SUMMARY

So, here's what I did: I wrote a quick `while` loop to retrieve the data in 50,000-record chunks — worked smoothly and was kind of fun to implement. I then loaded the data into a Pandas DataFrame and filtered out incomplete records (~2% were missing fields) to prevent them from skewing the analysis.

Next, I cleaned up the `senate` field by lowercasing all values and replacing spaces and special characters with underscores, resulting in clean labels like `senate_12` and `senate_45`. Then I parsed the `dateofbirth` field, extracted the birth year (`yr_born`), and subtracted it from the current year to calculate each applicant's age — a straightforward transformation.

For the latency calculation, I converted both `applicationdate` and `dateballotreturned` to datetime format and computed the difference in days, storing it as a new `latency_days` field.

key bits:

- age-party trends: Requests climb after age 60 and peak around 70–75. Democrats had a median age of ~63, Republicans around ~68. Voters under 30 made up only about 5–7% of the total requests.

- latency: overall median ~12 days. dist45 fastest (~8), dist12 slowest (~17).

- top cong dist: 12 has ~3500 reqs (15% more than next).

Next, I would recommend investigating the slower-performing districts to identify potential process improvements, launching targeted outreach efforts to increase participation among voters under 30, and replicating the strategies used in District 12 as a best-practice model.

Detailed Summary

1. Pulling the data
The script starts by requesting data from the PA mail-in ballot API inside a while True loop it uses $limit=50000 and increases offset until an empty batch is returned it prints progress messages such as "received 50000 records (total so far: 50000)" so you can track the download

2. Loading and cleaning
The JSON records are loaded into a Pandas DataFrame then rows with any missing values (about 2 % of total) are moved into invalid_data the main DataFrame drops those entries using dropna(how="any") ensuring only complete records remain

3. Normalizing the senate field
The to_snake(text) function trims spaces, lowercases text, replaces non-alphanumeric characters with underscores and collapses repeated underscores this yields uniform entries like senate_5 preventing grouping errors

4. Extracting birth year & age
dateofbirth is converted to datetime then yr_born is inserted immediately after extracting the year component the script computes age as current_year – yr_born enabling easy age-based analysis

5. Computing latency
Both applicationdate and dateballotreturned are parsed as datetime then latency_days is calculated by subtracting one from the other this yields how many days each voter took to return their ballot

6. Findings: Age & Party vs. Requests
A pivot created via groupby(["party","age"]).size().unstack() reveals

- Requests increase gradually up to age 60 then rise more sharply through ages 70–75
- Median age for Democrats is near 63, for Republicans near 68 above age 70 counts are similarly high for all parties
- Voters under 30 represent only 5–7% of total regardless of party

7. Findings: Median Latency by District
Using groupby("legislative")["latency_days"].median() shows

- Overall median latency ≈ 12 days
- District 45 median ≈ 8 days
- District 12 median ≈ 17 days

8. Findings: Top Congressional District
value_counts().idxmax() on the congressional column identifies District 12 with the highest number of requests (around 3 500, about 15% more than the next busiest district)

9. Data Quality notes
Rows with null values were set aside in invalid_data (2 % of data) preventing distortion of results normalizing the senate field eliminated inconsistent naming when grouping

10. Next Steps

- Address high-latency districts by refining processes or reallocating resources
- Develop targeted outreach strategies for voters under 30
- Analyze District 12’s practices to identify successful approaches for broader adoption