What Do Monsoon Power Outages Cost Tucson? A Regression and Classification Approach

Jake Moore

2026-11-10

Contents

1. Introduction

2. Data Loading and Cleaning

- N/A Value Analysis
- Cleaning Steps
- Missing Intervals: Zero or Unknown?
- Sanity Check

3. Feature Engineering

4. Exploratory Data Analysis (EDA)

- When Do Outages Happen?
- The Worst Nights
- Does Humidity Set Up the Storm?
- Cross-Checking Against TEP's Own Numbers

5. What Does an Outage Cost?

- How Sure Is the Number?

6. Model 1: Multiple Linear Regression - What Drives Storm-Night Outage Severity?

- How It Works
- Training the Model
- Diagnostics

7. Model 2: Linear Regression - What Goes With Slower Restoration?

- How It Works
- Training the Model

8. Model 3: Logistic Regression - Can We Flag a Big Outage Night by 3 PM?

- How It Works
- Preparing the Data for Modeling
- Train-Test Split
- Choosing the Inputs
- Model Evaluation
- Does It Hold Up in Other Years?

9. Model 4: Decision Tree - An Interpretable Alternative
- How It Works
- Training the Tree
- Tree vs. Logistic Regression

10. Recommendations
- What Would Improvements Be Worth?
- Backtesting the 3 PM Flag

11. Conclusion & Reflection

12. References

1. Introduction

Summer storms are a recurring reliability problem in southern Arizona. A bad monsoon night can leave tens of thousands of people in Tucson without power, but an outage count on its own doesn't say what that costs anyone, what made the night so bad, or what a utility could do about it.

- This project asks: **What do power outages cost Pima County, what conditions go with a severe storm night, and can the worst nights be flagged by mid-afternoon?**
- **Why it matters**
- Tucson Electric Power (TEP), which serves most of Pima County, reported 168.7 outage minutes per customer in 2023, and 106.8 of those minutes came from major-event days.
- Utilities ask regulators for grid-hardening budgets (stronger poles, tree trimming, automated switches). Each request has to show where the money pays off.
- **Data Overview**
- **Outages:** 15-minute counts of customers without power in Pima County, 2020-2025, from Oak Ridge National Laboratory's EAGLE-I system: the yearly 2020-2025 files on its public data record, with the method described in Brelsford et al. (2024).
- **Weather:** hourly temperature, dew point, wind gusts and rain for central Tucson (Open-Meteo, ERA5 reanalysis). Reanalysis is rebuilt after the fact for consistency, so it stands in for what a utility could have observed, not for the exact feed it had at the time.
- **Storm reports:** NOAA's Storm Events Database for Pima County (thunderstorm wind, hail, lightning, flash floods).
- **Cost of an outage:** Lawrence Berkeley National Laboratory's published cost per interruption for homes and businesses (Sullivan et al., 2015), weighted by TEP's customers and electricity sales from EIA-861M.
- **Key Definitions**
- **Customer-minutes:** one customer without power for one minute. Utilities report reliability this way (SAIDI is customer-minutes divided by customers served).
- **Storm night:** 3 PM on a given day through 6 AM the next morning. Monsoon storms build in the afternoon and often run past midnight, so a calendar day splits one storm in two.
- **Big outage night:** a storm night when 5,000 or more customers are out at the same time, about 1.1% of the county's customers.
- **Restoration time:** hours from an outage event's peak until the countywide count falls to 10% of that peak.

The goal is to see what public data can realistically tell a utility about outage risk, and to be clear about where it can't.

2. Data Loading and Cleaning

```
# Load the six yearly outage files (Pima County rows, FIPS 04019)
outage_files <- list.files("data/raw", pattern = "eaglei_pima_", full.names = TRUE)
outages_raw <- map_dfr(outage_files, ~ read_csv(.x, col_types = cols(.default = "c")) %>%
  rename(customers_out = 4) %>%           # the 2023 file names this column "sum"
  select(customers_out, run_start_time))

weather_raw <- read_csv("data/tucson_weather_hourly.csv", col_types = cols(local_time = "c"))
storms_raw  <- read_csv("data/noaa_storm_events_pima.csv")
customers   <- read_csv("data/tep_customers_2025.csv")
lbnl_cost   <- read_csv("data/lbnl_cost_per_event_2013usd.csv")
lbnl_timing <- read_csv("data/lbnl_cost_by_timing_2013usd.csv")
cpi         <- read_csv("data/cpi_u_annual.csv")
reliability <- read_csv("data/eia861_reliability.csv")
```

The outage dataset contains 143,825 15-minute readings. The first step is to look at the data and clean it for analysis.

```
# Glimpse the data
glimpse(outages_raw)
```

```
## Rows: 143,825
## Columns: 2
## $ customers_out  <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", …
## $ run_start_time <chr> "2020-01-01 00:00:00", "2020-01-01 00:15:00", "2020-01-…
```

```
glimpse(weather_raw)
```

```
## Rows: 52,608
## Columns: 6
## $ local_time <chr> "2020-01-01T00:00", "2020-01-01T01:00", "2020-01-01T02:00",…
## $ temp_f     <dbl> 44.5, 44.5, 42.4, 38.6, 38.5, 39.8, 39.4, 41.7, 38.8, 42.9,…
## $ dewpoint_f <dbl> 38.3, 38.1, 37.3, 35.4, 35.0, 35.6, 35.0, 35.7, 35.2, 37.3,…
## $ precip_in  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
## $ wind_mph   <dbl> 2.2, 1.2, 3.0, 4.4, 3.6, 3.6, 3.4, 2.5, 2.9, 3.6, 3.3, 1.6,…
## $ gust_mph   <dbl> 6.9, 2.7, 3.4, 5.4, 5.6, 4.3, 4.9, 4.3, 3.4, 7.2, 9.4, 8.5,…
```

N/A Value Analysis

Before cleaning, let's examine the extent of missing data.

```
# Total N/A Values
sum(is.na(outages_raw$customers_out))
```

```
## [1] 1321
```

```r
sum(is.na(weather_raw))
```

```
## [1] 0
```

```r
# Percentage of N/A Data
(sum(is.na(outages_raw$customers_out)) / nrow(outages_raw)) * 100
```

```
## [1] 0.9184773
```

Less than one percent of the outage readings have a blank count, and the weather file is complete.

Cleaning Steps

1. **Remove blank readings:** a reading with no customer count tells us nothing.
2. **Convert time zones:** EAGLE-I records times in UTC. Arizona does not use daylight saving time, so every reading moves back 7 hours. I checked this against NOAA: on July 14, 2024, outages crossed 1,000 customers at 5:45 PM local time, the same minute NOAA logged thunderstorm wind in Tucson.
3. **Keep 2020-2025 only:** the UTC shift moves a few readings into December 31, 2019.
4. **Remove duplicate readings** for the same 15 minutes.

```r
step1 <- outages_raw %>% mutate(customers_out = as.numeric(customers_out)) %>%
         filter(!is.na(customers_out))
step2 <- step1 %>% mutate(time = with_tz(ymd_hms(run_start_time, tz = "UTC"), "America/Phoenix"))
         %>%
  filter(year(time) %in% 2020:2025)
outages_clean <- step2 %>% distinct(time, .keep_all = TRUE) %>% select(time, customers_out)

removed <- c(blank = nrow(outages_raw) - nrow(step1), outside_2020_2025 = nrow(step1) -
         nrow(step2),
             duplicate = nrow(step2) - nrow(outages_clean))
removed
```

```
##             blank outside_2020_2025         duplicate
##              1321                 3                 0
```

Missing Intervals: Zero or Unknown?

EAGLE-I writes a row only when at least one customer is out, so most missing 15-minute intervals mean nobody was out. But the dataset's documentation is clear that a missing interval can also be a gap in data collection, and the two can't be told apart from the file alone (Brelsford et al., 2024).

So every missing interval is filled with zero as the **base case**, and missing stretches that look suspicious are flagged. The rule: a missing stretch is **uncertain** when the last reading before it had 500 or more customers out. Restoration happens circuit by circuit, so a count going from hundreds

or thousands straight to zero is more likely a reporting gap than every customer coming back at once. As a **sensitivity case**, uncertain stretches are filled by drawing a straight line between the readings on either side.

```
outage_series <- tibble(time = seq(min(outages_clean$time), max(outages_clean$time), by = "15
        min")) %>%
  left_join(outages_clean, by = "time") %>%
  mutate(observed = !is.na(customers_out),
         before = customers_out, after = customers_out) %>%
  fill(before, .direction = "down") %>% fill(after, .direction = "up") %>%
  mutate(gap_id = cumsum(observed),
         uncertain = !observed & before >= 500) %>%
  group_by(gap_id) %>%
  mutate(pos = row_number() - 1, len = n() - 1) %>% ungroup() %>%
  mutate(customers_out_interp = case_when(observed ~ customers_out,
                                          uncertain ~ before + (replace_na(after, 0) - before) *
        pos / (len + 1),
                                          TRUE ~ 0),
         customers_out = replace_na(customers_out, 0)) %>%
  select(time, customers_out, customers_out_interp, observed, uncertain)

gap_runs <- outage_series %>% mutate(run = cumsum(uncertain != lag(uncertain, default = FALSE)))
        %>%
  filter(uncertain) %>% count(run)
expected_intervals <- as.numeric(difftime(as.POSIXct("2026-01-01", tz = "America/Phoenix"),
                                          as.POSIXct("2020-01-01", tz = "America/Phoenix"), units
        = "mins")) / 15

tibble(intervals = nrow(outage_series), expected_for_six_full_years = expected_intervals,
       first = format(min(outage_series$time)), last = format(max(outage_series$time)),
       missing_filled_zero = sum(!outage_series$observed), uncertain_intervals =
         sum(outage_series$uncertain),
       uncertain_stretches = nrow(gap_runs))
```

```
##   intervals expected_for_six_full_years               first                last
## 1    210360                      210432 2020-01-01 11:00:00 2025-12-31 16:45:00
##   missing_filled_zero uncertain_intervals uncertain_stretches
## 1               67864                3165                 112
```

Finding: The cleaned timeline has 210,360 15-minute intervals, 72 short of six full years (18 hours). That is the edge of the data, not a hole in it. The timeline runs from the first reading of 2020 (11:00 AM on January 1, since nobody was out before then) to the last one, which lands at 04:45 PM on December 31, 2025, Arizona time, because the files end at midnight UTC. Inside the timeline, 3,165 intervals in 112 stretches are flagged as uncertain. Filling them in raises total outage time by 17.7%, which is not small, so the gap question carries into the cost range in Section 5 and into how restoration is measured in Section 7.

Sanity Check

```
paste(sum(removed), "readings removed:", removed["blank"], "blank,", removed["outside_2020_2025"],
      "outside 2020-2025 after the time shift,", removed["duplicate"], "duplicate.")
```

```
## [1] "1324 readings removed: 1321 blank, 3 outside 2020-2025 after the time shift, 0 duplicate."
```

3. Feature Engineering

Next, we create the tables and features the models will use.

- **Storm nights:** customer-minutes and peak customers out from 3 PM to 6 AM, plus how many were still out at the last reading before 3 PM (2:45 PM).
- **Weather by 3 PM:** dew point at noon, the high temperature, gusts and rain before 3 PM. A utility would know these by mid-afternoon, before the storm.
- **Weather for the full storm:** the strongest gust and total rain from noon to midnight.
- **Outage events:** a run of 15-minute readings with 500 or more customers out, where gaps under an hour are joined. An event counts if it peaks at 1,000 customers or more.
- **Restoration time:** from the event's peak, the hours until the countywide count first falls to 10% of the peak, searched up to 72 hours ahead. If it never gets there, or an uncertain gap falls in between, the event is set aside rather than given a made-up time.
- **Storm date:** an event that peaks between midnight and 6 AM belongs to the previous afternoon's storm, so it gets that day's weather.
- **Monsoon season:** June 15 through September 30, the National Weather Service's official monsoon window.

```r
# Storm nights: 3 PM on day D through 5:45 AM on day D+1
nights <- outage_series %>%
  mutate(night = as_date(time - hours(15)),           # 3 PM-midnight and 12-6 AM map to the same
         night
         in_night = hour(time) >= 15 | hour(time) < 6) %>%
  group_by(night) %>%
  summarise(night_peak = max(customers_out[in_night], 0),
            night_cmin = sum(customers_out[in_night]) * 15,
            .groups = "drop")

# Customers still out at 2:45 PM, the last reading before the 3 PM decision
before_3pm <- outage_series %>%
  filter(hour(time) == 14, minute(time) == 45) %>%
  transmute(night = as_date(time), out_at_3pm = customers_out)

weather <- weather_raw %>% mutate(time = ymd_hm(local_time, tz = "America/Phoenix"), day =
         as_date(time))
wx_by_3pm <- weather %>% filter(hour(time) < 15) %>% group_by(day) %>%
  summarise(noon_dew = dewpoint_f[hour(time) == 12][1], high_by_3pm = max(temp_f),
            gust_by_3pm = max(gust_mph), rain_by_3pm = sum(precip_in), .groups = "drop")
wx_storm <- weather %>% filter(hour(time) >= 12) %>% group_by(day) %>%
  summarise(storm_gust = max(gust_mph), storm_rain = sum(precip_in), .groups = "drop")
wx_day <- weather %>% group_by(day) %>%
  summarise(day_dew = mean(dewpoint_f), day_high = max(temp_f), .groups = "drop")

storm_reports <- storms_raw %>%
  mutate(day = as_date(dmy_hms(BEGIN_DATE_TIME, tz = "America/Phoenix"))) %>%
  group_by(day) %>%
  summarise(wind_reports = sum(EVENT_TYPE == "Thunderstorm Wind"),
            flood_reports = sum(EVENT_TYPE %in% c("Flash Flood", "Flood", "Debris Flow")),
            hail_reports = sum(EVENT_TYPE == "Hail"), .groups = "drop")

storm_nights <- nights %>% rename(day = night) %>%
  left_join(before_3pm %>% rename(day = night), by = "day") %>%
  left_join(wx_by_3pm, by = "day") %>% left_join(wx_storm, by = "day") %>% left_join(wx_day, by =
         "day") %>%
  left_join(storm_reports, by = "day") %>%
  mutate(across(c(wind_reports, flood_reports, hail_reports, out_at_3pm), ~ replace_na(.x, 0)),
         monsoon = (month(day) == 6 & mday(day) >= 15) | month(day) %in% 7:9,
         big_night = as.integer(night_peak >= 5000),
         season_day = yday(day) - as.integer(leap_year(day) & month(day) > 2),   # day of year on
         a 365-day calendar, so 244 is always September 1
         year = year(day)) %>%
  arrange(day) %>%
  mutate(prev_day_dew = lag(day_dew), prev_night_peak = lag(night_peak)) %>%
  filter(year %in% 2020:2025, !is.na(prev_day_dew), !is.na(noon_dew))

# Outage events, built by a function so the cost sensitivity can rebuild them with other settings
build_events <- function(series, value = "customers_out", on_thr = 500, peak_thr = 1000) {
  x <- series[[value]]; unc <- series$uncertain; tm <- series$time
  runs <- rle(x >= on_thr); run_end <- cumsum(runs$lengths); run_start <- run_end - runs$lengths +
         1
  spans <- tibble(start = run_start[runs$values], end = run_end[runs$values])
  keep <- c(TRUE, spans$start[-1] - spans$end[-nrow(spans)] > 4)       # join runs less than an
         hour apart
  spans <- spans %>% mutate(grp = cumsum(keep)) %>% group_by(grp) %>%
```

```
    summarise(start = min(start), end = max(end), .groups = "drop")
  map_dfr(seq_len(nrow(spans)), function(k) {
    s <- spans$start[k]; e <- spans$end[k]; seg <- x[s:e]
    pk <- max(seg); pi <- s - 1 + which.max(seg)
    lim <- min(length(x), pi + 288)                                    # look up to 72 hours
         ahead
    back <- which(x[pi:lim] <= 0.1 * pk)[1]
    stop_i <- if (is.na(back)) lim else pi + back - 1
    tibble(start = s, end = e, peak = pk, peak_time = tm[pi], cust_min = sum(seg) * 15,
           restore_hrs = if (is.na(back)) NA_real_ else (back - 1) * 0.25,
           censored = is.na(back), gap_touch = any(unc[pi:stop_i]), start_time = tm[s], end_time =
         tm[e])
  }) %>%
    filter(peak >= peak_thr) %>%
    mutate(storm_date = as_date(peak_time) - days(as.integer(hour(peak_time) < 6)),
           peak_hour = hour(peak_time),
           night = as.integer(peak_hour >= 20 | peak_hour < 6),
           monsoon = (month(storm_date) == 6 & mday(storm_date) >= 15) | month(storm_date) %in%
         7:9,
           weekend = as.integer(wday(peak_time) %in% c(1, 7)),
           back_to_back = as.integer(replace_na(as.numeric(difftime(start_time, lag(end_time),
         units = "hours")) <= 24, FALSE)),
           avg_minutes_out = cust_min / peak)
}

events <- build_events(outage_series) %>% left_join(wx_storm, by = c("storm_date" = "day"))

tibble(n_storm_nights = nrow(storm_nights), n_events = nrow(events),
       restoration_measured = sum(!events$censored & !events$gap_touch),
       never_reached_10pct = sum(events$censored), crossed_uncertain_gap = sum(!events$censored &
         events$gap_touch))
```

```
##   n_storm_nights n_events restoration_measured never_reached_10pct
## 1           2191      596                  510                   0
##   crossed_uncertain_gap
## 1                    86
```

4. Exploratory Data Analysis (EDA)

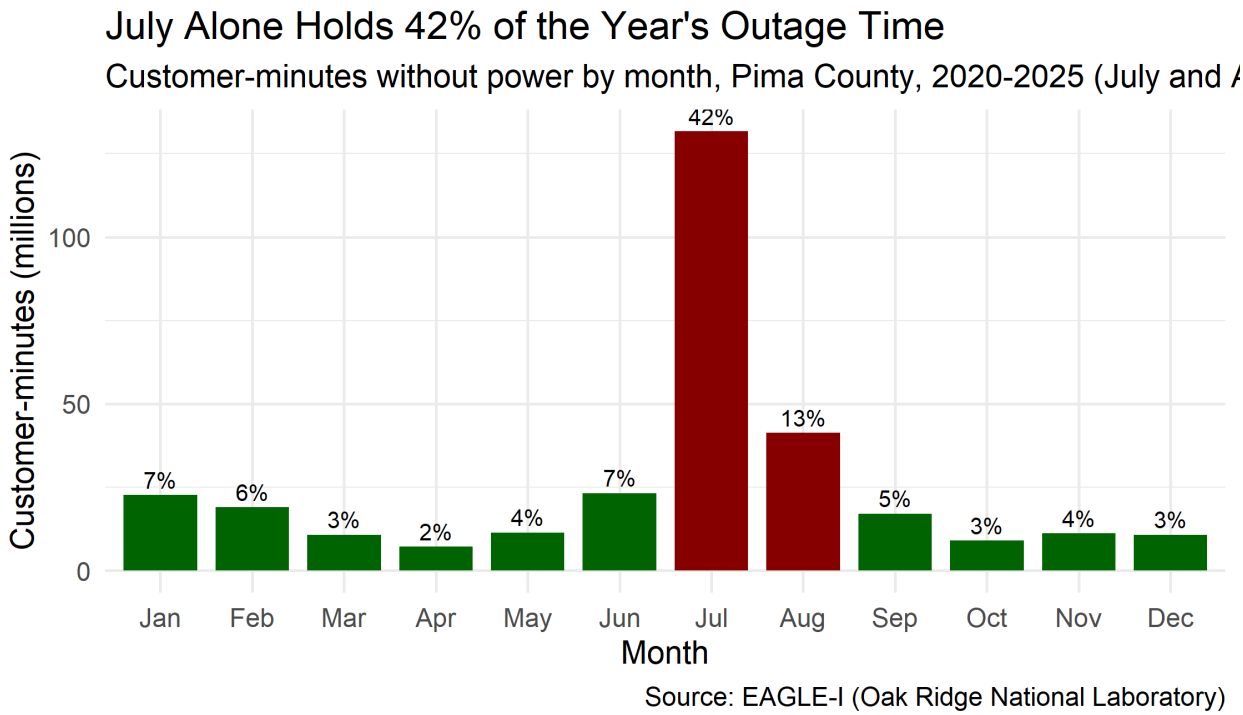
With the data cleaned and new features created, we can explore the patterns in outages.

When Do Outages Happen?

First, let's look at how outage time is spread across the year.

```r
by_month <- outage_series %>%
  group_by(month = month(time, label = TRUE)) %>%
  summarise(cust_min = sum(customers_out) * 15 / 1e6, .groups = "drop") %>%
  mutate(share = cust_min / sum(cust_min))

ggplot(by_month, aes(x = month, y = cust_min, fill = month %in% c("Jul", "Aug"))) +
  geom_col(width = 0.8) +
  geom_text(aes(label = scales::percent(share, accuracy = 1)), vjust = -0.4, size = 3.5) +
  scale_fill_manual(values = c("FALSE" = "darkgreen", "TRUE" = "darkred")) +
  labs(
    title = paste0("July Alone Holds ", scales::percent(by_month$share[by_month$month == "Jul"],
        1),
                   " of the Year's Outage Time"),
    subtitle = "Customer-minutes without power by month, Pima County, 2020-2025 (July and August
        in red)",
    x = "Month", y = "Customer-minutes (millions)",
    caption = "Source: EAGLE-I (Oak Ridge National Laboratory)"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "none")
```



Finding: July holds 42% of all recorded customer-minutes without power and August another 13%. The June 15 to September 30 monsoon window, under a third of the year, holds 66% of all outage time. That timing points at the storms, but it doesn't prove every one of those outages was storm-caused; equipment failures and planned work happen in summer too.

The Worst Nights

```
worst <- storm_nights %>%
  arrange(desc(night_cmin)) %>%
  slice_head(n = 8) %>%
  transmute(Night = format(day, "%b %d, %Y"), `Peak customers out` = fmt(night_peak),
            `Customer-minutes (3 PM-6 AM)` = fmt(night_cmin), `Gust (mph)` = round(storm_gust),
            `Rain (in)` = round(storm_rain, 2), `NOAA wind reports` = wind_reports)
kable(worst, caption = "Eight worst outage nights (3 PM-6 AM), 2020-2025")
```

Eight worst outage nights (3 PM-6 AM), 2020-2025

Night	Peak customers out	Customer-minutes (3 PM-6 AM)	Gust (mph)	Rain (in)	NOAA wind reports
Jul 28, 2023	54,076	26,633,835	31	0.00	2
Jul 14, 2024	38,456	14,404,005	30	0.02	1
Jul 17, 2023	20,224	8,418,720	26	0.00	5
Jul 11, 2024	20,461	5,310,315	37	0.12	2
Jul 11, 2020	21,950	5,126,865	24	0.00	1
Jul 10, 2021	21,567	4,712,805	25	0.01	6
Jul 23, 2023	13,034	4,450,815	25	0.00	1
Nov 12, 2025	18,508	3,896,865	11	0.00	0

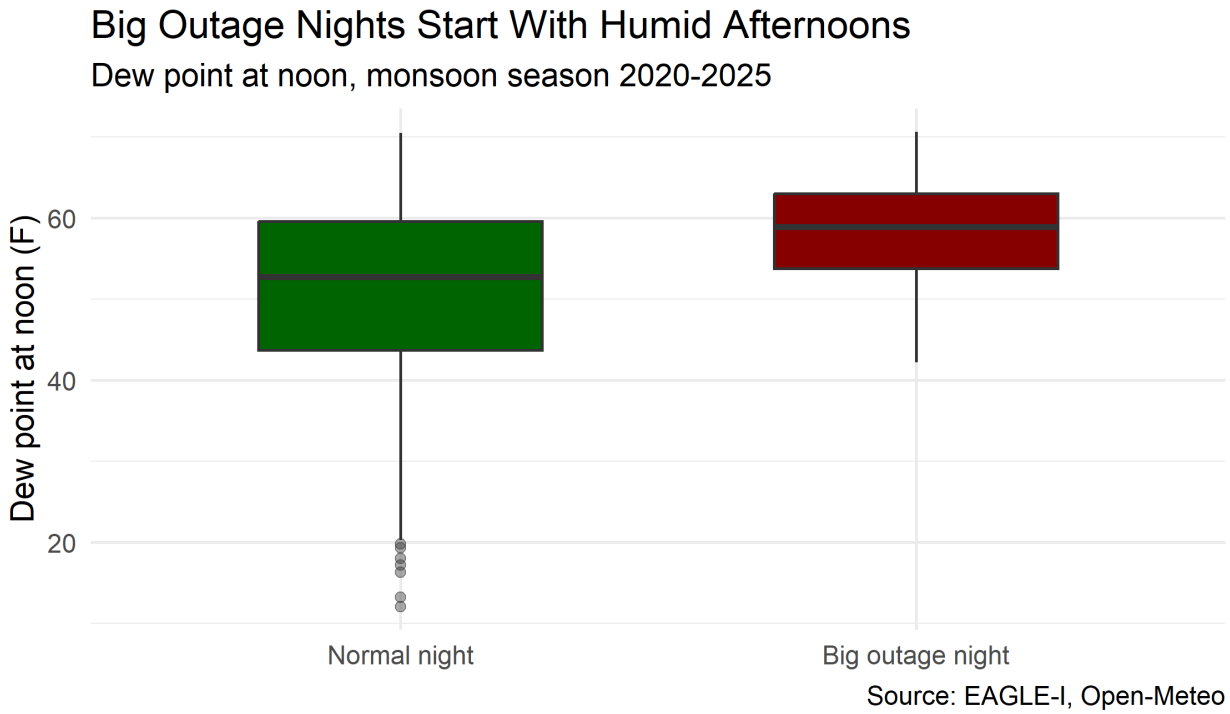
Finding: A handful of nights drive the total. The eight worst nights, out of 2,191, account for 33% of all overnight outage time, and 7 of the eight were in July. The worst, July 28, 2023, left 54,076 customers in the dark at the same moment, and between 3 PM and 6 AM it added up to 26.6 million customer-minutes.

Two details matter for the models later. The central Tucson weather point recorded almost no rain on most of these nights, even with tens of thousands of customers out, because monsoon storms are only a few miles wide and can miss the one point where weather is measured. And one of the eight (November 2025) had light wind and no storm reports, a reminder that not every large outage is weather.

Does Humidity Set Up the Storm?

```
monsoon_nights <- storm_nights %>% filter(monsoon)

ggplot(monsoon_nights, aes(x = factor(big_night, labels = c("Normal night", "Big outage night")),
                           y = noon_dew, fill = factor(big_night))) +
  geom_boxplot(width = 0.55, outlier.alpha = 0.4) +
  scale_fill_manual(values = c("0" = "darkgreen", "1" = "darkred")) +
  labs(
    title = "Big Outage Nights Start With Humid Afternoons",
    subtitle = "Dew point at noon, monsoon season 2020-2025",
    x = NULL, y = "Dew point at noon (F)",
    caption = "Source: EAGLE-I, Open-Meteo"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "none")
```



Finding: On big outage nights the median noon dew point was 59°F, against 53°F on normal monsoon nights. Monsoon thunderstorms feed on moisture, and the moisture is measurable hours before the storm. That is the idea Model 3 tests.

Cross-Checking Against TEP's Own Numbers

The outage counts should move the same way as what TEP reports to the federal government (EIA-861, outage minutes per customer). This is a cross-check on direction, not proof the county data is exact.

```
total_customers <- 456766   # EAGLE-I's Pima County customer count
validation <- outage_series %>%
  group_by(Year = year(time)) %>%
  summarise(`EAGLE-I minutes per customer` = round(sum(customers_out) * 15 / total_customers, 1))
        %>%
  left_join(reliability %>% filter(utility == "TEP", events == "all") %>%
              transmute(Year = year, `TEP reported (all events)` = round(saidi_min, 1)), by =
        "Year")
kable(validation, caption = "Outage minutes per customer: county data vs. TEP's EIA-861 filing")
```

Outage minutes per customer: county data vs. TEP's EIA-861 filing

Year	EAGLE-I minutes per customer	TEP reported (all events)
2020	51.8	60.0
2021	121.7	102.1
2022	82.9	68.8
2023	232.5	168.7
2024	138.8	113.2
2025	61.7	41.8

Finding: The two sources rise and fall in the same years, with 2023 the worst in both. The county figure runs higher than TEP's in five of the six years, likely because Pima County also includes Trico Electric Cooperative's customers, EAGLE-I samples every 15 minutes rather than tracking each customer, and utilities may exclude some events from their filing under IEEE rules. 2020 is the exception, the year EAGLE-I's national coverage was still growing.

5. What Does an Outage Cost?

Lawrence Berkeley National Laboratory pooled 34 customer surveys, run by 10 utilities between 1989 and 2012, on what an outage costs: spoiled food, lost sales, idle staff, damaged equipment (Sullivan et al., 2015). Their Table ES-1 gives the cost per outage by customer type and length, in 2013 dollars. The lab has since released newer versions of its Interruption Cost Estimate (ICE) Calculator, most recently version 2.2 in February 2026; this analysis uses the 2015 tables because they are published in full and can be checked line by line.

```
kable(lbnl_cost, col.names = c("Customer class", "Momentary", "30 min", "1 hour", "4 hours", "8
        hours", "16 hours"),
      caption = "LBNL cost per interruption event (2013 dollars)")
```

LBNL cost per interruption event (2013 dollars)

Customer class	Momentary	30 min	1 hour	4 hours	8 hours	16 hours
Residential	3.9	4.5	5.1	9.5	17.2	32.4
Small C&I	412.0	520.0	647.0	1880.0	4690.0	9055.0
Medium and Large C&I	12952.0	15241.0	17804.0	39458.0	84083.0	165482.0

Three steps turn this into a cost for each Pima County outage:

1. **Customer mix:** LBNL splits businesses at 50,000 kWh a year, not by billing class, and TEP doesn't publish its accounts by size. So the number of medium and large accounts is estimated: if TEP's businesses used what LBNL's average small business (19,214 kWh a year) and average medium or large one (7,140,501 kWh) use, how many large accounts would make TEP's 2025 business sales add up? That is an estimate under LBNL's averages, not a count, and it assumes the customers hit by outages look like TEP's customer base as a whole.
2. **Outage length:** for each event, customer-minutes divided by peak customers gives an average length per customer. This is a proxy, since a long storm can knock out different customers at different times. The cost per customer is read off the LBNL curve at that length, held at the curve's ends below 5 minutes and above 16 hours.
3. **Today's dollars:** multiply by the change in the Consumer Price Index from 2013 to 2025.

```
# The file has one row per month of 2025: customer counts are averaged, sales are summed to a year
stopifnot(nrow(customers) == 12)
mix <- customers %>% summarise(res = mean(res_cust), ci = mean(com_cust) + mean(ind_cust),
                               ci_kwh = (sum(com_mwh) + sum(ind_mwh)) * 1000)
large_n <- (mix$ci_kwh - mix$ci * 19214) / (7140501 - 19214)     # energy balance with LBNL's
        average customer sizes
mix_weights <- function(large) c(Residential = mix$res, `Small C&I` = mix$ci - large, `Medium and
        Large C&I` = large) /
  (mix$res + mix$ci)
weights <- mix_weights(large_n)
cpi_factor <- cpi$cpi_u_annual_avg[cpi$year == 2025] / cpi$cpi_u_annual_avg[cpi$year == 2013]

duration_points <- c(5, 30, 60, 240, 480, 960)          # minutes: momentary, 30 min, 1, 4, 8, 16
        hours
cost_matrix <- as.matrix(lbnl_cost[match(names(weights), lbnl_cost$customer_class), -1])
blend <- function(w) as.numeric(w %*% cost_matrix)
blended_2013 <- blend(weights)
residential_2013 <- cost_matrix[1, ]

cost_per_customer <- function(minutes, curve = blended_2013) {
  approx(duration_points, curve, xout = pmin(pmax(minutes, 5), 960))$y * cpi_factor
}

events <- events %>%
  mutate(cost = peak * cost_per_customer(avg_minutes_out),
         cost_res_only = peak * cost_per_customer(avg_minutes_out, residential_2013))

c(large_accounts = round(large_n), share_of_business_accounts = round(large_n / mix$ci, 4))
```

```
##             large_accounts share_of_business_accounts
##                   590.0000                     0.0138
```

```
round(weights, 4)
```

```
##          Residential            Small C&I Medium and Large C&I
##               0.9068               0.0919               0.0013
```

```
round(cpi_factor, 3)
```

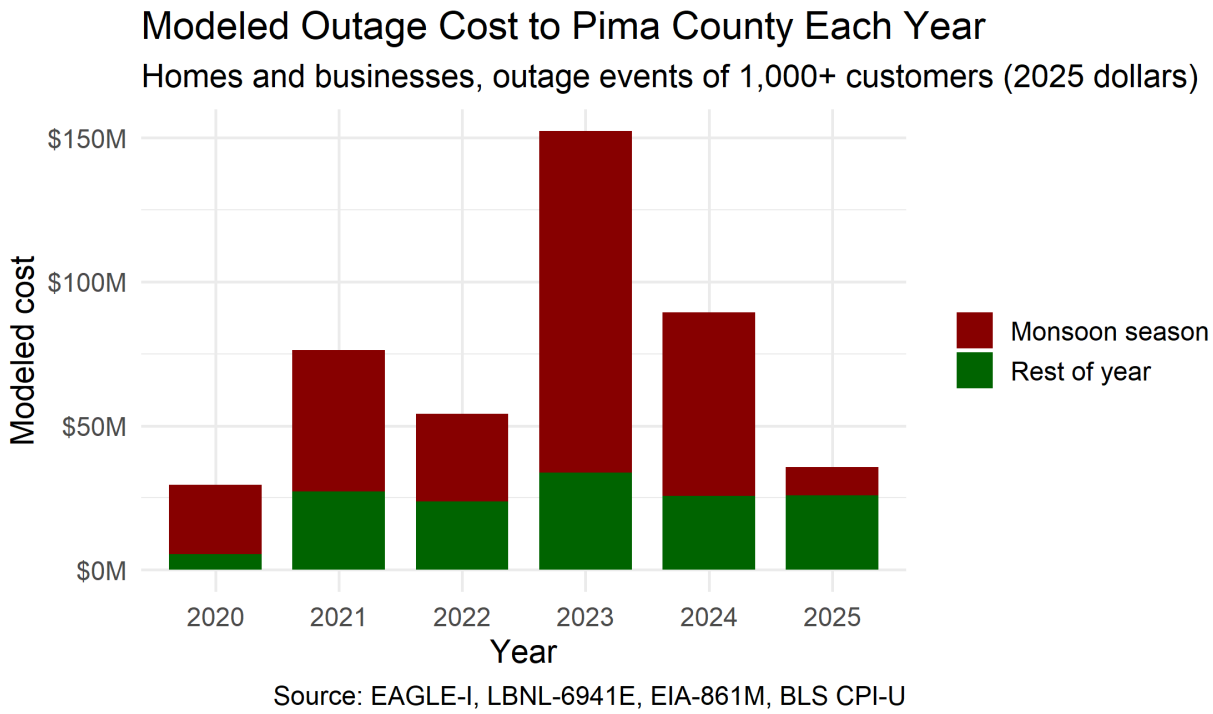
```
## [1] 1.382
```

```
c(events_below_5_min = sum(events$avg_minutes_out < 5), events_above_16_hours =
         sum(events$avg_minutes_out > 960))
```

```
##    events_below_5_min events_above_16_hours
##                     0                     0
```

```
annual_cost <- events %>%
  group_by(year = year(storm_date)) %>%
  summarise(monsoon_cost = sum(cost[monsoon]) / 1e6, rest_cost = sum(cost[!monsoon]) / 1e6,
         .groups = "drop") %>%
  pivot_longer(c(monsoon_cost, rest_cost), names_to = "season", values_to = "cost_m")

ggplot(annual_cost, aes(x = factor(year), y = cost_m, fill = season)) +
  geom_col(width = 0.75) +
  scale_fill_manual(values = c(monsoon_cost = "darkred", rest_cost = "darkgreen"),
                    labels = c("Monsoon season", "Rest of year")) +
  scale_y_continuous(labels = scales::dollar_format(suffix = "M")) +
  labs(
    title = "Modeled Outage Cost to Pima County Each Year",
    subtitle = "Homes and businesses, outage events of 1,000+ customers (2025 dollars)",
    x = "Year", y = "Modeled cost", fill = NULL,
    caption = "Source: EAGLE-I, LBNL-6941E, EIA-861M, BLS CPI-U"
  ) +
  theme_minimal(base_size = 14)
```



Finding: Under these assumptions, outages cost Pima County's homes and businesses an average of **$72,896,178 a year** over 2020-2025. That is a modeled estimate, not a measured loss. 68% of it falls in monsoon season, and 2023 was the worst year at $152,349,541. Businesses carry 96% of the modeled cost even though they are about one customer in ten, which means the business assumptions drive the number more than anything else.

How Sure Is the Number?

Every assumption above can move the estimate, so the model is re-run with each one changed on its own: the share of large business accounts, the share of businesses among the customers actually interrupted (businesses may sit on different feeders than homes), how outage length is measured, how the uncertain gaps are handled, what counts as an outage event, and when the outage happened (LBNL's Table ES-2 prices outages by season and time of day; summer is June-September, morning 6 AM-12 PM, afternoon 12-5 PM, evening 5-10 PM, night 10 PM-6 AM).

```r
annual <- function(ev, w = weights, len_mult = 1) {
  sum(ev$peak * cost_per_customer(ev$avg_minutes_out * len_mult, blend(w))) / yrs_cost
}
# Businesses' share of interrupted customers scaled by k; homes make up the rest
exposure <- function(k, w = weights) { w[2:3] <- w[2:3] * k; w[1] <- 1 - sum(w[2:3]); w }

# Priced by season and hour of the peak (LBNL Table ES-2)
curve_for <- function(class, season, slot) {
  as.numeric(lbnl_timing %>% filter(customer_class == class, season == !!season, time_of_day ==
        slot) %>%
              select(momentary:hr_16))
}
timed_cost <- map_dbl(seq_len(nrow(events)), function(i) {
  h <- events$peak_hour[i]; season <- ifelse(month(events$peak_time[i]) %in% 6:9, "summer", "non-
        summer")
  res_slot <- case_when(h >= 12 & h < 17 ~ "afternoon", h >= 17 & h < 22 ~ "evening", TRUE ~
        "morning/night")
  com_slot <- case_when(h >= 6 & h < 12 ~ "morning", h >= 12 & h < 17 ~ "afternoon", TRUE ~
        "evening/night")
  curve <- weights["Residential"] * curve_for("Residential", season, res_slot) +
           weights["Small C&I"] * curve_for("Small C&I", season, com_slot) +
           weights["Medium and Large C&I"] * curve_for("Medium and Large C&I", season, "all")
  events$peak[i] * cost_per_customer(events$avg_minutes_out[i], curve)
})

events_interp <- build_events(outage_series, value = "customers_out_interp")
events_wide <- build_events(outage_series, on_thr = 250, peak_thr = 500)

sensitivity <- tibble(
  Assumption = c("Base case",
                 "Half as many large business accounts", "Twice as many large business accounts",
                 "Businesses' share of those interrupted halved", "Businesses' share of those
        interrupted doubled",
                 "Outages 25% shorter than the proxy", "Outages 25% longer than the proxy",
                 "Uncertain gaps filled in", "Smaller events counted (peak 500+)",
                 "Priced by season and hour (Table ES-2)",
                 "Low settings together", "High settings together",
                 "Every customer priced as a home"),
  Type = c("Base", rep("One at a time", 9), "Combined", "Combined", "Floor"),
  `Annual cost` = c(annual(events),
                    annual(events, mix_weights(large_n / 2)), annual(events, mix_weights(large_n *
        2)),
                    annual(events, exposure(0.5)), annual(events, exposure(2)),
                    annual(events, len_mult = 0.75), annual(events, len_mult = 1.25),
                    annual(events_interp), annual(events_wide),
                    sum(timed_cost) / yrs_cost,
                    annual(events, exposure(0.5, mix_weights(large_n / 2)), len_mult = 0.75),
                    annual(events_interp, exposure(2, mix_weights(large_n * 2)), len_mult = 1.25),
                    avg_res)) %>%
  mutate(`vs. base` = pct(`Annual cost` / `Annual cost`[1] - 1), value = `Annual cost`, `Annual
        cost` = dollars(`Annual cost`))
kable(sensitivity %>% select(-value), caption = "Modeled annual outage cost when assumptions
        change (2025 dollars)")
```

Modeled annual outage cost when assumptions change (2025 dollars)

Assumption	Type	Annual cost	vs. base
Base case	Base	$72,896,178	0%
Half as many large business accounts	One at a time	$64,917,449	-11%
Twice as many large business accounts	One at a time	$88,853,636	22%
Businesses' share of those interrupted halved	One at a time	$38,066,892	-48%
Businesses' share of those interrupted doubled	One at a time	$142,554,750	96%
Outages 25% shorter than the proxy	One at a time	$59,099,367	-19%
Outages 25% longer than the proxy	One at a time	$87,162,418	20%
Uncertain gaps filled in	One at a time	$84,500,289	16%
Smaller events counted (peak 500+)	One at a time	$77,510,063	6%
Priced by season and hour (Table ES-2)	One at a time	$70,624,190	-3%
Low settings together	Combined	$27,604,880	-62%
High settings together	Combined	$242,356,049	232%
Every customer priced as a home	Floor	$3,237,605	-96%

Finding: The business assumptions move the result the most. If businesses make up half or twice their assumed share of the customers interrupted, the estimate moves -48% to 96%. Halving or doubling the number of large accounts moves it -11% to 22%, since one hour without power costs a large business about 3,500 times what it costs a home. The outage-length proxy (-19% to 20%) and the uncertain gaps (16%) come next. The event definition and time of day move it less.

Changed one at a time, the alternatives span $38,066,892 to $142,554,750 a year. That is a **one-at-a-time sensitivity range**, not a confidence interval: it shows which assumptions matter, not where the true cost probably falls. The assumptions can also stack. With every low setting at once the estimate is $27,604,880, and with every high setting at once it's $242,356,049. Putting odds on those corners would take a probability distribution for each input (a Monte Carlo simulation), and public data can't back those distributions up. So $72,896,178 is the central estimate under stated assumptions, and the business mix is the first thing better data should pin down.

One more point on scope: the $72,896,178 covers every qualifying outage event in the year, not just monsoon storms. The monsoon-season share is about $49,300,678, and even that is a seasonal total, not a count of storm-caused outages.

6. Model 1: Multiple Linear Regression - What Drives Storm-Night Outage Severity?

How It Works

- Multiple linear regression fits a straight-line relationship between one outcome and several inputs at once, so each coefficient is the association with that input **holding the others fixed**.
- The outcome is outage severity: customer-minutes on each monsoon storm night. It is very skewed, with most nights near zero and a few in the millions, so the model uses the **log** of customer-minutes. A coefficient *b* then means each one-unit increase goes with a (e^b - 1) x 100 percent change.
- Rain is measured in **tenths of an inch**, because rain at a single point is usually well under an inch.
- The inputs are on different scales, so the model is also fit on **standardized** inputs (each one in standard deviations). That puts the effects on a common footing: the change in outage time for a typical-sized change in each input.
- **Weather only.** NOAA storm reports are left out of the main model on purpose: in Arizona a "thunderstorm wind" report is often filed *because* wind knocked down a power line, so using it to predict outages would be partly circular.

Training the Model

```
monsoon_nights <- monsoon_nights %>% mutate(storm_rain_tenths = storm_rain * 10)
model1 <- lm(log1p(night_cmin) ~ storm_gust + storm_rain_tenths + noon_dew + day_high, data =
        monsoon_nights)

model1_table <- tidy(model1, conf.int = TRUE) %>%
  filter(term != "(Intercept)") %>%
  mutate(`% change` = round((exp(estimate) - 1) * 100, 1),
        `95% CI (%)` = paste0(round((exp(conf.low) - 1) * 100), " to ", round((exp(conf.high) -
        1) * 100)),
        across(c(estimate, std.error, statistic), ~ round(.x, 3)), p.value = signif(p.value, 2))
        %>%
  select(term, estimate, std.error, p.value, `% change`, `95% CI (%)`)
kable(model1_table, caption = "Model 1: log(customer-minutes + 1) on storm-night weather, per unit
        of each input")
```

Model 1: log(customer-minutes + 1) on storm-night weather, per unit of each input

term	estimate	std.error	p.value	% change	95% CI (%)
storm_gust	0.085	0.022	7.9e-05	8.9	4 to 14
storm_rain_tenths	0.677	0.152	9.9e-06	96.8	46 to 165
noon_dew	0.063	0.012	1.0e-07	6.5	4 to 9
day_high	0.104	0.023	5.4e-06	10.9	6 to 16

```
model1_std <- lm(log1p(night_cmin) ~ scale(storm_gust) + scale(storm_rain_tenths) +
         scale(noon_dew) + scale(day_high),
                data = monsoon_nights)
std_table <- tidy(model1_std, conf.int = TRUE) %>% filter(term != "(Intercept)") %>%
  mutate(term = gsub("scale\\(|\\)", "", term),
         `% change per 1 SD` = round((exp(estimate) - 1) * 100),
         `95% CI (%)` = paste0(round((exp(conf.low) - 1) * 100), " to ", round((exp(conf.high) -
         1) * 100)),
         `1 SD equals` = c(paste(round(sd(monsoon_nights$storm_gust), 1), "mph"),
                           paste(round(sd(monsoon_nights$storm_rain), 2), "in"),
                           paste(round(sd(monsoon_nights$noon_dew), 1), "°F"),
                           paste(round(sd(monsoon_nights$day_high), 1), "°F"))) %>%
  select(term, `1 SD equals`, `% change per 1 SD`, `95% CI (%)`)
kable(std_table, caption = "The same model on standardized inputs")
```

The same model on standardized inputs

term	1 SD equals	% change per 1 SD	95% CI (%)
storm_gust	5.7 mph	63	28 to 108
storm_rain_tenths	0.08 in	76	37 to 125
noon_dew	11.5 °F	108	59 to 172
day_high	6.6 °F	98	48 to 165

```
glance(model1) %>% select(r.squared, adj.r.squared, nobs) %>% round(3)
```

```
##   r.squared adj.r.squared nobs
## 1     0.105           0.1  648
```

Finding: Every weather input goes with more outage time. Holding the rest constant, each extra mph of peak gust is associated with about 8.9% more storm-night customer-minutes, each tenth of an inch of rain with 96.8%, each °F of noon dew point with 6.5% and each °F of daily high with 10.9%. These are associations, not proof that one more tenth of an inch would cause that much more outage time.

The standardized version compares them fairly. A one-standard-deviation increase goes with 63% more outage time for gusts, 76% for rain, 108% for dew point and 98% for the daily high, so on a common scale the heat and moisture inputs look broadly comparable to wind and rain. That's a rough comparison: the intervals are wide and overlap, and overlapping intervals don't prove two effects are equal.

The model explains 11% of the night-to-night variation in *log* customer-minutes, which is weak. One plausible reason is that weather is measured at one point in central Tucson while monsoon storms are only a few miles across, so a storm can flatten lines in Marana while the Tucson weather point stays dry. The data can't show that this is the main reason, though. Which lines are exposed, what actually broke, and how outages get reported all vary in ways the model doesn't see. Adding NOAA's

storm-report counts raises R^2 to 19%, but as explained above, those reports are partly a record of the outages themselves.

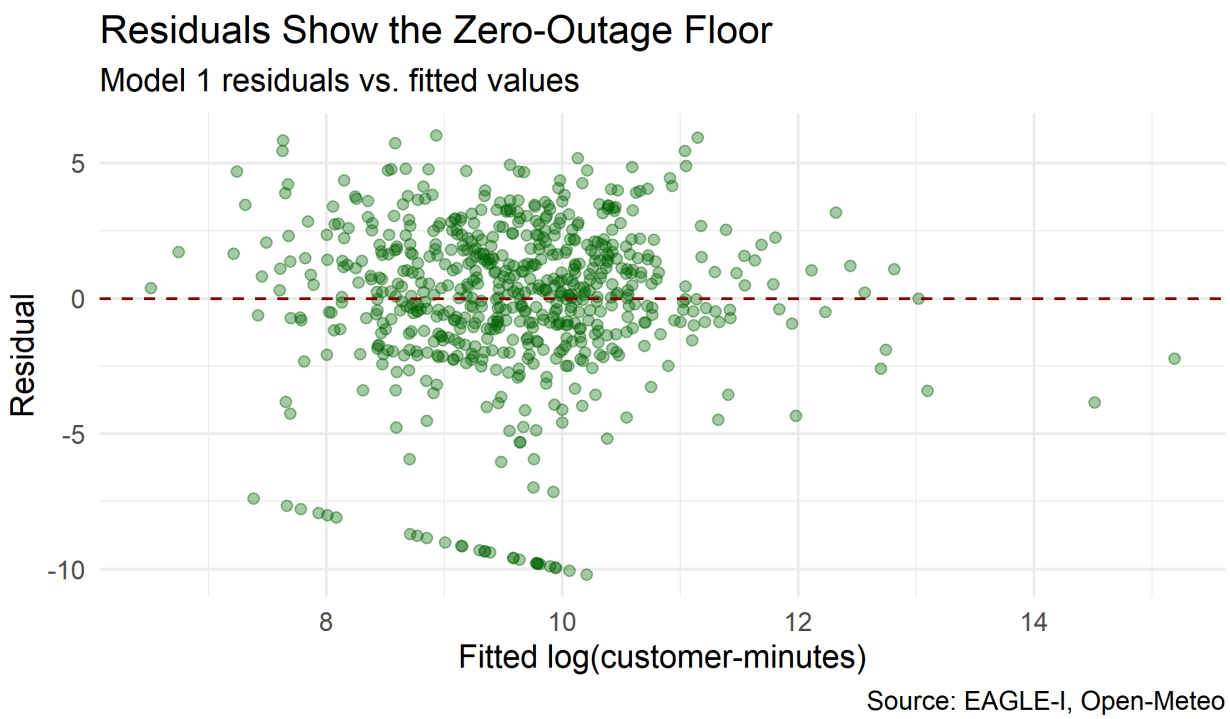
Diagnostics

- **Multicollinearity (VIF):** the variance inflation factor measures how much each input overlaps with the others. Values under 5 mean each coefficient can be read on its own.
- **Residuals:** the errors should scatter evenly around zero with no pattern.

```
round(vif(model1), 2)
```

```
##        storm_gust storm_rain_tenths          noon_dew          day_high
##              1.12              1.18              1.37              1.63
```

```
ggplot(augment(model1), aes(x = .fitted, y = .resid)) +
  geom_point(alpha = 0.35, color = "darkgreen") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "darkred") +
  labs(
    title = "Residuals Show the Zero-Outage Floor",
    subtitle = "Model 1 residuals vs. fitted values",
    x = "Fitted log(customer-minutes)", y = "Residual",
    caption = "Source: EAGLE-I, Open-Meteo"
  ) +
  theme_minimal(base_size = 14)
```



Finding: All VIF values are low, so heat, humidity, wind and rain each carry their own information. The residual plot shows a diagonal edge on the lower left: nights with almost no outages sit on a floor the model can't go below. That is a limit of using a straight line for an outcome that is often near zero, and it is one reason the yes/no question in Model 3 is worth asking separately.

7. Model 2: Linear Regression - What Goes With Slower Restoration?

How It Works

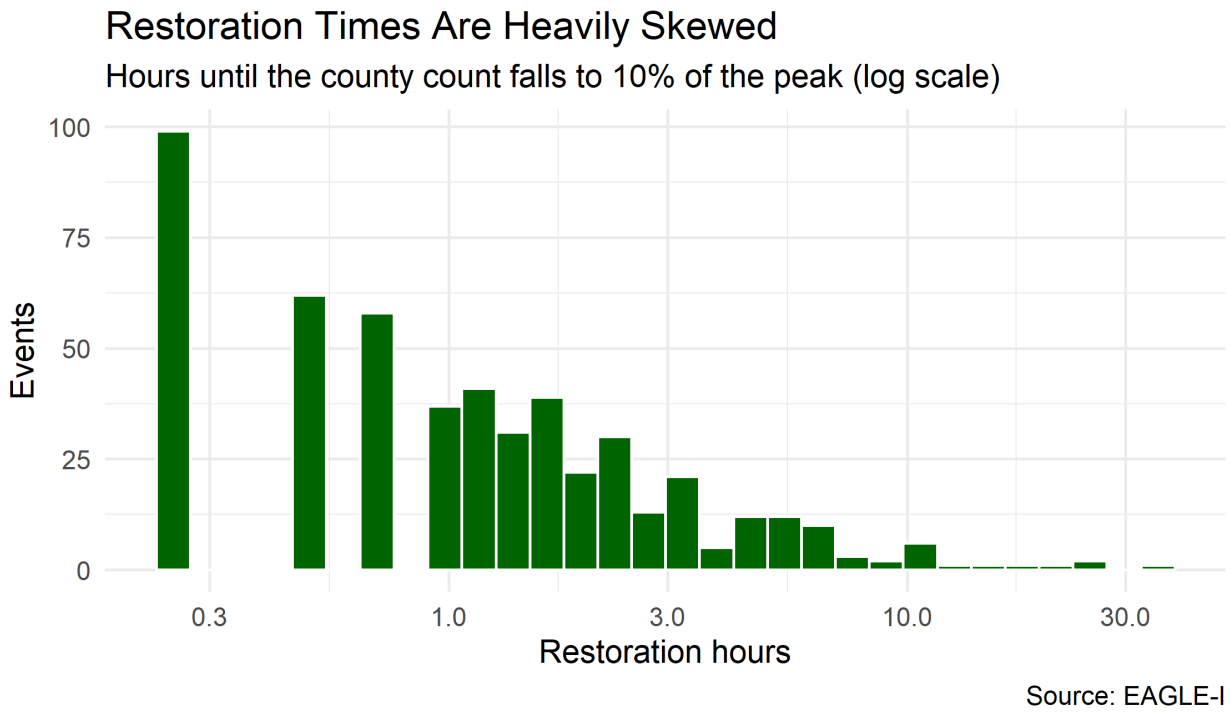
- The same method, a different unit: each outage event, all year round.
- The outcome is the **log of restoration hours**, so coefficients read as percent changes.
- Restoration is measured from county totals, so it is the time until the countywide count falls to 10% of the event's peak. That isn't quite the same as following the same customers until they're back on, because new outages can start while old ones are being fixed.
- Events are only used when that point is actually observed: events where the count never fell to 10% within 72 hours, or where an uncertain gap falls in between, are set aside. If the set-aside events differ from the rest, the model describes only the measurable ones, so the two groups are compared below.
- Several events can come from the same storm, so the confidence intervals are also computed **clustered by storm date**.
- The inputs: event size, whether it peaked at night (8 PM to 6 AM), whether another event hit in the previous 24 hours, weekends, monsoon season, and the storm's wind and rain.

Training the Model

```
events_m2 <- events %>% filter(!censored, !gap_touch, restore_hrs > 0, !is.na(storm_gust)) %>%
  mutate(storm_rain_tenths = storm_rain * 10)
c(events = nrow(events), used = nrow(events_m2), never_reached_10pct = sum(events$censored),
  crossed_uncertain_gap = sum(!events$censored & events$gap_touch))
```

```
##                events                  used   never_reached_10pct
##                   596                   510                     0
## crossed_uncertain_gap
##                    86
```

```
ggplot(events_m2, aes(restore_hrs)) +
  geom_histogram(bins = 30, fill = "darkgreen", color = "white") +
  scale_x_log10() +
  labs(title = "Restoration Times Are Heavily Skewed", subtitle = "Hours until the county count
         falls to 10% of the peak (log scale)",
       x = "Restoration hours", y = "Events", caption = "Source: EAGLE-I") +
  theme_minimal(base_size = 14)
```




```
model2 <- lm(log(restore_hrs) ~ log(peak) + night + back_to_back + weekend + monsoon + storm_gust
         + storm_rain_tenths,
             data = events_m2)

# Heteroskedasticity-robust (HC3) confidence intervals, since the spread of restoration times
         varies by event size
robust_se <- sqrt(diag(hccm(model2, type = "hc3")))
model2_table <- tidy(model2) %>%
  mutate(robust_se = robust_se, low = estimate - 1.96 * robust_se, high = estimate + 1.96 *
         robust_se,
         p_robust = 2 * pnorm(-abs(estimate / robust_se))) %>%
  filter(term != "(Intercept)") %>%
  mutate(scale = ifelse(term == "log(peak)", log(2), 1),
         `% change` = round((exp(estimate * scale) - 1) * 100, 1),
         `95% CI (%)` = paste0(round((exp(low * scale) - 1) * 100), " to ", round((exp(high *
         scale) - 1) * 100)),
         p.value = signif(p_robust, 2), estimate = round(estimate, 3)) %>%
  select(term, estimate, p.value, `% change`, `95% CI (%)`)
kable(model2_table, caption = "Model 2: log(restoration hours), robust 95% intervals. For
         log(peak), the % is for twice as many customers out.")
```

Model 2: log(restoration hours), robust 95% intervals. For log(peak), the % is for twice as many customers out.

term	estimate	p.value	% change	95% CI (%)
log(peak)	0.213	1.8e-03	15.9	6 to 27
night	0.478	7.0e-06	61.2	31 to 99
back_to_back	-0.055	5.9e-01	-5.3	-23 to 16
weekend	0.135	1.5e-01	14.5	-5 to 38
monsoonTRUE	0.306	1.1e-03	35.8	13 to 63
storm_gust	0.008	2.7e-01	0.8	-1 to 2
storm_rain_tenths	0.059	3.2e-02	6.1	1 to 12

```
glance(model2) %>% select(r.squared, adj.r.squared, nobs) %>% round(3)
```

```
##   r.squared adj.r.squared nobs
## 1      0.11         0.097  510
```

```
# Intervals clustered by storm date (events from the same storm share errors)
X <- model.matrix(model2); e <- resid(model2); g <- events_m2$storm_date
XtXi <- solve(crossprod(X)); u <- rowsum(X * e, g)
G <- length(unique(g)); n <- nrow(X); k <- ncol(X)
vc_cluster <- G / (G - 1) * (n - 1) / (n - k) * XtXi %*% crossprod(u) %*% XtXi
night_cl <- (exp(coef(model2)["night"] + c(-1.96, 1.96) * sqrt(vc_cluster["night", "night"])) - 1)
         * 100
c(storm_dates = G, night_low = round(night_cl[1]), night_high = round(night_cl[2]))
```

```
## storm_dates   night_low  night_high
##         447          31          99
```

```
# Are the set-aside events different?
selection <- events %>%
  mutate(group = ifelse(!censored & !gap_touch & restore_hrs > 0 & !is.na(storm_gust), "Used in
         Model 2", "Set aside")) %>%
  group_by(group) %>%
  summarise(Events = n(), `Median peak` = fmt(median(peak)), `Peaked at night` = pct(mean(night)),
            `Monsoon season` = pct(mean(monsoon)), `Median gust (mph)` = round(median(storm_gust,
         na.rm = TRUE)),
            `Share of modeled cost` = pct(sum(cost) / sum(events$cost)), .groups = "drop")
kable(selection, caption = "Events used in Model 2 vs. events set aside")
```

Events used in Model 2 vs. events set aside

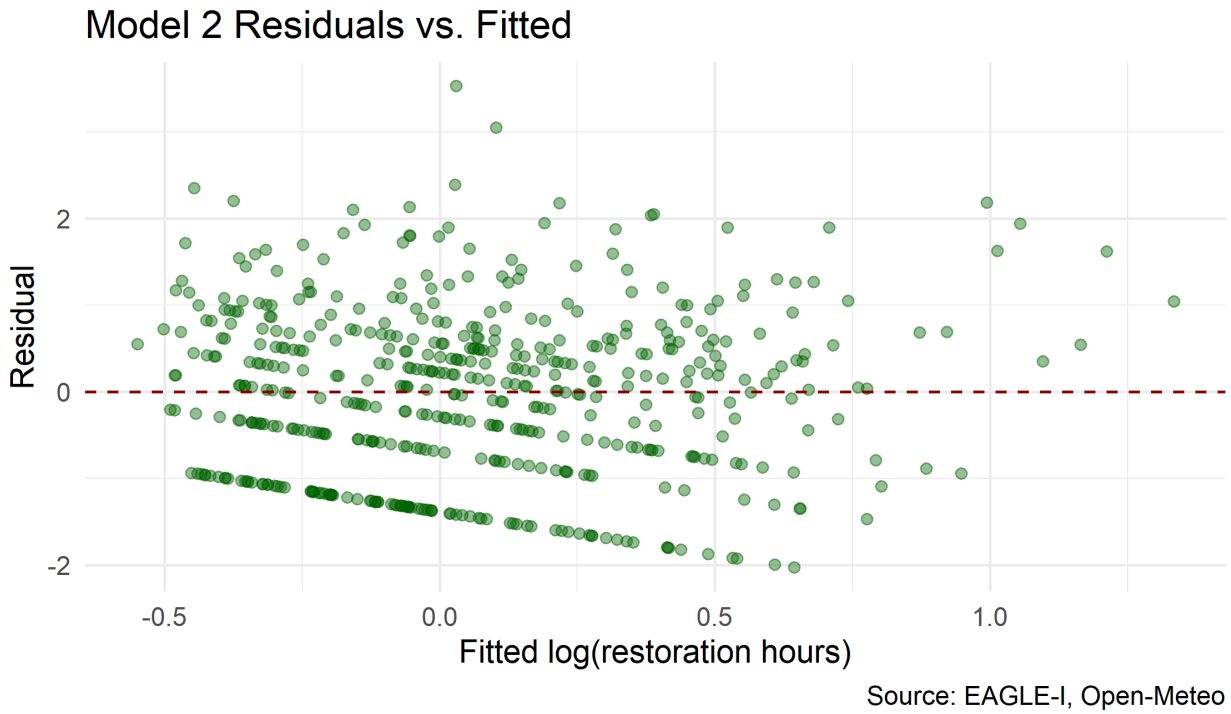
group	Events	Median peak	Peaked at night	Monsoon season	Median gust (mph)	Share of modeled cost
Set aside	86	2,192	21%	29%	21	8%
Used in Model 2	510	2,100	25%	50%	22	92%

```r
cooks <- cooks.distance(model2)
c(influential_events = sum(cooks > 4 / nrow(events_m2)))
```

```
## influential_events
##                 31
```

```r
model2_trim <- update(model2, data = events_m2[cooks <= 4 / nrow(events_m2), ])

ggplot(augment(model2), aes(.fitted, .resid)) +
  geom_point(alpha = 0.4, color = "darkgreen") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "darkred") +
  labs(title = "Model 2 Residuals vs. Fitted", x = "Fitted log(restoration hours)", y =
         "Residual",
       caption = "Source: EAGLE-I, Open-Meteo") +
  theme_minimal(base_size = 14)
```



Finding: For the 510 events where restoration could be measured, the median time for the county count to fall to 10% of the peak was 1.0 hours. Holding size and weather fixed, an outage that peaks at night is associated with 61.2% longer restoration (robust 95% interval 31 to 99%, $p < 0.001$). Dropping the 31 most influential events gives 69%, so the result doesn't hinge on a few unusual events. Clustering the errors by storm date, so that several events from one storm don't count as independent evidence, gives 31% to 99%. Twice as many customers out goes with 15.9% longer restoration. The terms significant at the 5% level are: log(peak), night, monsoonTRUE, storm_rain_tenths.

The 86 set-aside events aren't the severe ones. They have a similar median peak (2,192 vs. 2,100) and a similar night share (21% vs. 25%), but fewer fall in monsoon season (29% vs. 50%), and together they carry only 8% of modeled cost. So excluding them doesn't look like it hides the worst events, but strictly the model describes the events whose restoration could be measured.

This is an association. Night outages may differ from daytime ones in ways the data doesn't show: harder access in the dark, more lightning damage, different equipment, or fewer crews on shift. The analysis can't separate those, so the night difference is a place to look, not a measured staffing effect.

8. Model 3: Logistic Regression - Can We Flag a Big Outage Night by 3 PM?

How It Works

- Logistic regression predicts a **probability** for a yes/no outcome: will this be a big outage night (5,000+ customers out at once)?
- It models the **log-odds** of "yes" as a straight line in the inputs, then turns that into a probability between 0 and 1. A coefficient *b* means each one-unit increase multiplies the **odds** by e^b, the **odds ratio**.
- Every input is something an operations center knows **by 3 PM**, before most monsoon storms hit.
- Days where 5,000+ customers are **still** out at 2:45 PM, the last reading before the 3 PM decision, are removed. On those days a big outage is already under way, and keeping them would let the model cheat. A morning outage that was fixed by noon stays in, because at 3 PM the night ahead is still unknown.

Preparing the Data for Modeling

```
already_out <- sum(monsoon_nights$out_at_3pm >= 5000)
paste(already_out, "monsoon days removed: 5,000+ customers still out at 2:45 PM.")
```

```
## [1] "4 monsoon days removed: 5,000+ customers still out at 2:45 PM."
```

```
model_data <- monsoon_nights %>%
  filter(out_at_3pm < 5000) %>%
  select(day, big_night, year, noon_dew, high_by_3pm, gust_by_3pm, rain_by_3pm, prev_day_dew,
         prev_night_peak, season_day) %>%
  mutate(log_prev_night_peak = log1p(prev_night_peak)) %>%
  na.omit()
table(model_data$big_night)
```

```
##
##   0   1
## 610  34
```

Train-Test Split

A random 80/20 split is the standard for most classification problems, but it doesn't work for data over time: a random split would let the model learn from July 14, 2024 and then be "tested" on the day after. Instead, the main model **trains on 2020-2023 and is tested once on 2024-2025**, the way it would actually be used: built on the past, judged on the future.

```
train_data <- model_data %>% filter(year <= 2023)
test_data  <- model_data %>% filter(year >= 2024)
c(train_nights = nrow(train_data), train_big = sum(train_data$big_night),
  test_nights = nrow(test_data), test_big = sum(test_data$big_night))
```

```
## train_nights    train_big  test_nights     test_big
##          430           24          214           10
```

Choosing the Inputs

With only 24 big nights to learn from, letting an algorithm pick inputs (like stepwise selection) can latch onto noise. So the model is fixed in advance from how monsoon storms work: thunderstorms need **moisture** and **heat**, so the inputs are the noon dew point and the high temperature by 3 PM. A full model with all seven available inputs is kept as a comparison.

```
logit_model <- glm(big_night ~ noon_dew + high_by_3pm, data = train_data, family = "binomial")
full_formula <- big_night ~ noon_dew + high_by_3pm + gust_by_3pm + rain_by_3pm + prev_day_dew +
         log_prev_night_peak + season_day
full_model <- glm(full_formula, data = train_data, family = "binomial")

odds_table <- tidy(logit_model, conf.int = TRUE, exponentiate = TRUE) %>%
  filter(term != "(Intercept)") %>%
  mutate(across(c(estimate, std.error, statistic, conf.low, conf.high), ~ round(.x, 3)), p.value =
         signif(p.value, 2)) %>%
  rename(odds_ratio = estimate)
kable(odds_table, caption = "Model 3: odds ratios (trained on 2020-2023)")
```

Model 3: odds ratios (trained on 2020-2023)

term	odds_ratio	std.error	statistic	p.value	conf.low	conf.high
noon_dew	1.146	0.040	3.420	0.00063	1.068	1.248
high_by_3pm	1.166	0.047	3.296	0.00098	1.070	1.285

Finding: Each °F of noon dew point multiplies the odds of a big outage night by 1.15, so a 10°F more humid afternoon multiplies them by about 3.9. Each °F of afternoon heat multiplies the odds by 1.17. A hot, humid monsoon afternoon goes with higher risk, and it shows up hours before the storm. Whether that holds up as a forecast is the next question.

Model Evaluation

```
train_prob <- predict(logit_model, train_data, type = "response")
test_prob  <- predict(logit_model, test_data, type = "response")
roc_train <- roc(train_data$big_night, train_prob, quiet = TRUE)
roc_test <- roc(test_data$big_night, test_prob, quiet = TRUE)
roc_full <- roc(test_data$big_night, predict(full_model, test_data, type = "response"), quiet =
        TRUE)
calendar_model <- glm(big_night ~ season_day + I(season_day^2), data = train_data, family =
        "binomial")
roc_calendar <- roc(test_data$big_night, predict(calendar_model, test_data, type = "response"),
        quiet = TRUE)

# Two cutoffs, both set on the training years and applied unchanged to the test years:
#  - Balanced (Youden's J): the point that best balances catching big nights against false alarms
#  - Operating budget: flag the riskiest 20% of nights, about one evening in five
youden_cutoff <- coords(roc_train, "best", best.method = "youden", ret = "threshold")$threshold[1]
cutoff <- as.numeric(quantile(train_prob, 0.80))
cutoff_compare <- tibble(Cutoff = c("Balanced (Youden)", "Operating budget (top 20%)"),
                         Probability = round(c(youden_cutoff, cutoff), 3)) %>%
  rowwise() %>%
  mutate(`Test nights flagged` = sum(test_prob > Probability),
         `Big nights caught` = sum(test_prob > Probability & test_data$big_night == 1),
         `False alarms` = sum(test_prob > Probability & test_data$big_night == 0),
         Precision = pct(`Big nights caught` / `Test nights flagged`)) %>% ungroup()
kable(cutoff_compare, caption = "Cutoff choice, test years 2024-2025")
```

Cutoff choice, test years 2024-2025

Cutoff	Probability	Test nights flagged	Big nights caught	False alarms	Precision
Balanced (Youden)	0.044	116	10	106	9%
Operating budget (top 20%)	0.092	52	8	44	15%

```
predicted_classes <- ifelse(test_prob > cutoff, 1, 0)
conf_matrix <- confusionMatrix(factor(predicted_classes, levels = 0:1),
        factor(test_data$big_night, levels = 0:1),
                               positive = "1")
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 160   2
##          1  44   8
##
##                Accuracy : 0.785
##                  95% CI : (0.7239, 0.8381)
##     No Information Rate : 0.9533
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.195
##
##  Mcnemar's Test P-Value : 1.493e-09
##
##             Sensitivity : 0.80000
##             Specificity : 0.78431
##          Pos Pred Value : 0.15385
##          Neg Pred Value : 0.98765
##              Prevalence : 0.04673
##          Detection Rate : 0.03738
##    Detection Prevalence : 0.24299
##       Balanced Accuracy : 0.79216
##
##        'Positive' Class : 1
##
```

```
auc_ci <- ci.auc(roc_test)
auc_ci
```

```
## 95% CI: 0.8026-0.9797 (DeLong)
```

```
roc.test(roc_test, roc_calendar)
```

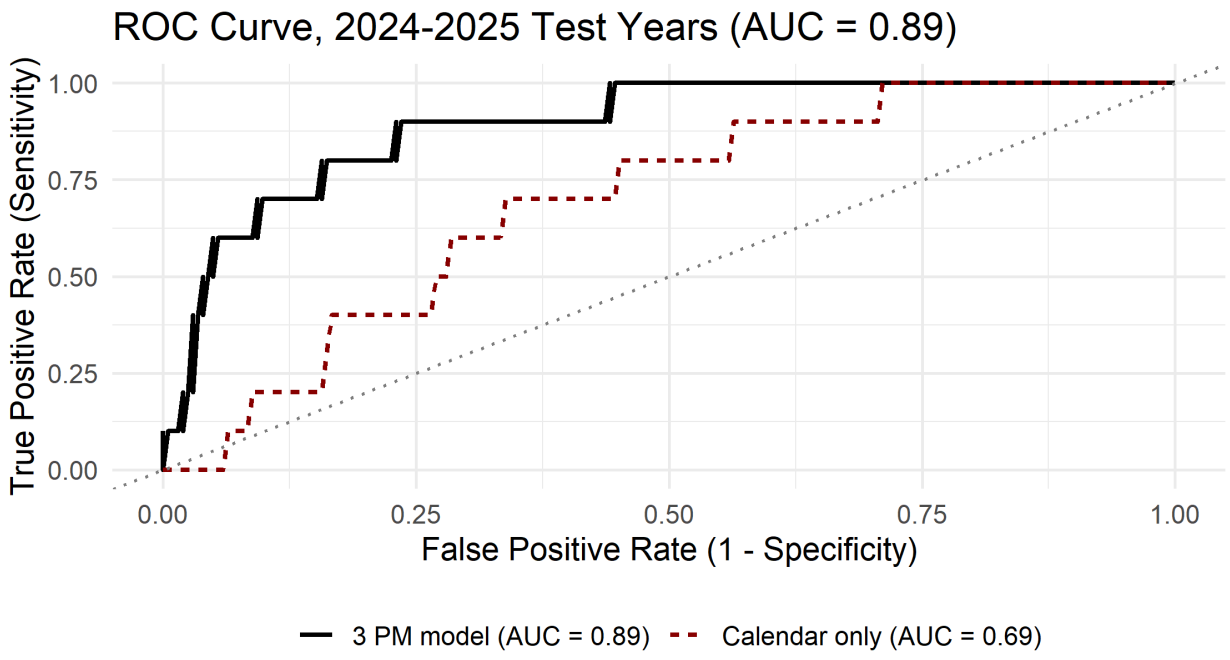
```
##
##  DeLong's test for two correlated ROC curves
##
## data:  roc_test and roc_calendar
## Z = 2.0822, p-value = 0.03733
## alternative hypothesis: true difference in AUC is not equal to 0
## 95 percent confidence interval:
##  0.01165227 0.38540655
## sample estimates:
## AUC of roc1 AUC of roc2
##   0.8911765   0.6926471
```

```
c(two_input_auc = round(auc(roc_test), 3), full_model_auc = round(auc(roc_full), 3))
```

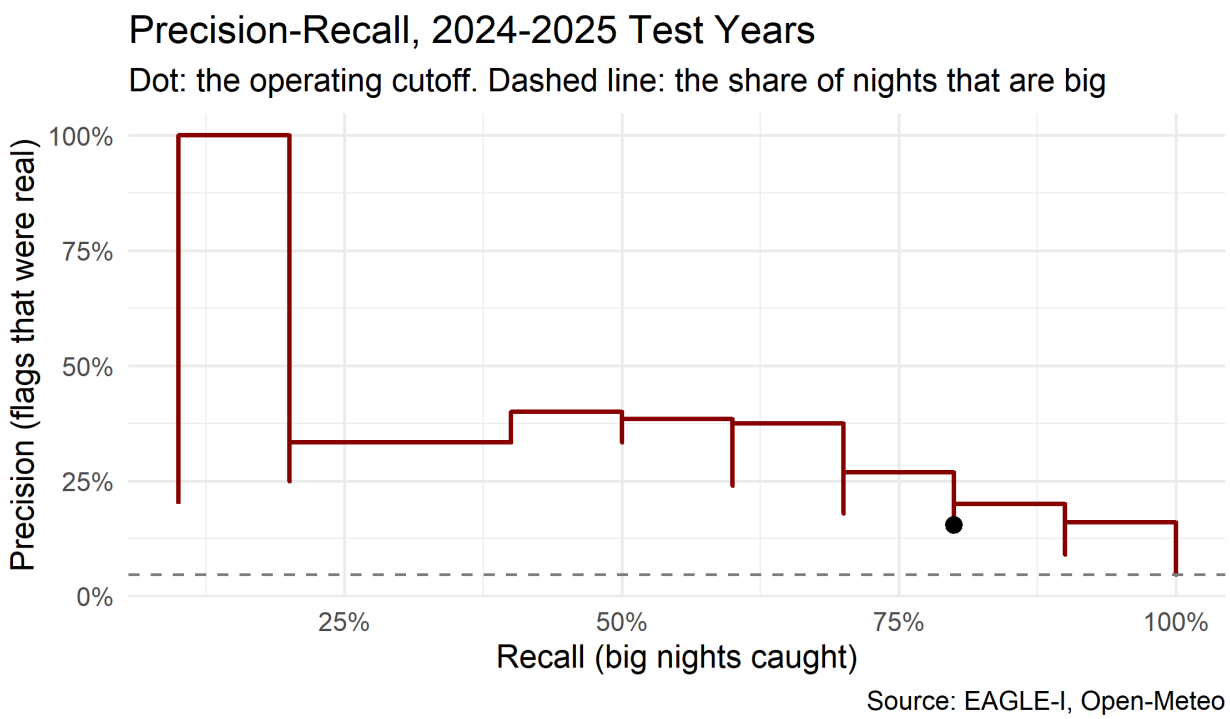
```
##  two_input_auc full_model_auc
##          0.891          0.892
```

```r
plot_data <- bind_rows(
  tibble(model = paste0("3 PM model (AUC = ", f2(auc(roc_test)), ")"),
         FPR = 1 - roc_test$specificities, TPR = roc_test$sensitivities),
  tibble(model = paste0("Calendar only (AUC = ", f2(auc(roc_calendar)), ")"),
         FPR = 1 - roc_calendar$specificities, TPR = roc_calendar$sensitivities))

ggplot(plot_data, aes(x = FPR, y = TPR, color = model, linetype = model)) +
  geom_line(linewidth = 1) +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted", color = "grey50") +
  scale_color_manual(values = c("black", "darkred")) +
  labs(
    title = paste0("ROC Curve, 2024-2025 Test Years (AUC = ", f2(auc(roc_test)), ")"),
    x = "False Positive Rate (1 - Specificity)", y = "True Positive Rate (Sensitivity)", color =
        NULL, linetype = NULL,
    caption = "Source: EAGLE-I, Open-Meteo"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom")
```



```r
# Precision-recall: of the nights flagged, how many were really big?
pr <- coords(roc_test, "all", ret = c("threshold", "recall", "precision"), transpose = FALSE) %>%
  filter(!is.na(precision))
ggplot(pr, aes(recall, precision)) +
  geom_step(color = "darkred", linewidth = 1) +
  geom_hline(yintercept = mean(test_data$big_night), linetype = "dashed", color = "grey50") +
  annotate("point", x = conf_matrix$byClass["Sensitivity"], y = conf_matrix$byClass["Pos Pred
        Value"], size = 3) +
  scale_x_continuous(labels = scales::percent) + scale_y_continuous(labels = scales::percent) +
  labs(title = "Precision-Recall, 2024-2025 Test Years", subtitle = "Dot: the operating cutoff.
        Dashed line: the share of nights that are big",
       x = "Recall (big nights caught)", y = "Precision (flags that were real)", caption =
        "Source: EAGLE-I, Open-Meteo") +
  theme_minimal(base_size = 14)
```



Does It Hold Up in Other Years?

One train-test split with 10 big nights is a thin test. So the same two-input model is also rebuilt each year on everything before it and tested on that year alone (rolling-origin validation), with the cutoff set the same way each time.

```
rolling <- map_dfr(2022:2025, function(y) {
  tr <- model_data %>% filter(year < y); te <- model_data %>% filter(year == y)
  fit <- glm(big_night ~ noon_dew + high_by_3pm, data = tr, family = "binomial")
  p <- predict(fit, te, type = "response"); cut_y <- quantile(predict(fit, tr, type = "response"),
         0.80)
  tibble(`Test year` = y, `Trained on` = paste0("2020-", y - 1), `Big nights` = sum(te$big_night),
         AUC = if (sum(te$big_night) > 0) round(as.numeric(auc(roc(te$big_night, p, quiet =
         TRUE))), 2) else NA,
         Flagged = sum(p > cut_y), Caught = sum(p > cut_y & te$big_night == 1))
}) %>% mutate(Precision = ifelse(Flagged > 0, pct(Caught / Flagged), "-"))
kable(rolling, caption = "Rolling-origin validation of the two-input model")
```

Rolling-origin validation of the two-input model

Test year	Trained on	Big nights	AUC	Flagged	Caught	Precision
2022	2020-2021	5	0.50	21	0	0%
2023	2020-2022	8	0.77	19	2	11%
2024	2020-2023	10	0.85	39	8	21%
2025	2020-2024	0	NA	9	0	0%

Findings: The two-input model scored an AUC of **0.89** on the 2024-2025 test years (95% interval 0.80 to 0.98), against 0.69 for a model that only knows the date. The interval is wide because big nights are rare: only 10 in two test seasons, after removing the 4 days when a big outage was still under way at 3 PM. The difference from the date-only model ($p = 0.037$) is statistically significant. The full seven-input model scored 0.89 on the same test years, so the extra inputs didn't add much. The rolling check is less flattering. Trained on fewer years, the model scored 0.50 in 2022, which is no better than chance, 0.77 in 2023 and 0.85 in 2024, and 2025 had no big outage nights at all, so it can't be scored and every flag that year was a false alarm. So the 0.89 is the best case, earned on 2024's 10 big nights, and the honest read is that the signal looks real but isn't proven yet.

At the operating cutoff:

- **Sensitivity (big nights caught): 80%**, 8 of the 10 big outage nights.
- **Specificity (quiet nights left alone): 78%**, 160 of 204.
- **Precision (flags that were real): 15%**. The flag went up 52 times and 44 of those were false alarms, about 6.5 flags for every big night caught.

Plain accuracy is misleading here: only 5% of test nights were big, so a model that always says "no" would be 95% accurate and catch nothing. The balanced cutoff caught 10 big nights but flagged 116 of 214 nights. The operating cutoff flags about one evening in five. Whether that trade is worth it depends on what a false alarm costs (crews held late, overtime) against what an early start on a big night is worth, and neither is in public data.

9. Model 4: Decision Tree - An Interpretable Alternative

How It Works

- A decision tree asks a series of yes/no questions ("Is noon dew point above 55°F?") and sorts each night into a leaf. Each split is chosen to make the leaves as pure as possible, measured by the **Gini index**.
- Trees grow until they memorize the training data, so they are **pruned**: cross-validation tests each tree size on data it didn't train on, and the complexity parameter (cp) with the lowest cross-validated error wins. Random folds would mix nights from the same storm stretch across training and testing, the same problem as a random train-test split, so the folds are **whole seasons**: each training year is held out in turn.
- Big nights are rare, so the tree is told to weigh both outcomes equally (equal **priors**); otherwise its best guess would always be "no". That also means the percentages shown in its leaves are weighted, not real-world probabilities.
- The tree learns its own rules from the same inputs. It is an alternative classifier that is easy to read, not a translation of the logistic model's decisions.

Training the Tree

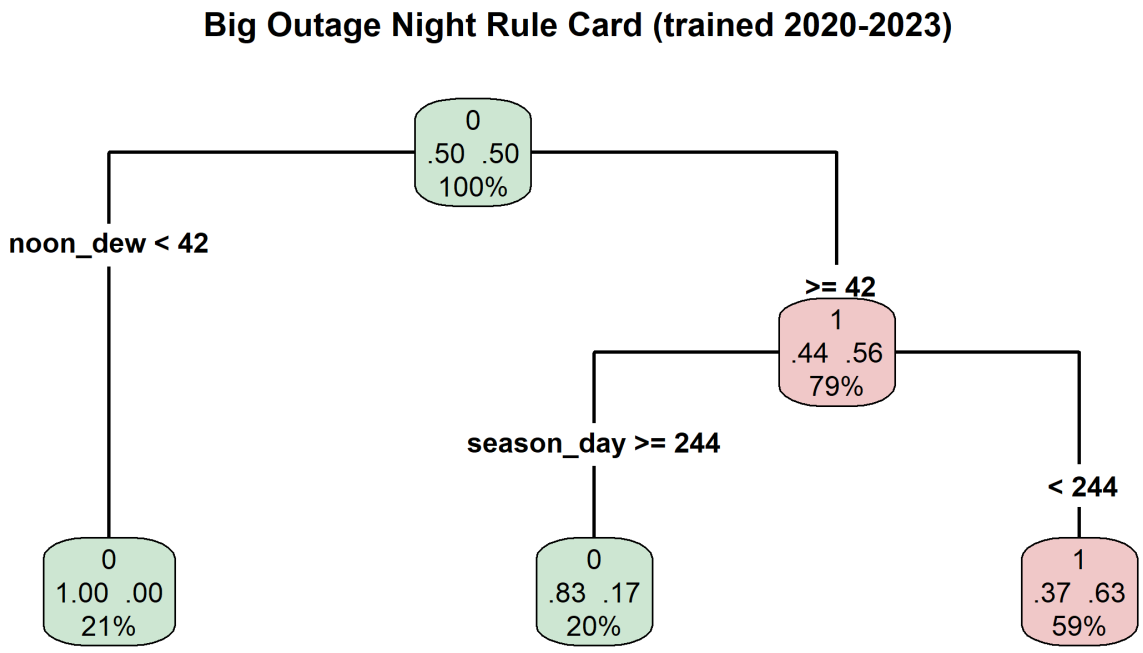
```
set.seed(123) # for reproducibility
season_folds <- as.integer(factor(train_data$year))   # cross-validation folds are whole seasons,
        not random nights
tree_inputs <- big_night ~ noon_dew + high_by_3pm + gust_by_3pm + rain_by_3pm + prev_day_dew +
        log_prev_night_peak + season_day

# First, a tree grown with no limits, then pruned by cross-validation (random folds and whole-
        season folds)
best_free <- rpart(tree_inputs, data = train_data %>% mutate(big_night = factor(big_night)),
        method = "class",
                  parms = list(prior = c(0.5, 0.5)), control = rpart.control(cp = 0, minsplit =
        2, xval = season_folds))
prune_by <- function(folds) {
  t <- rpart(tree_inputs, data = train_data %>% mutate(big_night = factor(big_night)), method =
        "class",
            parms = list(prior = c(0.5, 0.5)), control = rpart.control(cp = 0.001, xval = folds))
  prune(t, cp = t$cptable[which.min(t$cptable[, "xerror"]), "CP"])
}
random_free <- prune_by(10)              # random 10-fold
season_free <- prune_by(season_folds)    # whole seasons as folds
c(unpruned_leaves = sum(best_free$frame$var == "<leaf>"), random_fold_leaves =
        sum(random_free$frame$var == "<leaf>"),
  season_fold_leaves = sum(season_free$frame$var == "<leaf>"))
```

```
##    unpruned_leaves random_fold_leaves season_fold_leaves
##                 41                  3                  3
```

```r
# Then a rule card: at most three questions deep, at least 20 nights per leaf
rule_tree <- rpart(tree_inputs, data = train_data %>% mutate(big_night = factor(big_night)),
         method = "class",
                  parms = list(prior = c(0.5, 0.5)),
                  control = rpart.control(maxdepth = 3, minbucket = 20, cp = 0.001, xval =
         season_folds))
rule_tree <- prune(rule_tree, cp = rule_tree$cptable[which.min(rule_tree$cptable[, "xerror"]),
         "CP"])

rpart.plot(rule_tree, type = 4, extra = 104, box.palette = c("#CFE6D3", "#F1C9C9"), fallen.leaves
         = TRUE, varlen = 0,
           main = "Big Outage Night Rule Card (trained 2020-2023)", cex = 1.0, branch.lwd = 2)
```




```r
# The same tree as plain-language rules
rpart.rules(rule_tree, cover = TRUE)
```

```
##  big_night                                                cover
##       0.00 when noon_dew <  42                             21%
##       0.17 when noon_dew >= 42 & season_day >= 244         20%
##       0.63 when noon_dew >= 42 & season_day <  244         59%
```

`season_day` is the day of the year counted on a 365-day calendar (February 29 is skipped), so 244 is September 1 in every year, leap years included. The tree's split values are read from the fitted model in the text below, so the rule stays in sync if the data changes. Each leaf shows the predicted class, the weighted chance of a big night and the share of training nights that land there.

Tree vs. Logistic Regression

```
tree_prob <- predict(rule_tree, test_data, type = "prob")[, "1"]
roc_tree <- roc(test_data$big_night, tree_prob, quiet = TRUE)
tree_classes <- predict(rule_tree, test_data, type = "class")
roc_free <- roc(test_data$big_night, predict(best_free, test_data, type = "prob")[, "1"], quiet =
        TRUE)

comparison <- tibble(
  Model = c("Logistic regression (Model 3)", "Rule-card tree (Model 4)", "Unrestricted tree, no
        pruning", "Calendar only"),
  `Train AUC` = round(c(auc(roc_train),
                        auc(roc(train_data$big_night, predict(rule_tree, train_data, type =
        "prob")[, "1"], quiet = TRUE)),
                        auc(roc(train_data$big_night, predict(best_free, train_data, type =
        "prob")[, "1"], quiet = TRUE)),
                        auc(roc(train_data$big_night, fitted(calendar_model), quiet = TRUE))), 2),
  `Test AUC` = round(c(auc(roc_test), auc(roc_tree), auc(roc_free), auc(roc_calendar)), 2),
  `Share of test nights flagged` = c(pct(mean(predicted_classes)), pct(mean(tree_classes == "1")),
        "-", "-"))
kable(comparison, caption = "Train vs. test AUC, and how often each model flags")
```

Train vs. test AUC, and how often each model flags

Model	Train AUC	Test AUC	Share of test nights flagged
Logistic regression (Model 3)	0.74	0.89	24%
Rule-card tree (Model 4)	0.70	0.71	60%
Unrestricted tree, no pruning	1.00	0.48	-
Calendar only	0.64	0.69	-

Finding: The unrestricted tree is a lesson in overfitting. Grown with no limits, it splits until almost every leaf is pure: 41 leaves, many of them a rule built on one or two of the 24 big nights in the training seasons. It scores 1.00 AUC on the years it learned from and 0.48 on the years it didn't: it memorized 2020-2023 instead of learning what makes a big night. Pruning fixes that. With whole seasons as the cross-validation folds, the tree shrinks to 3 leaves, the same tree as the rule card below. Random 10-fold pruning lands on 3 leaves here, so the choice of folds didn't change the answer on this data, but season folds are the version that matches how the model would be used.

Limited to three questions with at least 20 nights per leaf, the tree is a 3-leaf rule card that a shift supervisor can read, scoring 0.71 against 0.89 for logistic regression. It is also a much broader rule: it flags 60% of test nights, against 24% for the logistic model at its operating cutoff. So the two models make different decisions: logistic regression ranks the nights and supports a tighter flag, and the tree trades accuracy for rules anyone can repeat.

10. Recommendations

What Would Improvements Be Worth?

Using the cost model from Section 5, we can estimate what four hypothetical improvements would be worth to customers. These are benefit scenarios under the cost model's assumptions, not forecasts of what a utility would save, and none of them include what the improvement would cost to achieve.

1. **Prevention:** 10% fewer customers interrupted per storm (hardening, tree trimming, more automated switches).
2. **Faster restoration:** every outage restored 10% faster.
3. **Night outages at daytime pace:** if the whole night difference in average outage length could be closed.
4. **Half the night difference:** a more cautious version, since some of the difference is probably not fixable.

The night scenarios need care. Model 2 measures restoration as the hours until the county count falls to 10% of the peak, but the cost model prices the **average outage length** (customer-minutes divided by peak). Those are related but not the same: a few hundred customers left out for hours add customer-minutes after the 10% point is reached. So before pricing a night scenario, the two measures are compared, and the night difference is re-estimated on the cost model's own measure.

```
# How closely do the two duration measures move together? (events used in Model 2)
link <- lm(log(avg_minutes_out) ~ log(restore_hrs), data = events_m2)
c(correlation = round(cor(log(events_m2$restore_hrs), log(events_m2$avg_minutes_out)), 2),
  elasticity = round(unname(coef(link)[2]), 2))
```

```
## correlation  elasticity
##        0.74        0.62
```

```
# The night difference measured directly on average outage length, all events with weather
events_len <- events %>% filter(!is.na(storm_gust)) %>% mutate(storm_rain_tenths = storm_rain *
        10)
model2_len <- lm(log(avg_minutes_out) ~ log(peak) + night + back_to_back + weekend + monsoon +
        storm_gust + storm_rain_tenths,
                data = events_len)
night_len_ci <- (exp(confint(model2_len)["night", ]) - 1) * 100
c(night_pct_avg_length = unname(round((exp(coef(model2_len)["night"]) - 1) * 100)), low =
        unname(round(night_len_ci[1])),
  high = unname(round(night_len_ci[2])), events = nobs(model2_len))
```

```
## night_pct_avg_length                  low                 high
##                   39                   20                   61
##               events
##                  596
```

```
night_factor <- exp(coef(model2_len)["night"])     # the scenarios use the night difference in
        average outage length
link_cor <- cor(log(events_m2$restore_hrs), log(events_m2$avg_minutes_out)); link_el <- coef(link)
        [2]
night_len <- round((exp(coef(model2_len)["night"]) - 1) * 100)
night_ev <- events %>% filter(night == 1)
savings <- function(new_cost) sum(new_cost) / yrs_cost

scenarios <- tibble(
  Scenario = c("1. 10% fewer customers interrupted", "2. All outages restored 10% faster",
               "3. Night outages at daytime pace", "4. Half the night difference closed"),
  value = c(
    savings(events$cost * 0.10),
    savings(events$cost - events$peak * cost_per_customer(events$avg_minutes_out * 0.9)),
    savings(night_ev$cost - night_ev$peak * cost_per_customer(night_ev$avg_minutes_out /
        night_factor)),
    savings(night_ev$cost - night_ev$peak * cost_per_customer(night_ev$avg_minutes_out /
        sqrt(night_factor)))
  )) %>%
  mutate(`Annual benefit` = dollars(value))
kable(scenarios %>% select(-value), caption = "Hypothetical annual benefit to Pima County
        customers (2025 dollars, base cost assumptions)")
```

Hypothetical annual benefit to Pima County customers (2025 dollars, base cost assumptions)

Scenario	Annual benefit
1. 10% fewer customers interrupted	$7,289,618
2. All outages restored 10% faster	$5,529,806
3. Night outages at daytime pace	$8,717,610
4. Half the night difference closed	$4,676,575

Finding: The two duration measures move together (correlation 0.74 on the log scale), but not one for one: each 10% longer time to the 10% point goes with about 6.2% longer average outage. Measured on average outage length, night outages run 39% longer (95% interval 20% to 61%), smaller than the 61.2% from Model 2. Scenarios 3 and 4 use the 39% figure, so they price the night difference on the same measure the cost model uses. They still assume the whole difference, or half of it, could actually be closed.

Backtesting the 3 PM Flag

```
flagged <- test_data %>% mutate(flag = test_prob > cutoff)
backtest <- tibble(
  `Monsoon nights, 2024-2025` = nrow(flagged),
  `Nights flagged` = sum(flagged$flag),
  `Big nights caught` = sum(flagged$flag & flagged$big_night == 1),
  `Big nights missed` = sum(!flagged$flag & flagged$big_night == 1),
  `False alarms` = sum(flagged$flag & flagged$big_night == 0))
kable(backtest)
```

Monsoon nights, 2024-2025	Nights flagged	Big nights caught	Big nights missed	False alarms
214	52	8	2	44

```
half_gap_saving <- function(ev) ev$cost - ev$peak * cost_per_customer(ev$avg_minutes_out /
        sqrt(night_factor))
test_night_ev <- night_ev %>% filter(year(storm_date) %in% 2024:2025) %>%
  mutate(flagged = storm_date %in% flagged$day[flagged$flag])
targeting <- tibble(
  `Night outages, 2024-2025` = c("All night outages (year-round)", "After a flagged afternoon"),
  Events = c(nrow(test_night_ev), sum(test_night_ev$flagged)),
  `Half-difference benefit per year` = c(sum(half_gap_saving(test_night_ev)),
                                         sum(half_gap_saving(test_night_ev[test_night_ev$flagged,
        ]))) / 2) %>%
  mutate(Share = pct(`Half-difference benefit per year` / `Half-difference benefit per year`[1]),
        `Half-difference benefit per year` = dollars(`Half-difference benefit per year`))
kable(targeting, caption = "How much of the night scenario follows a flagged afternoon, test
        seasons")
```

How much of the night scenario follows a flagged afternoon, test seasons

Night outages, 2024-2025	Events	Half-difference benefit per year	Share
All night outages (year-round)	36	$3,129,983	100%
After a flagged afternoon	7	$2,583,102	83%

Recommendations (for a utility's distribution operations and grid planning teams):

1. **Look at prevention before across-the-board speed.** Under the cost model, a 10% cut in customers interrupted is worth about $7,289,618 a year to customers, against $5,529,806 for restoring every outage 10% faster. That follows from the shape of the cost curve: cost grows slowly with outage length but in direct proportion to the number of customers hit. It suggests where to look first, not which project to fund, since the cost of achieving either is outside this analysis.
2. **Investigate the night difference.** Outages that peak at night are associated with 61.2% longer restoration (39% longer average outage) and carry 47% of modeled cost. If half the difference in average outage length could be closed, it would be worth about $4,676,575 a year. The next step is finding out why night restoration is slower, using crew schedules and outage records only the utility has.
3. **Treat the 3 PM flag as a promising but unstable signal, not a trigger.** In the 2024-2025 test seasons it went up on 24% of monsoon nights and caught 8 of the 10 big outage nights, but 85% of its alerts were false alarms, and in earlier years it did worse. It needs more seasons before anyone schedules around it. Night outages after a flagged afternoon held 83% of the half-difference benefit in those years, so if late crews do turn out to help, the flag shows which evenings to try first.
4. **Keep the rule card as a plain-language companion.** Noon dew point at least 42°F before September 1 is easy to repeat, but it flags far more nights than the logistic model. Refit both

each spring with the latest season.

11. Conclusion & Reflection

This analysis put a modeled price on Pima County's power outages (about $72,896,178 a year as a central estimate, 68% of it in monsoon season) and used four models to look at where it comes from. Storm-night outage severity rises with wind, rain, heat and humidity (Model 1), though weather at one point explains only a small share of it. Outages that peak at night take longer to restore (Model 2), for reasons the data can't pin down. A humid, hot afternoon is a promising but unstable signal for big outage nights: it scored a test AUC of 0.89 on 2024-2025 (Model 3), but only 0.50 to 0.85 in the rolling check and with several false alarms per catch, and a short rule card gives a readable but broader alternative (Model 4).

The hardest part to defend wasn't a model. It was the inputs: deciding whether a missing reading meant zero, how to measure restoration from county totals, how many large businesses TEP serves, and whether two ways of measuring outage length can stand in for each other. Each of those turned out to matter more than the choice between algorithms.

Limitations:

- **County, not utility:** EAGLE-I reports Pima County as a whole, which includes Trico Electric Cooperative's customers alongside TEP's.
- **Missing readings:** a missing interval can be zero or a reporting gap. Flagged gaps change total outage time by 17.7%, but they matter more for restoration timing.
- **One weather point:** weather comes from one grid point in central Tucson, which is why Model 1 explains only 11% of the variation.
- **Few big nights:** only 34 qualified in six monsoon seasons, and none in 2025. The test AUC interval (0.80 to 0.98) is wide, and the rolling check ranged from 0.50 to 0.85.
- **Associations, not causes:** none of the models show that changing an input would change the outcome, including the night restoration difference.
- **Cost rests on assumptions:** the business mix moves it the most, and the large-account count is an estimate from LBNL's average sizes, not a TEP figure. The one-at-a-time range is not a confidence interval. The LBNL tables pool surveys from 1989-2012, adjusted for inflation.
- **3 PM is reanalysis weather, not a forecast:** ERA5 is rebuilt after the fact, so the model is a historical feasibility test, not a test with the data a utility had at 3 PM on those days. Open-Meteo's archive of day-ahead model runs (its Previous Runs API) starts in January 2024 for most models, too short to train on.

Next Steps & Implications

- **Better cost inputs:** LBNL's ICE Calculator 2.2 (February 2026) has newer survey data, and TEP's count of business customers by size would replace the energy-balance estimate.

- **Add forecasts:** Open-Meteo's Historical Forecast API holds archived short-range forecasts, including CAPE (storm energy) and gusts, from 2018 for NOAA's HRRR model and March 2021 for GFS. Retraining on those would test the flag with information a utility actually had at 3 PM.
- **Go below the county:** circuit-level outage and crew data, which only the utility holds, would test the night difference directly and rank which feeders to harden.
- **Operational use:** even with its limits, the analysis gives a utility a mid-afternoon risk signal and a structured way to compare prevention and restoration, both of which get stronger with the utility's own data.

12. References

- Brelsford, C., Tennille, S., Myers, A., et al. (2024). A dataset of recorded electricity outages by United States county 2014-2022. *Scientific Data* 11, 271 (method paper; validates 2014-2022).
- Oak Ridge National Laboratory. *EAGLE-I Recorded Electricity Outages*, figshare record 24237376, CC BY 4.0. Yearly files used: 2020 (file 42547894), 2021 (42547891), 2022 (42547897), 2023 (44574907), 2024 (53581661), 2025 (62164877). The 2023-2025 files were added after the method paper.
- NOAA National Centers for Environmental Information. *Storm Events Database*, details files 2020-2025.
- Open-Meteo. *Historical Weather API* (ERA5 reanalysis), central Tucson (32.22 N, 110.97 W). Forecast archive dates from Open-Meteo's *Historical Forecast API* and *Previous Runs API* documentation, checked November 2026.
- U.S. Energy Information Administration. *Form EIA-861M* (sales to ultimate customers, 2025) and *Form EIA-861* (reliability), Tucson Electric Power.
- Sullivan, M., Schellenberg, J., and Blundell, M. (2015). *Updated Value of Service Reliability Estimates for Electric Utility Customers in the United States*. Lawrence Berkeley National Laboratory, LBNL-6941E, Tables ES-1 and ES-2.
- Lawrence Berkeley National Laboratory and Resource Innovations (2026). *ICE Calculator 2.0, Phase 1 and 2 Final Report* (ICE Calculator version 2.2). OSTI 3021993.
- U.S. Bureau of Labor Statistics. *Consumer Price Index for All Urban Consumers (CPI-U)*, series CUUR0000SA0.
- Logit Regression. UCLA: Statistical Consulting Group. https://stats.oarc.ucla.edu/r/dae/logit-regression/
- Classification: ROC and AUC. Google Machine Learning Crash Course. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc