEPTION NAMED ({section}), ITS CONDITIONS NOT RECORDED AS MET")
            if coverw != "yes":
                hard.append("NOTICE DOES NOT SAY HOW TO APPEAL" if meaning == "appeal" else "NOTICE
DOES NOT NAME THE EXCEPTION")

```

Step 4. Check each request against the dates in the text

An opt-out request is due 15 business days after receipt. An access or appeal request needs a confirmation of receipt within 10 business days, even when it was already answered in that time (§ 7021(a)), and an answer within 45 calendar days, or 90 if the log holds an extension notice dated by day 45. A request with no answer is a finding once the as-of date passes its due date and open before that. Business days skip weekends and the holidays you list.

```

def add_business_days(day, n, holidays):
    """The date n business days after `day` (the day itself is day 0): Monday to Friday, not a
listed holiday."""
    d = day
    while n > 0:
        d += datetime.timedelta(days=1)
        if d.weekday() < 5 and d not in holidays:
            n -= 1
    return d

```

```

        elif kind == "opt-out":
            due = add_business_days(received, OPT_OUT_BUSINESS_DAYS, holidays)
            if not a_given:
                (hard if as_of > due else soft).append(f"NOT PROCESSED, 15 BUSINESS DAYS PASSED
(due {due})" if as_of > due else f"open, due {due}")
            elif answered is None:
                hard.append("processed date not read")
            elif answered > due:
                hard.append(f"PROCESSED {(answered - received).days} DAYS AFTER RECEIPT (15
business days: due {due})")

```

```

conf_due = add_business_days(received, CONFIRM_BUSINESS_DAYS, holidays)
limit, limit_days = received + datetime.timedelta(days=RESPOND_DAYS), RESPOND_DAYS
if e_given and ext is None:
    hard.append("extension notice date not read")
elif e_given:
    if ext > received + datetime.timedelta(days=RESPOND_DAYS):
        hard.append(f"EXTENSION NOTICE DATED DAY {(ext - received).days}, AFTER DAY
{RESPOND_DAYS}")
    else:
        limit_days = RESPOND_DAYS_EXTENDED
        limit = received + datetime.timedelta(days=limit_days)
        extended += 1
# 7021(a): receipt must be confirmed within 10 business days even when the request is
already answered; only the
# description of the verification process and timing is not needed then
if not c_given:
    if as_of > conf_due:
        hard.append(f"NO CONFIRMATION, 10 BUSINESS DAYS PASSED (due {conf_due})")
    else:
        soft.append(f"confirmation due {conf_due}")
elif confirmed is None:
    hard.append("confirmation date not read")
elif confirmed > conf_due:
    hard.append(f"CONFIRMED {(confirmed - received).days} DAYS AFTER RECEIPT (10
business days: due {conf_due})")
if not a_given:
    (hard if as_of > limit else soft).append(f"NOT ANSWERED, DEADLINE PASSED (due
{limit})" if as_of > limit else f"open, due {limit}")
elif answered is None:
    hard.append("answer date not read")
elif answered > limit:
    hard.append(f"ANSWERED {(answered - received).days} DAYS AFTER RECEIPT (limit
{limit_days})")

```

Step 5. Run the input checks before trusting any list

- **Ids:** register and request ids listed more than once, ignoring case, and rows with no id (trip on: a duplicated or unnumbered row).
- **Answers in no list:** a decision type, a yes, no or n/a answer, an exception, or a request kind that is in none of your lists (trip on: selection, an empty human-involvement answer, complaint; an empty answer counts).
- **A request that names a use not on the register,** including a blank use id (trips on: a request that names UC-999).
- **Dates not written YYYY-MM-DD, and record dates after the as-of date** (trip on: 6/1/26, or a record dated after the as-of date). A first-use date after the as-of date is not a fault: it is a planned use, and its notice is shown as not yet due.
- **An answer, confirmation or extension notice dated before its request was received** (trips on: a request answered two days before it arrived). An extension notice dated before receipt

is a typo, and counting it would quietly allow 90 days.

- **The regulation PDF** differs from its pinned size or SHA-256, or cannot be fetched, when you run with `--check-source`. No run here trips that check, and the holiday-format check is not tripped either.

A word in your list that no row uses is printed as information. A failed input check marks every list INCOMPLETE.

The code

The complete download, `ccpa_admt_claude_check.py`, exactly as published at www.datameaning.com/guides/ccpa-admt-rules-and-claude-at-work/ccpa_admt_claude_check.py.

```
#!/usr/bin/env python3
"""California's CCPA rules on automated decisionmaking and Claude at work: the records a company
would keep, checked in DuckDB.

python3 ccpa_admt_claude_check.py --as-of 2026-10-31 --check-source          # clean generated
records, and re-check the pinned regulation PDF
python3 ccpa_admt_claude_check.py --plant --as-of 2027-03-01              # planted faults
python3 ccpa_admt_claude_check.py --strict ...                            # exit 1 when an input check failed or a
notice, opt-out, request or record needs follow-up;
                                                                    # FOR LEGAL TO CHECK uses alone do not,
they are a prompt for Legal
python3 ccpa_admt_claude_check.py --work-dir some/folder                  # where generated files and results
are written

Builds three lists, each printed with its filters, row count and a SHA-256 of its rows, and each
written as a CSV:
    use sort                    every row of your Claude use-case register. An employment use is sorted by the
three things the regulation says a human
                                reviewer must have (section 7001(e)(1)(A), (B), (C)) as YOU recorded them: all
three recorded yes is HUMAN INVOLVEMENT
                                RECORDED; anything else, including an empty or unknown answer, is FOR LEGAL TO
CHECK and names the element.
    notice review              for each FOR LEGAL TO CHECK employment use: a Pre-use Notice recorded (section
7220) and dated on time (section 7200(b)), and
                                an opt-out method, or the one exception from section 7221(b) you rely on, named,
matched to the decision type, with its
                                conditions and the notice's mention of it recorded
    requests                   every opt-out, access and appeal request, against the deadlines the regulation
states: opt-out processing within 15 business
                                days (7221(n)(1)); a confirmation within 10 business days and an answer within 45
calendar days, or 90 with a notice
                                of extension (7021(a), (b))
Input checks run first (unreadable dates, answers in no list, duplicate ids, a use that is not on
the register); a failed check marks the
lists INCOMPLETE.

The regulation is the CPPA's approved text, effective January 1, 2026
(https://cppa.ca.gov/regulations/ccpa\_updates.html). --check-source fetches
the PDF and compares its byte size and SHA-256 with the pins below. This is NOT legal advice and
```

```

NOT a compliance test. Legal decides whether a
use is ADMT, whether the rules reach employees, and what the exceptions require. The generated
records are NOT a real organization's data.
"""

import csv
import datetime
import hashlib
import pathlib
import random
import re
import sys
import urllib.request

import duckdb

# ---- CHANGE THESE FOR YOUR OWN DATA -----
REGISTER_FILE = None      # your Claude use-case register (CSV with a header row); None = generate
REQUESTS_FILE = None      # your log of opt-out, access and appeal requests (CSV with a header
row); None = generate
# the column of YOUR files that holds each field. Dates must be written YYYY-MM-DD.
REG_COLS = {"use_case_id": "use_case_id", "decision": "decision", "human_a": "human_a", "human_b":
"human_b", "human_c": "human_c",
            "first_used_on": "first_used_on", "notice_on": "notice_on", "opt_out_link":
"opt_out_link", "exception": "exception",
            "exception_conditions": "exception_conditions", "notice_covers":
"notice_covers_exception", "created_on": "created_on"}
REQ_COLS = {"request_id": "request_id", "use_case_id": "use_case_id", "kind": "kind",
"received_on": "received_on",
            "confirmed_on": "confirmed_on", "answered_on": "answered_on", "extension_notice_on":
"extension_notice_on",
            "created_on": "created_on"}
# YOUR answers (lower case, no spaces at the ends) -> the meaning the script gives them. An answer
in your files that is in
# none of these lists is counted and fails an input check; it is never guessed.
DECISION_WORDS = {"hiring": "hiring", "promotion": "promotion", "pay": "work or pay", "work
assignment": "work or pay",
                  "discipline": "demotion, suspension or termination", "demotion": "demotion,
suspension or termination",
                  "suspension": "demotion, suspension or termination", "termination": "demotion,
suspension or termination",
                  "neither": "neither"}
YESNO_WORDS = {"yes": "yes", "no": "no", "n/a": "n/a"}      # human involvement A, B, C; opt-out
link; exception conditions; notice covers
EXCEPTION_WORDS = {"none": "none", "human appeal": "appeal", "hiring assessment": "hiring", "work
or pay allocation": "work or pay"}
KIND_WORDS = {"opt-out": "opt-out", "access": "access", "appeal": "appeal"}
AS_OF = None        # YYYY-MM-DD: the day the check is for; None = today
# "business day" is not defined in the regulation text we read. Monday to Friday count; list the
other days your counsel does not count.
BUSINESS_HOLIDAYS = []    # YYYY-MM-DD each, for example ["2026-11-26", "2026-12-25"]
# -----

DUTY_FROM = datetime.date(2027, 1, 1)    # section 7200(b): a business using ADMT for a significant
decision before this date must comply no later than it
OPT_OUT_BUSINESS_DAYS = 15                # section 7221(n)(1): cease processing "no later than 15
business days from the date the business receives the request"
CONFIRM_BUSINESS_DAYS = 10                # section 7021(a): confirm receipt "no later than 10
business days after receiving" the request

```

```

RESPOND_DAYS = 45 # section 7021(b): respond "no later than 45 calendar days
after receipt"
RESPOND_DAYS_EXTENDED = 90 # section 7021(b): up to 45 more days, "a maximum total of
90 calendar days", with notice and an explanation
REG_URL = "https://cppa.ca.gov/regulations/pdf/ccpa_updates_cyber_risk_admt_appr_text.pdf"
REG_BYTES = 666571
REG_SHA256 = "7a34306cebf12ae9050490568b1d7ed532cfd38dc6ed8c7c3dc40afb23328650"
USER_AGENT = "ccpa_admt_claude_check.py (https://www.datameaning.com/guides/ccpa-admt-rules-and-
claude-at-work/; info@datameaning.com)"
SOURCES = [("CPPA approved regulation text", REG_URL), ("CPPA rulemaking page",
"https://cppa.ca.gov/regulations/ccpa_updates.html")]
EMPLOYMENT = {"hiring", "promotion", "work or pay", "demotion, suspension or termination"}

def qcol(name):
    return ''' + name.replace('\'', '""') + '''

def strict_date(col):
    """YYYY-MM-DD only: '8/5/26' and '2026/09/24' are not dates here, and neither is '2026-02-
30'."""
    c = f"trim(coalesce({qcol(col)}, ''))"
    pattern = "'\\d{4}-\\d{2}-\\d{2}'"
    return ("CASE WHEN regexp_full_match(" + c + ", " + pattern + ") THEN try_strptime(" + c + ",
'%Y-%m-%d')::DATE END",
        c + " <> ''")

def read_day(text):
    if text is None or not re.fullmatch(r"\d{4}-\d{2}-\d{2}", str(text).strip()):
        return None
    try:
        return datetime.date.fromisoformat(str(text).strip())
    except ValueError:
        return None

def add_business_days(day, n, holidays):
    """The date n business days after `day` (the day itself is day 0): Monday to Friday, not a
listed holiday."""
    d = day
    while n > 0:
        d += datetime.timedelta(days=1)
        if d.weekday() < 5 and d not in holidays:
            n -= 1
    return d

def generate(mode, gen_dir, as_of):
    """Seeded stand-in for an organization's records. mode: clean or plant."""
    plant = mode == "plant"
    rnd = random.Random(227)
    gen_dir.mkdir(parents=True, exist_ok=True)
    iso = lambda d: d.isoformat() if d else ""
    day = datetime.date.fromisoformat
    header = ["use_case_id", "use_case", "decision", "human_a", "human_b", "human_c",
"first_used_on", "notice_on", "opt_out_link",
        "exception", "exception_conditions", "notice_covers_exception", "created_on"]

```

```

    reg = [{"UC-301", "Summarize resumes for recruiters", "hiring", "yes", "yes", "yes", "2026-02-02", "2026-01-20", "yes", "none", "n/a", "n/a"},
           ["UC-302", "Rank resumes against a job description", "hiring", "yes", "no", "yes", "2026-03-02", "2026-02-20", "no", "hiring assessment", "yes", "yes"],
           ["UC-303", "Draft job postings", "neither", "n/a", "n/a", "n/a", "2026-01-15", "", "n/a", "none", "n/a", "n/a"],
           ["UC-304", "Suggest a promotion shortlist", "promotion", "yes", "yes", "yes", "2026-04-01", "2026-03-15", "yes", "none", "n/a", "n/a"],
           ["UC-305", "Answer staff questions about HR policy", "neither", "n/a", "n/a", "n/a", "2026-01-15", "", "n/a", "none", "n/a", "n/a"],
           ["UC-306", "Draft performance review summaries used for pay", "pay", "yes", "yes", "no", "2026-05-04", "2026-04-20", "yes", "none", "n/a", "n/a"],
           ["UC-307", "Suggest shift assignments", "work assignment", "yes", "no", "yes", "2026-06-01", "2026-05-15", "no", "work or pay allocation", "yes", "yes"]]
    reg = [r + ["2026-01-02"] for r in reg]
    if plant:
        reg += [{"UC-308", "Score interview notes", "hiring", "yes", "yes", "", "6/1/26", "2026-12-01", "yes", "none", "n/a", "n/a", "2026-12-20"}], # planted: C blank, a date not YYYY-MM-DD
                [{"uc-309", " Flag attendance problems ", "Discipline", " Yes ", "no", " YES", "2026-09-01", "2026-12-15", "no", "none", "n/a", "n/a", "2026-12-20"}], # planted: other case and spaces; still read
                [{"UC-310", "Pick finalists", "selection", "n/a", "n/a", "n/a", "2026-09-01", "", "n/a", "none", "n/a", "n/a", "2027-04-01"}], # planted: a decision type in no list, a record dated after the as-of date
                list(reg[1]),
# planted: duplicate id
                [{"", "Chat helper for recruiters", "", "", "", "", "", "", "", "", "", "", "2026-12-20"}], # planted: no id, no answers
                [{"UC-311", "Auto-assign overtime", "work assignment", "no", "no", "no", "2026-11-15", "", "no", "none", "n/a", "n/a", "2026-12-20"}], # planted: no notice, no opt-out, no exception
                [{"UC-312", "Rank internal candidates", "promotion", "yes", "no", "yes", "2027-01-20", "2027-02-10", "no", "hiring assessment", "yes", "yes", "2027-02-10"}], # planted: late notice, exception for another decision type
                [{"UC-313", "Screen resumes, second stage", "hiring", "yes", "no", "yes", "2026-10-01", "2026-12-01", "no", "human appeal", "no", "no", "2026-12-20"}], # planted: conditions and notice wording not recorded
                [{"UC-314", "Draft shift schedules", "work assignment", "no", "yes", "yes", "2027-04-15", "", "yes", "none", "n/a", "n/a", "2027-02-20"}]] # planted: a planned use, first used after the as-of date: not yet due, not an input fault
    with open(gen_dir / "register.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(header)
        w.writerows(reg)
    uses = ["UC-302", "UC-306", "UC-307", "UC-301", "UC-304"]
    requests = []
    for i in range(1, 25):
        kind = ["opt-out", "access", "appeal"][i % 3 - 1] if i % 3 else "appeal"
        received = day("2026-10-20") if i == 24 else day("2026-02-01") +
datetime.timedelta(days=int(i * 8.5))
        if i == 24:
            kind = "access"
        confirmed = received + datetime.timedelta(days=rnd.randint(1, 7)) if kind != "opt-out" else
None
        answered = received + datetime.timedelta(days=rnd.randint(3, 12) if kind == "opt-out" else
rnd.randint(10, 40))
        requests.append([f"R-{i:03d}", uses[i % 5], kind, iso(received), iso(confirmed if confirmed
and confirmed <= as_of else None),

```

```

        iso(answered if answered <= as_of else None), "", iso(received)])
    if plant:
        def at(n, col, value):
            requests[n - 1][col] = value

        def rec(n):
            return day(requests[n - 1][3])
        at(4, 5, "") # planted: R-004 (opt-
out) never processed
        at(6, 5, iso(rec(6) + datetime.timedelta(days=60))) # planted: R-006 (appeal)
        answered after 60 days, no extension
        at(8, 5, iso(rec(8) - datetime.timedelta(days=2))) # planted: R-008 answered
        before it was received
        at(10, 2, "complaint") # planted: R-010 a kind
    in no list
        at(11, 6, iso(rec(11) + datetime.timedelta(days=50))) # planted: R-011
    extension notice on day 50, answered on day 70
        at(11, 5, iso(rec(11) + datetime.timedelta(days=70)))
        at(14, 6, iso(rec(14) + datetime.timedelta(days=30))) # planted: R-014 a valid
    extension, answered on day 80 (no flag)
        at(14, 5, iso(rec(14) + datetime.timedelta(days=80)))
        at(17, 6, iso(rec(17) + datetime.timedelta(days=30))) # planted: R-017
    extension notice on day 30, answered on day 95
        at(17, 5, iso(rec(17) + datetime.timedelta(days=95)))
        at(19, 5, iso(rec(19) + datetime.timedelta(days=35))) # planted: R-019 (opt-
out) processed after 35 days
        at(20, 4, "") # planted: R-020 (access)
    no confirmation, answered on day 30
        at(20, 5, iso(rec(20) + datetime.timedelta(days=30)))
        at(22, 1, "UC-999") # planted: R-022 names a
    use that is not on the register
        requests.append(list(requests[11])) # planted: a duplicated
    request id
        requests.append(["", "UC-301", "access", "2026-05-01", "2026-05-03", "2026-05-20", "",
"2026-05-01"]) # planted: a request with no id
        with open(gen_dir / "requests.csv", "w", newline="") as f:
            w = csv.writer(f)
            w.writerow(["request_id", "use_case_id", "kind", "received_on", "confirmed_on",
"answered_on", "extension_notice_on", "created_on"])
            w.writerows(requests)
        return str(gen_dir / "register.csv"), str(gen_dir / "requests.csv")

def fingerprint(rows):
    body = "\n".join("|".join(" " if v is None else str(v) for v in r) for r in rows)
    return hashlib.sha256(body.encode()).hexdigest()

def population(out_dir, name, control, filters, rows, header, complete):
    out_dir.mkdir(parents=True, exist_ok=True)
    with open(out_dir / f"{name}.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(header)
        w.writerows(rows)
    print(f"\n== {control}: {name}" + (" " if complete else " ** INCOMPLETE: see input checks
**"))
    print(f"filters : {filters}")
    print(f"rows : {len(rows)}")

```

```

print(f"sha256      : {fingerprint(rows)}")

def flag(argv, name):
    """The value after a --flag, or None; stops with a message if it is missing."""
    if name not in argv:
        return None
    at = argv.index(name) + 1
    if at >= len(argv) or argv[at].startswith("--"):
        sys.exit(f"usage: {name} needs a value after it")
    return argv[at]

def check_source():
    """Fetch the regulation PDF and compare it with the pins. Returns True when both match."""
    req = urllib.request.Request(REG_URL, headers={"User-Agent": USER_AGENT})
    try:
        with urllib.request.urlopen(req, timeout=60) as r:
            body, modified = r.read(), r.headers.get("Last-Modified", "not sent")
    except OSError as e:
        print(f"source      : COULD NOT FETCH {REG_URL}: {e}")
        return False
    sha = hashlib.sha256(body).hexdigest()
    same = len(body) == REG_BYTES and sha == REG_SHA256
    print(f"source      : {REG_URL}")
    print(f"          {len(body)} bytes, Last-Modified: {modified}, sha256 {sha}")
    print(f"          pinned: {REG_BYTES} bytes, sha256 {REG_SHA256}: " + ("MATCH" if same else
"DIFFERENT, THE SOURCE CHANGED: re-read it before trusting the dates and limits here"))
    return same

def main():
    argv = sys.argv
    strict = "--strict" in argv
    mode = "plant" if "--plant" in argv else "clean"
    base = pathlib.Path(flag(argv, "--work-dir") or ".")
    out_dir, gen_dir = base / "evidence-out", base / "generated"
    as_of_text = flag(argv, "--as-of") or AS_OF
    as_of = read_day(as_of_text) if as_of_text else datetime.date.today()
    if as_of is None:
        sys.exit(f"the as-of date {as_of_text!r} is not written as YYYY-MM-DD, for example 2026-10-
31")
    for name, words in [("DECISION_WORDS", DECISION_WORDS), ("YESNO_WORDS", YESNO_WORDS),
("EXCEPTION_WORDS", EXCEPTION_WORDS),
("KIND_WORDS", KIND_WORDS)]:
        for k in words:
            if k != k.strip().lower():
                sys.exit(f"{name}: the answer {k!r} must be lower case with no spaces at the ends")
    for v in DECISION_WORDS.values():
        if v != "neither" and v not in EMPLOYMENT:
            sys.exit(f"DECISION_WORDS: {v!r} must be neither, hiring, promotion, work or pay, or
demotion, suspension or termination")
    for v in YESNO_WORDS.values():
        if v not in ("yes", "no", "n/a"):
            sys.exit(f"YESNO_WORDS: {v!r} must be yes, no or n/a")
    for v in EXCEPTION_WORDS.values():
        if v not in ("none", "appeal", "hiring", "work or pay"):
            sys.exit(f"EXCEPTION_WORDS: {v!r} must be none, appeal, hiring or work or pay")

```

```

for v in KIND_WORDS.values():
    if v not in ("opt-out", "access", "appeal"):
        sys.exit(f"KIND_WORDS: {v!r} must be opt-out, access or appeal")
files = [REGISTER_FILE, REQUESTS_FILE]
if all(f is None for f in files):
    reg_file, req_file = generate(mode, gen_dir, as_of)
    print(f"input      : GENERATED records (not a real organization), seed 227, mode={mode}")
elif any(f is None for f in files):
    sys.exit("set REGISTER_FILE and REQUESTS_FILE together")
else:
    reg_file, req_file = files
    print("input      : " + ", ".join(files))
for label, url in SOURCES:
    print(f"source      : {label}, {url}")
source_ok = check_source() if "--check-source" in argv else None

holidays = {read_day(h) for h in BUSINESS_HOLIDAYS}
bad_holidays = sum(1 for h in holidays if h is None)
holidays.discard(None)
print(f"as of      : {as_of}    business days: Monday to Friday except {len(holidays)} listed
holiday(s)")

con = duckdb.connect()
# every column your settings name must exist in your files; a wrong name stops the run with the
names it did find
for label, path, cols in [("register", reg_file, REG_COLS), ("request log", req_file,
REQ_COLS)]:
    have = {r[0].lower() for r in con.execute("DESCRIBE SELECT * FROM read_csv(?, header=true,
all_varchar=true)", [path]).fetchall()}
    lost = [c for c in cols.values() if c.lower() not in have]
    if lost:
        sys.exit(f"{label}: no column named {' '.join(lost)} in {path}; its columns are {' '.join(sorted(have))}")
    word = lambda c: f"lower(trim(coalesce({qcol(c)}, '')))"
    ident = lambda c: f"nullif(lower(trim(coalesce({qcol(c)}, ''))), '')"
    shown = lambda c: f"nullif(trim(coalesce({qcol(c)}, '')), '')"

    def table(name, path, cols, ids, words, dates):
        parts = [f"row_number() OVER () AS row_no", f"{ident(cols[ids[0]])} AS {ids[0]}", f"
{shown(cols[ids[0]])} AS id_shown"]
        parts += [f"{word(cols[w])} AS {w}_word" for w in words]
        for n in dates:
            parsed, given = strict_date(cols[n])
            parts.append(f"{parsed} AS {n}, {given} AS {n}_given")
        con.execute(f"CREATE TABLE {name} AS SELECT {' '.join(parts)} FROM read_csv(?,
header=true, all_varchar=true)", [path])

    reg_words = ["decision", "human_a", "human_b", "human_c", "opt_out_link", "exception",
"exception_conditions", "notice_covers"]
    table("register", reg_file, REG_COLS, ["use_case_id"], reg_words, ["first_used_on",
"notice_on", "created_on"])
    table("requests", req_file, REQ_COLS, ["request_id"], ["kind"], ["received_on",
"confirmed_on", "answered_on", "extension_notice_on", "created_on"])
    con.execute(f"CREATE TABLE req_use AS SELECT row_number() OVER () AS row_no,
{ident(REQ_COLS['use_case_id'])} AS use_case_id FROM read_csv(?, header=true, all_varchar=true)",
[req_file])
    q = lambda s, *p: con.execute(s, list(p)).fetchone()[0]
    n_reg, n_req = q("SELECT count(*) FROM register"), q("SELECT count(*) FROM requests")

```

```

# ---- input checks: each says which planted fault trips it
dup = lambda tbl, col: q(f"SELECT count(*) FROM (SELECT {col} FROM {tbl} WHERE {col} IS NOT
NULL GROUP BY 1 HAVING count(*) > 1)")
def not_in(tbl, col, words):
    con.execute("CREATE OR REPLACE TEMP TABLE w(word VARCHAR)")
    con.executemany("INSERT INTO w VALUES (?)", [(k,) for k in words])
    return q(f"SELECT count(*) FROM {tbl} WHERE {col} NOT IN (SELECT word FROM w)")
unreadable = lambda tbl, c: f"({c}_given AND {c} IS NULL)"
reg_dates = ["first_used_on", "notice_on", "created_on"]
req_dates = ["received_on", "confirmed_on", "answered_on", "extension_notice_on", "created_on"]
unread_reg = sum(q(f"SELECT count(*) FROM register WHERE {unreadable('register', c)}") for c in
reg_dates)
unread_req = (q("SELECT count(*) FROM requests WHERE NOT received_on_given OR NOT
created_on_given") +
sum(q(f"SELECT count(*) FROM requests WHERE {unreadable('requests', c)}") for c
in req_dates))
# first_used_on is left out: a use that starts after the as-of date is a planned use, reviewed
as not yet due
after = sum(q(f"SELECT count(*) FROM register WHERE {c} > ?::DATE", str(as_of)) for c in
["notice_on", "created_on"]) + \
sum(q(f"SELECT count(*) FROM requests WHERE {c} > ?::DATE", str(as_of)) for c in req_dates)
yn_cols = ["human_a_word", "human_b_word", "human_c_word", "opt_out_link_word",
"exception_conditions_word", "notice_covers_word"]
checks = [
("holidays not written as YYYY-MM-DD", bad_holidays, "a BUSINESS_HOLIDAYS entry such as
25/12/2026"),
("register ids listed more than once (ignoring case)", dup("register", "use_case_id"), "a
duplicated register row"),
("register rows with no use case id", q("SELECT count(*) FROM register WHERE use_case_id IS
NULL"), "a register row with no id"),
("register decision types in no list", not_in("register", "decision_word", DECISION_WORDS),
"a decision type such as 'selection' (an empty answer counts)"),
("register yes/no answers in no list (human involvement A, B, C, opt-out link, exception
conditions, notice covers)",
sum(not_in("register", c, YESNO_WORDS) for c in yn_cols), "an answer left empty, such as
human involvement C"),
("register exception answers in no list", not_in("register", "exception_word",
EXCEPTION_WORDS), "an exception answer left empty or spelled another way"),
("request ids listed more than once (ignoring case)", dup("requests", "request_id"), "a
duplicated request row"),
("request rows with no request id", q("SELECT count(*) FROM requests WHERE request_id IS
NULL"), "a request row with no id"),
("request kinds in no list", not_in("requests", "kind_word", KIND_WORDS), "a kind such as
'complaint'"),
("requests naming a use that is not on the register",
q("SELECT count(*) FROM req_use WHERE use_case_id IS NULL OR use_case_id NOT IN (SELECT
use_case_id FROM register WHERE use_case_id IS NOT NULL)"),
"a request that names UC-999, or names no use"),
("register dates not written as YYYY-MM-DD (a blank first-use or notice date is reviewed
per use, not counted here)", unread_reg,
"a first-use date such as 6/1/26"),
("request dates not written as YYYY-MM-DD, or received or created date missing (a blank
confirmation, answer or extension date is a finding)",
unread_req, "a received date such as 6/12/26"),
("record dates after the as-of date (a first-use date after it is a planned use, reviewed
as not yet due)", after,
"a record dated 2027-04-01 when the as-of date is 2027-03-01"),

```

```

        ("answers, confirmations or extension notices dated before their request was received",
         q("SELECT count(*) FROM requests WHERE answered_on < received_on OR confirmed_on <
received_on OR extension_notice_on < received_on"),
         "a request answered two days before it was received"),
    ]
    print(f"\n== Input checks (register rows: {n_reg}; request rows: {n_req})")
    # the source check runs only with --check-source; when it runs and differs, or cannot fetch, it
    fails the run
    ok = source_ok is not False
    if source_ok is None:
        print("[info] regulation PDF not re-checked against its pinned size and SHA-256 (add --
check-source)")
    else:
        print(f"[{'ok' if source_ok else 'FAIL'}] regulation PDF matches its pinned size and SHA-
256" +
              (" if source_ok else " (trips on: a changed PDF, or no network)))
    for label, n, fault in checks:
        ok &= n == 0
        print(f"[{'ok' if n == 0 else 'FAIL'}] {label}: {n}" + (" if n == 0 else f" (trips on:
{fault})))
    # a word list entry that matches nothing in your files is reported, never silently ignored
    used = lambda tbl, cols: {r[0] for c in cols for r in con.execute(f"SELECT DISTINCT {c} FROM
{tbl}")}.fetchall()
    for src, words, seen in [("decision", DECISION_WORDS, used("register", ["decision_word"])),
                             ("yes/no", YESNO_WORDS, used("register", yn_cols)),
                             ("exception", EXCEPTION_WORDS, used("register", ["exception_word"])),
                             ("request kind", KIND_WORDS, used("requests", ["kind_word"]))]:
        dead = sorted(k for k in words if k not in seen)
        if dead:
            print(f"[info] {src} answers in your word list that no row uses: " + "; ".join(dead))

    # ---- 1. the use sort: the three elements of human involvement, as recorded
    reg_rows = con.execute("""SELECT row_no, coalesce(id_shown, '-') , decision_word, human_a_word,
human_b_word, human_c_word, opt_out_link_word,
                             exception_word, exception_conditions_word, notice_covers_word,
first_used_on, first_used_on_given,
                             notice_on, notice_on_given FROM register ORDER BY
row_no""").fetchall()
    sorted_uses, r_uc = {}, []
    for row_no, uid, dec, a, b, c, opt, exc, cond, cover, first, first_given, notice, notice_given
in reg_rows:
        decision = DECISION_WORDS.get(dec)
        if decision is None:
            status, reason = "FOR LEGAL TO CHECK", "decision type not recorded or not recognized"
        elif decision == "neither":
            status, reason = "NOT AN EMPLOYMENT USE AS RECORDED", ""
        else:
            answers = {"A": YESNO_WORDS.get(a), "B": YESNO_WORDS.get(b), "C": YESNO_WORDS.get(c)}
            missing = [k for k, v in answers.items() if v not in ("yes", "no")]
            no = [k for k, v in answers.items() if v == "no"]
            if not missing and not no:
                status, reason = "HUMAN INVOLVEMENT RECORDED (A, B, C)", ""
            else:
                parts = ([f"{k} not recorded" for k in missing] + [f"{k} recorded as no" for k in
no])
                status, reason = "FOR LEGAL TO CHECK", "may substantially replace human
decisionmaking: " + ", ".join(sorted(parts))
            sorted_uses[row_no] = (uid, status, decision)

```

```

        r_uc.append((row_no, uid, status, reason))
    population(out_dir, "use_sort", "CCPA ADMT rules: the use sort",
        "every register row, none dropped; the three elements of human involvement (7001(e)
(1)(A), (B), (C)) as recorded; "
        "an empty or unrecognized answer is FOR LEGAL TO CHECK, never involvement recorded",
    r_uc, ["row_no", "use_case_id", "status", "reason"], ok)
    for r in r_uc:
        print(" " + " | ".join(str(v) for v in r))
    counts = {k: sum(1 for r in r_uc if r[2] == k) for k in sorted({r[2] for r in r_uc})}
    print(" " + "; ".join(f"{k}: {v}" for k, v in counts.items()))
    print(" This sorts uses by what the register says. It does not say a use is ADMT or is not:
Legal decides.")

# ---- 2. notice and opt-out, for each employment use marked FOR LEGAL TO CHECK
r_not, follow_not, open_not = [], 0, 0
for row_no, uid, dec, a, b, c, opt, exc, cond, cover, first, first_given, notice, notice_given
in reg_rows:
    _, use_status, decision = sorted_uses[row_no]
    if use_status != "FOR LEGAL TO CHECK" or decision is None:
        why = {"NOT AN EMPLOYMENT USE AS RECORDED": "not an employment use as recorded",
            "HUMAN INVOLVEMENT RECORDED (A, B, C)": "human involvement
recorded"}.get(use_status, "decision type not recorded or not recognized")
        r_not.append((row_no, uid, "NOT REVIEWED", why))
        continue
    hard, soft = [], []
    if first is None:
        hard.append("first-use date not read" if first_given else "first-use date missing")
    due = max(first, DUTY_FROM) if first else DUTY_FROM
    if not notice_given:
        (hard if as_of > due else soft).append(f"NO PRE-USE NOTICE RECORDED (due {due})" if
as_of > due else f"notice due {due}")
    elif notice is None:
        hard.append("notice date not read")
    elif notice > due:
        hard.append(f"NOTICE DATED {(notice - due).days} DAYS AFTER ITS DUE DATE (due {due})")
    meaning, optw, condw, coverw = EXCEPTION_WORDS.get(exc), YESNO_WORDS.get(opt),
YESNO_WORDS.get(cond), YESNO_WORDS.get(cover)
    if meaning is None:
        hard.append("exception answer not recognized")
    elif meaning == "none":
        if optw == "no":
            hard.append("NO OPT-OUT METHOD AND NO EXCEPTION NAMED")
        elif optw != "yes":
            hard.append("opt-out method answer not recorded or not recognized")
    else:
        section = {"appeal": "7221(b)(1)", "hiring": "7221(b)(2)", "work or pay": "7221(b)(3)"}
[meaning]
        if meaning == "hiring" and decision != "hiring" or meaning == "work or pay" and
decision != "work or pay":
            hard.append(f"EXCEPTION NAMED ({exc}, {section}) DOES NOT MATCH THE DECISION TYPE
({dec})")
        if condw != "yes":
            hard.append(f"EXCEPTION NAMED ({section}), ITS CONDITIONS NOT RECORDED AS MET")
        if coverw != "yes":
            hard.append("NOTICE DOES NOT SAY HOW TO APPEAL" if meaning == "appeal" else "NOTICE
DOES NOT NAME THE EXCEPTION")
        status = "FOR FOLLOW-UP" if hard else ("OPEN, NOT YET DUE" if soft else "NO FLAG")
        follow_not += status == "FOR FOLLOW-UP"

```

```

open_not += status.startswith("OPEN")
r_not.append((row_no, uid, status, "; ".join(hard + soft)))
population(out_dir, "notice_optout_review", "CCPA ADMT rules: Pre-use Notice and opt-out",
    f"each FOR LEGAL TO CHECK employment use; notice due the later of its first-use date
and {DUTY_FROM} (7200(b), 7220(b)(2)); "
    "an opt-out method or the one 7221(b) exception named, matched to the decision type,
with conditions and notice wording recorded; "
    "other uses NOT REVIEWED", r_not, ["row_no", "use_case_id", "status", "reason"], ok)
for r in r_not:
    if r[2] not in ("NO FLAG", "NOT REVIEWED"):
        print(" " + " | ".join(str(v) for v in r))
    ncounts = {k: sum(1 for r in r_not if r[2] == k) for k in sorted({r[2] for r in r_not})}
    print(" " + "; ".join(f"{k}: {v}" for k, v in ncounts.items()))
    print(" A flag says what the register records. It does not say the rules were or were not
followed.")

# ---- 3. requests: the deadlines the regulation states
rows = con.execute("""SELECT row_no, coalesce(id_shown, '-'), kind_word, received_on,
confirmed_on_given, confirmed_on, answered_on_given, answered_on,
                                extension_notice_on_given, extension_notice_on FROM requests ORDER
BY row_no""").fetchall()
r_req, follow_req, open_req, extended = [], 0, 0, 0
for row_no, rid, kind_word, received, c_given, confirmed, a_given, answered, e_given, ext in
rows:
    kind = KIND_WORDS.get(kind_word)
    hard, soft = [], []
    if kind is None:
        hard.append("kind not recorded or not recognized")
    if received is None:
        hard.append("received date not read")
    elif kind == "opt-out":
        due = add_business_days(received, OPT_OUT_BUSINESS_DAYS, holidays)
        if not a_given:
            (hard if as_of > due else soft).append(f"NOT PROCESSED, 15 BUSINESS DAYS PASSED
(due {due})" if as_of > due else f"open, due {due}")
        elif answered is None:
            hard.append("processed date not read")
        elif answered > due:
            hard.append(f"PROCESSED {(answered - received).days} DAYS AFTER RECEIPT (15
business days: due {due})")
    elif kind is not None:
        conf_due = add_business_days(received, CONFIRM_BUSINESS_DAYS, holidays)
        limit, limit_days = received + datetime.timedelta(days=RESPOND_DAYS), RESPOND_DAYS
        if e_given and ext is None:
            hard.append("extension notice date not read")
        elif e_given:
            if ext > received + datetime.timedelta(days=RESPOND_DAYS):
                hard.append(f"EXTENSION NOTICE DATED DAY {(ext - received).days}, AFTER DAY
{RESPOND_DAYS}")
        else:
            limit_days = RESPOND_DAYS_EXTENDED
            limit = received + datetime.timedelta(days=limit_days)
            extended += 1
        # 7021(a): receipt must be confirmed within 10 business days even when the request is
already answered; only the
        # description of the verification process and timing is not needed then
        if not c_given:
            if as_of > conf_due:

```

```

        hard.append(f"NO CONFIRMATION, 10 BUSINESS DAYS PASSED (due {conf_due})")
    else:
        soft.append(f"confirmation due {conf_due}")
    elif confirmed is None:
        hard.append("confirmation date not read")
    elif confirmed > conf_due:
        hard.append(f"CONFIRMED {(confirmed - received).days} DAYS AFTER RECEIPT (10
business days: due {conf_due})")
    if not a_given:
        (hard if as_of > limit else soft).append(f"NOT ANSWERED, DEADLINE PASSED (due
{limit})" if as_of > limit else f"open, due {limit}")
    elif answered is None:
        hard.append("answer date not read")
    elif answered > limit:
        hard.append(f"ANSWERED {(answered - received).days} DAYS AFTER RECEIPT (limit
{limit_days})")
    status = "FOR FOLLOW-UP" if hard else ("OPEN, INSIDE DEADLINE" if soft else "NO FLAG")
    follow_req += status == "FOR FOLLOW-UP"
    open_req += status.startswith("OPEN")
    r_req.append((row_no, rid, status, "; ".join(hard + soft)))
    population(out_dir, "request_review", "CCPA ADMT rules: requests answered",
        "every request row; opt-out processed within 15 business days of receipt (7221(n)
(1)); access and appeal confirmed within 10 business days "
        "(even when already answered), answered within 45 calendar days, or 90 with an
extension notice dated by day 45 (7021(a), (b)); "
        "a blank answer date is a finding", r_req, ["row_no", "request_id", "status",
"reason"], ok)
    for r in r_req:
        if r[2] != "NO FLAG":
            print(" " + " | ".join(str(v) for v in r))
    rcounts = {k: sum(1 for r in r_req if r[2] == k) for k in sorted({r[2] for r in r_req})}
    print(" " + "; ".join(f"{k}: {v}" for k, v in rcounts.items()))
    print(f" requests inside a valid 90-day extension: {extended}")

    uses_legal = sum(1 for r in r_uc if r[2] == "FOR LEGAL TO CHECK")
    print("\n== Result")
    print(f"input checks : {'all passed' if ok else 'FAILED, the lists above are not complete'}")
    print(f"for Legal      : {uses_legal} use(s) marked FOR LEGAL TO CHECK")
    print(f"notices         : {follow_not} use(s) for follow-up, {open_not} not yet due")
    print(f"requests         : {follow_req} for follow-up, {open_req} open inside the deadline")
    print("This is not legal advice and not a compliance test. Legal decides what the rules
require.")
    # --strict: FOR LEGAL TO CHECK uses are a prompt for Legal, not a failure, so they never set
the exit code
    return 1 if strict and (not ok or follow_not or follow_req) else 0

if __name__ == "__main__":
    sys.exit(main())

```

The runs

Run 1: clean generated records

Seed 227, seven invented register rows and 24 invented requests, with the as-of date 2026-10-31 and --check-source. The regulation PDF matches its pins, every input check passes, three of the seven uses are marked for Legal, no use or request has a flag and one request is still open inside its deadline.

```
input      : GENERATED records (not a real organization), seed 227, mode=clean
source     : CPPA approved regulation text,
https://cppa.ca.gov/regulations/pdf/ccpa_updates_cyber_risk_admt_appr_text.pdf
source     : CPPA rulemaking page, https://cppa.ca.gov/regulations/ccpa_updates.html
source     : https://cppa.ca.gov/regulations/pdf/ccpa_updates_cyber_risk_admt_appr_text.pdf
             666571 bytes, Last-Modified: Tue, 30 Sep 2025 15:40:50 GMT, sha256
7a34306cebf12ae9050490568b1d7ed532cfd38dc6ed8c7c3dc40afb23328650
             pinned: 666571 bytes, sha256
7a34306cebf12ae9050490568b1d7ed532cfd38dc6ed8c7c3dc40afb23328650: MATCH
as of      : 2026-10-31  business days: Monday to Friday except 0 listed holiday(s)

== Input checks  (register rows: 7; request rows: 24)
[ok ] regulation PDF matches its pinned size and SHA-256
[ok ] holidays not written as YYYY-MM-DD: 0
[ok ] register ids listed more than once (ignoring case): 0
[ok ] register rows with no use case id: 0
[ok ] register decision types in no list: 0
[ok ] register yes/no answers in no list (human involvement A, B, C, opt-out link, exception
conditions, notice covers): 0
[ok ] register exception answers in no list: 0
[ok ] request ids listed more than once (ignoring case): 0
[ok ] request rows with no request id: 0
[ok ] request kinds in no list: 0
[ok ] requests naming a use that is not on the register: 0
[ok ] register dates not written as YYYY-MM-DD (a blank first-use or notice date is reviewed per
use, not counted here): 0
[ok ] request dates not written as YYYY-MM-DD, or received or created date missing (a blank
confirmation, answer or extension date is a finding): 0
[ok ] record dates after the as-of date (a first-use date after it is a planned use, reviewed as
not yet due): 0
[ok ] answers, confirmations or extension notices dated before their request was received: 0
[info] decision answers in your word list that no row uses: demotion; discipline; suspension;
termination
[info] exception answers in your word list that no row uses: human appeal

== CCPA ADMT rules: the use sort: use_sort
filters    : every register row, none dropped; the three elements of human involvement (7001(e)(1)
(A), (B), (C)) as recorded; an empty or unrecognized answer is FOR LEGAL TO CHECK, never
involvement recorded
rows       : 7
sha256     : 8b885aa239138ac758b0e7076865b34f20cd8ef306b544725379cfb01c30628c
  1 | UC-301 | HUMAN INVOLVEMENT RECORDED (A, B, C) |
  2 | UC-302 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
  3 | UC-303 | NOT AN EMPLOYMENT USE AS RECORDED |
  4 | UC-304 | HUMAN INVOLVEMENT RECORDED (A, B, C) |
```

```

5 | UC-305 | NOT AN EMPLOYMENT USE AS RECORDED |
6 | UC-306 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: C recorded as
no
7 | UC-307 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
FOR LEGAL TO CHECK: 3; HUMAN INVOLVEMENT RECORDED (A, B, C): 2; NOT AN EMPLOYMENT USE AS
RECORDED: 2
This sorts uses by what the register says. It does not say a use is ADMT or is not: Legal
decides.

== CCPA ADMT rules: Pre-use Notice and opt-out: notice_optout_review
filters : each FOR LEGAL TO CHECK employment use; notice due the later of its first-use date and
2027-01-01 (7200(b), 7220(b)(2)); an opt-out method or the one 7221(b) exception named, matched to
the decision type, with conditions and notice wording recorded; other uses NOT REVIEWED
rows : 7
sha256 : f50cd4b9a81225743bcd77b28be0ab5c3fb665f3fa4558aca57ca82b9afa4924
NO FLAG: 3; NOT REVIEWED: 4
A flag says what the register records. It does not say the rules were or were not followed.

== CCPA ADMT rules: requests answered: request_review
filters : every request row; opt-out processed within 15 business days of receipt (7221(n)(1));
access and appeal confirmed within 10 business days (even when already answered), answered within
45 calendar days, or 90 with an extension notice dated by day 45 (7021(a), (b)); a blank answer
date is a finding
rows : 24
sha256 : 1a147ef77237276ecc62e94caab2e8366103b733a1b24aaedcda619fa7d93549
24 | R-024 | OPEN, INSIDE DEADLINE | open, due 2026-12-04
NO FLAG: 23; OPEN, INSIDE DEADLINE: 1
requests inside a valid 90-day extension: 0

== Result
input checks : all passed
for Legal : 3 use(s) marked FOR LEGAL TO CHECK
notices : 0 use(s) for follow-up, 0 not yet due
requests : 0 for follow-up, 1 open inside the deadline
This is not legal advice and not a compliance test. Legal decides what the rules require.

```

Run 2: planted faults

The same script with `--plant` and the as-of date 2027-03-01, after January 1, 2027, the date § 7200(b) sets for compliance with the Article, so the script marks a missing notice overdue. It plants a fault for each input check that reads the files, except the one on the format of request dates: a duplicated and a missing id in each file, decision types, answers and exceptions in no list, a request kind in no list, a request that names a use not on the register, a first-use date written 6/1/26, a record dated after the as-of date and a request answered before it was received. It also plants uses with no notice, with no opt-out method and no exception, with a late notice, with an exception for another decision type and with conditions and notice wording not recorded, and a planned use that starts after the as-of date; and requests that were never processed, answered late, confirmed never, or given an extension notice too late or too long. Every list is marked INCOMPLETE.

```

input : GENERATED records (not a real organization), seed 227, mode=plant
source : CPPA approved regulation text,
https://cppa.ca.gov/regulations/pdf/ccpa_updates_cyber_risk_admt_appr_text.pdf

```

```
source      : CCPA rulemaking page, https://coppa.ca.gov/regulations/ccpa_updates.html
as of       : 2027-03-01   business days: Monday to Friday except 0 listed holiday(s)
```

```
== Input checks   (register rows: 16; request rows: 26)
```

```
[info] regulation PDF not re-checked against its pinned size and SHA-256 (add --check-source)
[ok ] holidays not written as YYYY-MM-DD: 0
[FAIL] register ids listed more than once (ignoring case): 1   (trips on: a duplicated register
row)
[FAIL] register rows with no use case id: 1   (trips on: a register row with no id)
[FAIL] register decision types in no list: 2   (trips on: a decision type such as 'selection' (an
empty answer counts))
[FAIL] register yes/no answers in no list (human involvement A, B, C, opt-out link, exception
conditions, notice covers): 7   (trips on: an answer left empty, such as human involvement C)
[FAIL] register exception answers in no list: 1   (trips on: an exception answer left empty or
spelled another way)
[FAIL] request ids listed more than once (ignoring case): 1   (trips on: a duplicated request row)
[FAIL] request rows with no request id: 1   (trips on: a request row with no id)
[FAIL] request kinds in no list: 1   (trips on: a kind such as 'complaint')
[FAIL] requests naming a use that is not on the register: 1   (trips on: a request that names UC-
999, or names no use)
[FAIL] register dates not written as YYYY-MM-DD (a blank first-use or notice date is reviewed per
use, not counted here): 1   (trips on: a first-use date such as 6/1/26)
[ok ] request dates not written as YYYY-MM-DD, or received or created date missing (a blank
confirmation, answer or extension date is a finding): 0
[FAIL] record dates after the as-of date (a first-use date after it is a planned use, reviewed as
not yet due): 1   (trips on: a record dated 2027-04-01 when the as-of date is 2027-03-01)
[FAIL] answers, confirmations or extension notices dated before their request was received: 1
(trips on: a request answered two days before it was received)
[info] decision answers in your word list that no row uses: demotion; suspension; termination
```

```
== CCPA ADMT rules: the use sort: use_sort   ** INCOMPLETE: see input checks **
```

```
filters      : every register row, none dropped; the three elements of human involvement (7001(e)(1)
(A), (B), (C)) as recorded; an empty or unrecognized answer is FOR LEGAL TO CHECK, never
involvement recorded
```

```
rows         : 16
```

```
sha256       : 3b931c9a5d96dfab60273cd9a7cb26ba78c86949b2bf602559bd3b96dafcebdb
```

```
  1 | UC-301 | HUMAN INVOLVEMENT RECORDED (A, B, C) |
  2 | UC-302 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
  3 | UC-303 | NOT AN EMPLOYMENT USE AS RECORDED |
  4 | UC-304 | HUMAN INVOLVEMENT RECORDED (A, B, C) |
  5 | UC-305 | NOT AN EMPLOYMENT USE AS RECORDED |
  6 | UC-306 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: C recorded as
no
  7 | UC-307 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
  8 | UC-308 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: C not recorded
  9 | UC-309 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
 10 | UC-310 | FOR LEGAL TO CHECK | decision type not recorded or not recognized
 11 | UC-302 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
 12 | -      | FOR LEGAL TO CHECK | decision type not recorded or not recognized
 13 | UC-311 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: A recorded as
no, B recorded as no, C recorded as no
 14 | UC-312 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
no
 15 | UC-313 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: B recorded as
```

```

no
16 | UC-314 | FOR LEGAL TO CHECK | may substantially replace human decisionmaking: A recorded as
no
FOR LEGAL TO CHECK: 12; HUMAN INVOLVEMENT RECORDED (A, B, C): 2; NOT AN EMPLOYMENT USE AS
RECORDED: 2
This sorts uses by what the register says. It does not say a use is ADMT or is not: Legal
decides.

== CCPA ADMT rules: Pre-use Notice and opt-out: notice_optout_review ** INCOMPLETE: see input
checks **
filters : each FOR LEGAL TO CHECK employment use; notice due the later of its first-use date and
2027-01-01 (7200(b), 7220(b)(2)); an opt-out method or the one 7221(b) exception named, matched to
the decision type, with conditions and notice wording recorded; other uses NOT REVIEWED
rows : 16
sha256 : 894f1232fbd5248f0aacc5c103c344d35308108aec654ecdc3e1c2aac93f14b
8 | UC-308 | FOR FOLLOW-UP | first-use date not read
9 | uc-309 | FOR FOLLOW-UP | NO OPT-OUT METHOD AND NO EXCEPTION NAMED
13 | UC-311 | FOR FOLLOW-UP | NO PRE-USE NOTICE RECORDED (due 2027-01-01); NO OPT-OUT METHOD AND
NO EXCEPTION NAMED
14 | UC-312 | FOR FOLLOW-UP | NOTICE DATED 21 DAYS AFTER ITS DUE DATE (due 2027-01-20); EXCEPTION
NAMED (hiring assessment, 7221(b)(2)) DOES NOT MATCH THE DECISION TYPE (promotion)
15 | UC-313 | FOR FOLLOW-UP | EXCEPTION NAMED (7221(b)(1)), ITS CONDITIONS NOT RECORDED AS MET;
NOTICE DOES NOT SAY HOW TO APPEAL
16 | UC-314 | OPEN, NOT YET DUE | notice due 2027-04-15
FOR FOLLOW-UP: 5; NO FLAG: 4; NOT REVIEWED: 6; OPEN, NOT YET DUE: 1
A flag says what the register records. It does not say the rules were or were not followed.

== CCPA ADMT rules: requests answered: request_review ** INCOMPLETE: see input checks **
filters : every request row; opt-out processed within 15 business days of receipt (7221(n)(1));
access and appeal confirmed within 10 business days (even when already answered), answered within
45 calendar days, or 90 with an extension notice dated by day 45 (7021(a), (b)); a blank answer
date is a finding
rows : 26
sha256 : 4ad3e6768cadff973b68aab40db55110798b2322e8ce2a8ee86af9a287191be8
4 | R-004 | FOR FOLLOW-UP | NOT PROCESSED, 15 BUSINESS DAYS PASSED (due 2026-03-27)
6 | R-006 | FOR FOLLOW-UP | ANSWERED 60 DAYS AFTER RECEIPT (limit 45)
10 | R-010 | FOR FOLLOW-UP | kind not recorded or not recognized
11 | R-011 | FOR FOLLOW-UP | EXTENSION NOTICE DATED DAY 50, AFTER DAY 45; ANSWERED 70 DAYS AFTER
RECEIPT (limit 45)
17 | R-017 | FOR FOLLOW-UP | ANSWERED 95 DAYS AFTER RECEIPT (limit 90)
19 | R-019 | FOR FOLLOW-UP | PROCESSED 35 DAYS AFTER RECEIPT (15 business days: due 2026-07-31)
20 | R-020 | FOR FOLLOW-UP | NO CONFIRMATION, 10 BUSINESS DAYS PASSED (due 2026-08-04)
FOR FOLLOW-UP: 7; NO FLAG: 19
requests inside a valid 90-day extension: 2

== Result
input checks : FAILED, the lists above are not complete
for Legal : 12 use(s) marked FOR LEGAL TO CHECK
notices : 5 use(s) for follow-up, 1 not yet due
requests : 7 for follow-up, 0 open inside the deadline
This is not legal advice and not a compliance test. Legal decides what the rules require.

```

UC-311 has no notice and no opt-out method, so it is for follow-up. UC-312 has a notice dated 21 days after its due date and names the hiring exception for a promotion. UC-314 is a planned use that starts on 2027-04-15, after the as-of date, so its notice is shown as not yet due and no input check fails for it. R-014 is not flagged: its extension notice came on day 30 and it was answered on day 80,

inside 90. R-017 is flagged: the same extension, answered on day 95. R-011 is flagged twice, because its extension notice came on day 50 and it was answered on day 70. These findings come from invented records and say nothing about any real company.

The reader's own data

To run it on your own data, change only the settings between the lines marked CHANGE THESE FOR YOUR OWN DATA. Set REGISTER_FILE and REQUESTS_FILE to the paths of your two CSV files. In REG_COLS and REQ_COLS, put the name of the column in your files that holds each field. In DECISION_WORDS, YESNO_WORDS, EXCEPTION_WORDS and KIND_WORDS, put the answers your files use, in lower case, each with the meaning the script gives it. Set AS_OF to the day the check is for, written YYYY-MM-DD, or leave it as None to check as of today. If your counsel does not count some weekdays as business days, list them in BUSINESS_HOLIDAYS, each written YYYY-MM-DD. Dates in your files must be written YYYY-MM-DD. An answer that is in none of those lists is counted and fails an input check, so the run tells you which one to add.

We also tested the instructions above on a second set of generated files of our own. That test is not published with the guide, so you cannot re-run it from this page; what it showed is below. The files used other file names, other column names, other answer words, upper-case answers, a blank use id, a use id repeated in another case (`hr-5` beside `HR-5`), a decision type in no list, an empty answer for element C, two requests for uses not on the register, a planned use that starts after the as-of date, an extension notice dated before its request was received, a request answered in three days whose receipt was never confirmed, a day listed as a holiday and an opt-out never processed. It read `RECRUITMENT` and `Y` as a hiring use with all three elements recorded and named the element not recorded for `HR-5`. It failed the checks for the repeated id, the missing id, the decision type in no list, the empty answer, the two requests that name no register use and the extension notice dated before receipt. It showed the planned use `HR-10` as not yet due without failing a check, flagged a use with no notice and no opt-out method, and flagged four requests, including the one answered without a confirmation. The holiday moved an opt-out's due date from 2026-02-23 to 2026-02-24. We have not run it on a real company's records.

How to read the results

- **A status is a prompt for Legal, not a finding.** `FOR LEGAL TO CHECK` means the register records a missing or no answer for one of the three elements, or is silent or not recognized.
- **HUMAN INVOLVEMENT RECORDED is not a clearance.** It means you recorded yes three times. Whether a reviewer has the knowledge, the analysis and the authority is a fact the register cannot show.
- **FOR FOLLOW-UP means the register or log records something against a date or a requirement in the text.** It does not say the rules were breached: the record may be wrong, and the regulation is the authority.

- **OPEN is not a fault.** It means the due date has not yet passed on the as-of date.
- **An INCOMPLETE list is not a result.** Fix the failed input checks first.

Checklist

Before you rely on a list

Item	Done
Your register has one row for every use of Claude in HR, including drafts and summaries	☐
Each employment use has A, B and C recorded as yes or no, by the person who knows	☐
Every answer your files use is in the four answer lists, in lower case	☐
Dates in both files are written YYYY-MM-DD	☐
The as-of date is the day the check is for, and your counsel's non-business days are listed	☐
You ran with <code>--check-source</code> and the PDF matches its pins	☐
Every input check passes before you read a list	☐
Each <code>FOR LEGAL TO CHECK</code> use is with your counsel, with the approved text	☐
Each <code>FOR FOLLOW-UP</code> use or request has an owner and a date	☐
You kept the CSVs and the logs under your own access controls	☐

Limits

- **This is not a client result.** The records are generated. The runs show the script finds planted faults. They say nothing about any real company.
- **This is not legal advice,** and the script does not decide whether a use is ADMT, whether a worker is a consumer, or whether an exception applies. It sorts uses by what your register says. Legal decides.
- **We quote the text; we do not apply all of it.** The script does not test the carve-outs in § 7001(e)(3), the content a notice or an access response must hold (§§ 7220(c), 7222(b)), verification, or what happens to a request made before processing began (§ 7221(m)).
- **The exception conditions are your answers.** The script records whether you say the conditions in § 7221(b) are met. It cannot test "solely", "works for the business's purpose" or "does not unlawfully discriminate".

- **Our reading is not the agency's.** The notice due date, the business-day count and the mapping of your answers are ours. We have not asked the agency to review them.
- **The extension-notice day is ours.** The rule that an extension notice must be dated by day 45 is ours; § 7021(b) requires notice and an explanation and states no day. The script does not check the explanation.
- **A log can only show what it holds.** A request never entered in the log cannot be checked.
- **The regulation may change.** The text we read was approved on September 22, 2025. Check the agency's page for the current position.
- **The logs hold personal data.** Keep them under your own controls and out of the PDF, the CSVs and anything you share.

Sources

- CPPA, text of regulations (approved), https://cppa.ca.gov/regulations/pdf/ccpa_updates_cyber_risk_admt_appr_text.pdf (read 2026-10-31; 666,571 bytes; Last-Modified Tue, 30 Sep 2025 15:40:50 GMT)
- CPPA, CCPA updates, cyber, risk, ADMT and insurance regulations, https://cppa.ca.gov/regulations/ccpa_updates.html (read 2026-10-31)
- DuckDB, CSV overview, <https://duckdb.org/docs/2.0/data/csv/overview.html> (read 2026-10-31)
- Data Meaning, Control Tower for Claude, <https://www.datameaning.com/solutions/claude-control-tower.html> (read 2026-10-31)

Looking at your own Claude use?

Control Tower for Claude runs in your own Azure subscription. See the solution page.

[See Control Tower for Claude](https://www.datameaning.com/solutions/claude-control-tower.html)