# Chicago Voter Turnout and Demographic Analysis (2008–2024)

Group 37: Rajat Kanti Paul and Sakkhi Raheel

2026-03-09

## Table of contents

## Overview

This document carries forward and extends the data collection and cleaning work completed in `project.qmd`. It implements the full analytical pipeline described in the Final Project Workflow:

1. **Parts 1–3 (Already Completed):** ACS 5-year data download, post-processing, and voter turnout file cleaning — reproduced here for documentation.
2. **Part 2 – Geographic Foundation:** Load Chicago city boundary and 2020 census tract shapefile; filter to Chicago tracts.
3. **Part 3 – ACS → Census Tracts:** Merge ACS demographic data onto the tract shapefile.
4. **Part 4 – Precinct Geometry:** Load the correct precinct boundary file for each election year and create a standardized precinct ID.
5. **Part 5 – Spatial Overlay:** Area-weighted interpolation of tract-level demographics onto precinct polygons.
6. **Part 6 – Merge Turnout + Demographics:** Join precinct turnout to the interpolated demographics.
7. **Part 7 – Master Dataset:** Stack all five election years into one panel dataset and save to `data/master_panel.csv`.
8. **Part 8 – Analysis:** Correlation analysis, scatter plots, binned quintile plots, time-trend plots, and choropleth maps.
9. **Part 9 – Policy Narrative:** Interpret findings and connect to policy.

---

## Setup

```
import os
import time
import requests
import pandas as pd
import numpy as np
```

```
import geopandas as gpd
import altair as alt
import seaborn as sns
from pathlib import Path
import warnings
import tempfile
from IPython.display import Image
import vl_convert as vlc

warnings.filterwarnings("ignore")
alt.renderers.enable("png")
alt.data_transformers.disable_max_rows()


def display_altair_png(chart, scale=2):
    png_bytes = vlc.vegalite_to_png(chart.to_dict(), scale=scale)
    with tempfile.NamedTemporaryFile(suffix=".png") as tmp:
        tmp.write(png_bytes)
        tmp.flush()
        display(Image(filename=tmp.name))

# Paths
ROOT         = Path(".")
DATA_ACS     = ROOT / "data" / "raw" / "acs"
DATA_VOTER   = ROOT / "Data - Voter turnout and registration(chicagoelections.gov)"
PRECINCT_DIR = ROOT / "Chicago precincts shapefiles"
TRACT_SHP    = ROOT / "Census Tract Shapefile (ACS Merging)" / "tl_2020_17_tract.shp"
CITY_BNDRY   = ROOT / "Boundaries_-_City_20260206.geojson"

# Coordinate Reference Systems
# UTM Zone 16N (EPSG:26916) - metre-based, area-accurate for Chicago
CRS_PROJ = "EPSG:26916"
CRS_GEO  = "EPSG:4326"

# Election-year → boundary/ACS file mapping
PRECINCT_FILES = {
    2008: "Precincts (-2010)(2008 and 2012 elections).geojson",
    2012: "Precincts (-2010)(2008 and 2012 elections).geojson",
    2016: "Precincts(2013-2022) (2016 and 2020 elections).geojson",
    2020: "Precincts(2013-2022) (2016 and 2020 elections).geojson",
    2024: "Precincts(2023-) (2024 elections).geojson",
}
```

```
ACS_FILES = {
    2008: "acs_2010_for_election_2008.csv",
    2012: "acs_2014_for_election_2012.csv",
    2016: "acs_2018_for_election_2016.csv",
    2020: "acs_2020_for_election_2020.csv",
    2024: "acs_2024_for_election_2024.csv",
}

DEMO_COLS      = ["median_hh_income", "pct_college", "pct_black",
                  "pct_hispanic", "pct_renter", "pct_18_29"]
ELECTION_YEARS = [2008, 2012, 2016, 2020, 2024]

print("Setup complete.")
```

```
Setup complete.
```

---

## Part 1 – Data Collection (Already Completed)

The cells below reproduce the original data-collection and cleaning scripts from `project.qmd`. They are set to `eval: false` because the output files already exist on disk; they are retained here for full reproducibility.

### 1.1 ACS 5-Year Data Download via Census API

```
# Illinois / Cook County identifiers
STATE_FIPS  = "17"
COUNTY_FIPS = "031"

CENSUS_API_KEY = os.getenv("CENSUS_API_KEY", "").strip()

ELECTION_TO_ACS_API_YEAR = {
    2008: 2010,   # ACS 2006-2010 5-year
    2012: 2014,   # ACS 2010-2014
    2016: 2018,   # ACS 2014-2018
    2020: 2020,   # ACS 2016-2020
    2024: 2024,   # ACS 2020-2024
```

```
}

OUTDIR = "data/raw/acs"
os.makedirs(OUTDIR, exist_ok=True)


def _request_json(url: str, params: dict, timeout: int = 60) -> list:
    r = requests.get(url, params=params, timeout=timeout)
    if r.status_code != 200:
        raise RuntimeError(
            f"HTTP {r.status_code}\nURL: {r.url}\nResponse: {r.text[:800]}"
        )
    data = r.json()
    if not isinstance(data, list) or len(data) < 2:
        raise RuntimeError(
            f"Unexpected API response.\nURL: {r.url}\nGot: {data}"
        )
    return data


def census_get_group(year: int, group: str) -> pd.DataFrame:
    """Download a full ACS 5-year group table for all Cook County tracts."""
    base   = f"https://api.census.gov/data/{year}/acs/acs5"
    params = {
        "get": f"group({group})",
        "for": "tract:*",
        "in":  f"state:{STATE_FIPS} county:{COUNTY_FIPS}",
    }
    if CENSUS_API_KEY:
        params["key"] = CENSUS_API_KEY
    data = _request_json(base, params=params)
    df   = pd.DataFrame(data[1:], columns=data[0])
    df["GEOID"] = df["state"] + df["county"] + df["tract"]
    return df


def to_num(df: pd.DataFrame, cols: list) -> None:
    for c in cols:
        df[c] = pd.to_numeric(df[c], errors="coerce")


def build_tract_features(acs_year: int) -> pd.DataFrame:
```

```
    inc  = census_get_group(acs_year, "B19013")   # median income
    edu  = census_get_group(acs_year, "B15002")   # educational attainment
    race = census_get_group(acs_year, "B03002")   # race / Hispanic origin
    ten  = census_get_group(acs_year, "B25003")   # housing tenure
    age  = census_get_group(acs_year, "B01001")   # age structure

    out = pd.DataFrame({"GEOID": inc["GEOID"]})

    # Median household income
    out["median_hh_income"] = pd.to_numeric(inc["B19013_001E"], errors="coerce")

    # % College educated (BA+ among age 25+) via B15002
    edu_cols = [
        "B15002_001E",
        "B15002_015E", "B15002_016E", "B15002_017E", "B15002_018E",
        "B15002_032E", "B15002_033E", "B15002_034E", "B15002_035E",
    ]
    edu = edu[["GEOID"] + edu_cols].copy()
    to_num(edu, edu_cols)
    edu["ba_plus"] = (
        edu["B15002_015E"] + edu["B15002_016E"] +
        edu["B15002_017E"] + edu["B15002_018E"] +
        edu["B15002_032E"] + edu["B15002_033E"] +
        edu["B15002_034E"] + edu["B15002_035E"]
    )
    edu["pct_college"] = edu["ba_plus"] / edu["B15002_001E"]
    out = out.merge(edu[["GEOID", "pct_college"]], on="GEOID", how="left")

    # % Black and % Hispanic
    race_cols = ["B03002_001E", "B03002_004E", "B03002_012E"]
    race = race[["GEOID"] + race_cols].copy()
    to_num(race, race_cols)
    race["pct_black"]    = race["B03002_004E"] / race["B03002_001E"]
    race["pct_hispanic"] = race["B03002_012E"] / race["B03002_001E"]
    out = out.merge(race[["GEOID", "pct_black", "pct_hispanic"]],
                    on="GEOID", how="left")

    # % Renter occupied
    ten_cols = ["B25003_001E", "B25003_003E"]
    ten = ten[["GEOID"] + ten_cols].copy()
    to_num(ten, ten_cols)
    ten["pct_renter"] = ten["B25003_003E"] / ten["B25003_001E"]
```

```
    out = out.merge(ten[["GEOID", "pct_renter"]], on="GEOID", how="left")

    # % Age 18-29 and Voting Age Population (VAP)
    age_cols = [
        "B01001_001E",
        "B01001_003E", "B01001_004E", "B01001_005E", "B01001_006E",
        "B01001_027E", "B01001_028E", "B01001_029E", "B01001_030E",
        "B01001_007E", "B01001_031E",
        "B01001_008E", "B01001_009E", "B01001_010E",
        "B01001_032E", "B01001_033E", "B01001_034E",
        "B01001_011E", "B01001_035E",
    ]
    age = age[["GEOID"] + age_cols].copy()
    to_num(age, age_cols)
    age["pop_under18"] = (
        age["B01001_003E"] + age["B01001_004E"] +
        age["B01001_005E"] + age["B01001_006E"] +
        age["B01001_027E"] + age["B01001_028E"] +
        age["B01001_029E"] + age["B01001_030E"]
    )
    age["vap"] = age["B01001_001E"] - age["pop_under18"]
    age["pop_18_29"] = (
        (age["B01001_007E"] + age["B01001_031E"]) +
        (age["B01001_008E"] + age["B01001_009E"] + age["B01001_010E"] +
         age["B01001_032E"] + age["B01001_033E"] + age["B01001_034E"]) +
        (age["B01001_011E"] + age["B01001_035E"])
    )
    age["pct_18_29"] = age["pop_18_29"] / age["B01001_001E"]
    out = out.merge(age[["GEOID", "vap", "pct_18_29"]], on="GEOID", how="left")

    # Clip impossible share values
    for c in ["pct_college", "pct_black", "pct_hispanic", "pct_renter", "pct_18_29"]:
        out.loc[(out[c] < 0) | (out[c] > 1), c] = pd.NA

    return out


# Run for all election years
for election_year, acs_year in ELECTION_TO_ACS_API_YEAR.items():
    print(f"Downloading ACS {acs_year} for election {election_year}...")
    df = build_tract_features(acs_year)
    df.insert(0, "election_year", election_year)
```

```
    outpath = os.path.join(OUTDIR,
                           f"acs_{acs_year}_for_election_{election_year}.csv")
    df.to_csv(outpath, index=False)
    print(f"  Saved {outpath}  | rows={len(df):,}")
    time.sleep(0.5)

print("ACS download complete.")
```

### 1.2 ACS Post-Processing

```
# The Census API returns -666666666 for missing/suppressed income values.
# The original post-processing set these to 0; we retain that behaviour here
# for consistency with the saved files, but treat 0 as NaN in later steps.
for fname in ACS_FILES.values():
    fpath = DATA_ACS / fname
    df = pd.read_csv(fpath)
    df["median_hh_income"] = pd.to_numeric(df["median_hh_income"],
                                           errors="coerce")
    df.loc[df["median_hh_income"] < 0, "median_hh_income"] = 0
    df.to_csv(fpath, index=False)

print("Post-processing complete.")
```

### 1.3 Voter Turnout Cleaning

```
def clean_voter_csv(path_in: Path, path_out: Path) -> None:
    df = pd.read_csv(path_in, dtype=str)

    # Drop citywide summary rows at top
    df = df.iloc[2:].reset_index(drop=True)
    df = df.dropna(how="all").reset_index(drop=True)

    first_col = df.columns[0]
    fc = df[first_col].astype(str).str.strip()
    df[first_col] = fc

    # Forward-fill ward number from "Ward X" header rows
    is_ward = fc.str.startswith("Ward ")
```

```
    ward_series = (
        fc.where(is_ward)
          .str.replace("Ward", "", n=1)
          .str.strip()
          .str.extract(r"(\d+)")[0]
    )
    df["Ward"] = ward_series.ffill()

    # Remove ward headers, column-label rows, and ward-total rows
    is_precinct_header = fc.eq("Precinct")
    is_total           = fc.str.startswith("Total")
    df_clean = df[~(is_ward | is_precinct_header | is_total)].copy()

    # Standardise column names
    df_clean = df_clean.rename(columns={
        df_clean.columns[0]: "Precinct",
        df_clean.columns[1]: "Registered Voters",
        df_clean.columns[2]: "Ballots Cast",
        df_clean.columns[3]: "Turnout",
    })
    df_clean = df_clean[["Ward", "Precinct",
                         "Registered Voters", "Ballots Cast", "Turnout"]]

    df_clean["Ward"]    = pd.to_numeric(df_clean["Ward"],
                                        errors="coerce").astype("Int64")
    df_clean["Precinct"] = pd.to_numeric(df_clean["Precinct"],
                                         errors="coerce").astype("Int64")
    df_clean.to_csv(path_out, index=False)


for year in ELECTION_YEARS:
    in_path  = DATA_VOTER / f"{year}.csv"
    out_path = DATA_VOTER / f"{year}_clean.csv"
    clean_voter_csv(in_path, out_path)
    print(f"Cleaned {year}")

print("Voter Turnout data cleaning complete.")
```

---

## Part 2 – Geographic Foundation

We load the Chicago city boundary shapefile and the 2020 Census tract shapefile (all of Illinois), then filter to the 1,309+ Cook County tracts that intersect the Chicago city boundary. These **2020 tract boundaries are used consistently for all five election years** to ensure geographic comparability across time.

```
# Chicago city boundary (single polygon after dissolve)
chi_boundary = gpd.read_file(CITY_BNDRY).to_crs(CRS_PROJ)
chi_union    = chi_boundary.dissolve().geometry.iloc[0]

# 2020 Census tract shapefile - all of Illinois
tracts_all  = gpd.read_file(TRACT_SHP).to_crs(CRS_PROJ)

# Keep only Cook County tracts
tracts_cook = tracts_all[tracts_all["COUNTYFP"] == "031"].copy()

# Keep tracts that intersect the Chicago city boundary
tracts_chicago = (
    tracts_cook[tracts_cook.geometry.intersects(chi_union)]
    .copy()
    .reset_index(drop=True)
)

print(f"Illinois tracts total : {len(tracts_all):>6,}")
print(f"Cook County tracts    : {len(tracts_cook):>6,}")
print(f"Chicago tracts kept   : {len(tracts_chicago):>6,}")
print(f"Columns available     : {list(tracts_chicago.columns)}")

assert len(tracts_chicago) > 800, f"Expected 800+ Chicago tracts, got {len(tracts_chicago)}"
```

```
Illinois tracts total :  3,265
Cook County tracts    :  1,332
Chicago tracts kept   :    866
Columns available     : ['STATEFP', 'COUNTYFP', 'TRACTCE', 'GEOID', 'NAME', 'NAMELSAD', 'MTFC
```

---

## Part 3 – Attach ACS Demographics to Census Tracts

For each election year we merge the corresponding ACS 5-year CSV onto the Chicago tract `GeoDataFrame` using the `GEOID` key. Income values stored as `0` (the post-processing placeholder

for Census missing values) are converted to `NaN` before merging so they do not bias weighted averages.

```
def load_acs(year: int) -> pd.DataFrame:
    """Load ACS CSV and convert 0-income placeholder to NaN."""
    df = pd.read_csv(DATA_ACS / ACS_FILES[year])
    df["median_hh_income"] = pd.to_numeric(df["median_hh_income"],
                                           errors="coerce")
    # 0 was used as placeholder for missing/suppressed Census income values
    df.loc[df["median_hh_income"] == 0, "median_hh_income"] = np.nan
    return df


def make_tract_geodf(year: int, chicago_tracts: gpd.GeoDataFrame
                     ) -> gpd.GeoDataFrame:
    """Merge ACS demographic data onto the Chicago tract GeoDataFrame."""
    acs = load_acs(year)
    gdf = chicago_tracts[["GEOID", "geometry"]].copy()
    gdf["GEOID"] = gdf["GEOID"].astype(str)
    acs = acs.copy()
    acs["GEOID"] = acs["GEOID"].astype(str)

    # Merge diagnostics
    tract_set = set(gdf["GEOID"])
    acs_set   = set(acs["GEOID"])
    print(f"  Tracts not in ACS: {len(tract_set - acs_set)}")
    print(f"  ACS tracts not in Chicago: {len(acs_set - tract_set)}")

    gdf = gdf.merge(acs.drop(columns=["election_year"]),
                    on="GEOID", how="left")

    assert len(gdf) == len(chicago_tracts), \
        f"Merge changed row count: {len(chicago_tracts)} → {len(gdf)}"

    return gdf


# Quick verification for 2024
tract_2024 = make_tract_geodf(2024, tracts_chicago)
n_income   = tract_2024["median_hh_income"].notna().sum()
print(f"2024 - tracts: {len(tract_2024):,}  |  income coverage: {n_income} non-null")
tract_2024[["GEOID", "median_hh_income", "pct_college",
            "pct_black", "pct_hispanic", "pct_renter",
```

```
            "vap", "pct_18_29"]].head(4)
```

```
  Tracts not in ACS: 0
  ACS tracts not in Chicago: 466
2024 - tracts: 866  |  income coverage: 846 non-null
```

| | GEOID | median_hh_income | pct_college | pct_black | pct_hispanic | pct_renter | vap | pct_18 |
|---|---|---|---|---|---|---|---|---|
| 0 | 17031510300 | 36232.0 | 0.180707 | 0.990101 | 0.000000 | 0.525926 | 3363 | 0.1482 |
| 1 | 17031520100 | 46786.0 | 0.113309 | 0.013216 | 0.917952 | 0.430894 | 1221 | 0.1117 |
| 2 | 17031590700 | 72355.0 | 0.280214 | 0.005110 | 0.634827 | 0.254601 | 2116 | 0.1867 |
| 3 | 17031600400 | 66964.0 | 0.350883 | 0.059250 | 0.087787 | 0.521499 | 3459 | 0.2072 |

---

## Part 4 – Prepare Precinct Geometry

Each election year uses the appropriate precinct boundary file (see mapping table in the overview). We standardise the precinct identifier to a **5-character string** — zero-padded ward number (2 digits) followed by zero-padded precinct number (3 digits) — matching the format used in the voter-turnout CSVs.

```python
def load_precinct_gdf(year: int) -> gpd.GeoDataFrame:
    """
    Load the correct precinct GeoJSON for a given election year and return a
    GeoDataFrame with a standardised 'precinct_id' (WWPPP format).
    """
    fname = PRECINCT_FILES[year]
    gdf   = gpd.read_file(PRECINCT_DIR / fname).to_crs(CRS_PROJ)

    gdf["ward_num"]     = pd.to_numeric(gdf["ward"],     errors="coerce")
    gdf["precinct_num"] = pd.to_numeric(gdf["precinct"], errors="coerce")

    # Drop rows where ward/precinct cannot be parsed
    gdf = gdf.dropna(subset=["ward_num", "precinct_num"]).copy()
    gdf["ward_num"]     = gdf["ward_num"].astype(int)
    gdf["precinct_num"] = gdf["precinct_num"].astype(int)

    # 5-character ID:  ward 27, precinct 36  →  "27036"
    gdf["precinct_id"] = (
```

```
        gdf["ward_num"].apply(lambda w: f"{w:02d}") +
        gdf["precinct_num"].apply(lambda p: f"{p:03d}")
    )

    # Remove geometric duplicates (keep first)
    gdf = gdf.drop_duplicates("precinct_id").reset_index(drop=True)
    print(f"  CRS: {gdf.crs.to_epsg()}")
    return gdf[["precinct_id", "ward_num", "precinct_num", "geometry"]]


# Preview
for yr in ELECTION_YEARS:
    g = load_precinct_gdf(yr)
    print(f"{yr}: {len(g):,} unique precincts")
```

```
  CRS: 26916
2008: 2,574 unique precincts
  CRS: 26916
2012: 2,574 unique precincts
  CRS: 26916
2016: 2,069 unique precincts
  CRS: 26916
2020: 2,069 unique precincts
  CRS: 26916
2024: 1,291 unique precincts
```

---

## Part 5 – Spatial Overlay: Tract Demographics → Precinct Level

This is the core geographic transformation. We use **area-weighted interpolation** to convert census-tract-level ACS estimates into precinct-level estimates.

**Algorithm:**

1. Intersect precinct polygons with census-tract polygons.
2. For each intersection fragment: `weight = fragment_area / parent_precinct_area`.
3. For demographic *rates/percentages* (income, `pct_*`): weighted average, normalised by the sum of weights from tracts with non-null values.
4. For **VAP** (a population count): proportional sum — `sum(weight × vap_tract)`.

Both inputs are reprojected to UTM 16N before the overlay so that areas are measured in square metres.

```
def area_weighted_interpolate(
    precincts_gdf : gpd.GeoDataFrame,
    tracts_gdf    : gpd.GeoDataFrame,
    demo_cols     : list,
    vap_col       : str = "vap",
) -> pd.DataFrame:
    """
    Interpolate tract-level demographics onto precincts via area weighting.

    Uses gpd.sjoin to identify overlapping precinct-tract pairs, then
    computes intersection geometries and area-based weights.
    """
    prec   = precincts_gdf[["precinct_id", "geometry"]].copy()
    tracts = tracts_gdf[["GEOID", "geometry"] + demo_cols + [vap_col]].copy()

    # Precinct area in projected units (sq metres)
    prec["_prec_area"] = prec.geometry.area

    # Step 1: sjoin to find overlapping precinct-tract pairs
    joined = gpd.sjoin(prec, tracts, how="inner", predicate="intersects")
    if "index_right" in joined.columns:
        joined = joined.drop(columns=["index_right"])
    print(f"    sjoin pairs: {len(joined):,}")

    # Step 2: Compute intersection geometry and area weights
    # Retrieve the original tract geometry for each matched pair
    joined = joined.merge(
        tracts[["GEOID", "geometry"]].rename(columns={"geometry": "_tract_geom"}),
        on="GEOID",
        how="left",
    )
    joined["_inter_geom"] = joined.apply(
        lambda row: row["geometry"].intersection(row["_tract_geom"]), axis=1
    )
    joined["_inter_area"] = joined["_inter_geom"].apply(lambda g: g.area)
    joined["_w"] = joined["_inter_area"] / joined["_prec_area"]

    results = {}

    # Demographic rate / percentage columns
```

```
    for col in demo_cols:
        joined["_wv"] = joined["_w"] * joined[col]
        valid = joined.dropna(subset=[col])
        grp = valid.groupby("precinct_id").agg(
            _wsum   = ("_wv", "sum"),
            _wdenom = ("_w",  "sum"),
        )
        grp[col] = grp["_wsum"] / grp["_wdenom"]
        results[col] = grp[col]

    # VAP: proportional count
    joined["_w_vap"] = joined["_w"] * joined[vap_col].fillna(0)
    vap_grp = (
        joined.groupby("precinct_id")["_w_vap"]
              .sum()
              .rename(vap_col)
    )
    results[vap_col] = vap_grp

    out = pd.concat(results, axis=1).reset_index()
    return out
```

---

### Part 6 – Load and Standardise Voter Turnout

```
def load_turnout(year: int) -> pd.DataFrame:
    """
    Load cleaned voter CSV, build standardised precinct_id, and compute
    turnout rate (ballots cast / registered voters).
    """
    df = pd.read_csv(DATA_VOTER / f"{year}_clean.csv")

    # Drop rows without ward or precinct
    df = df.dropna(subset=["Ward", "Precinct"]).copy()
    df["Ward"]     = df["Ward"].astype(int)
    df["Precinct"] = df["Precinct"].astype(int)

    # 5-character precinct ID matching the geometry files
    df["precinct_id"] = (
```

```
        df["Ward"].apply(lambda w: f"{w:02d}") +
        df["Precinct"].apply(lambda p: f"{p:03d}")
    )

    df["registered_voters"] = pd.to_numeric(df["Registered Voters"],
                                            errors="coerce")
    df["ballots_cast"]      = pd.to_numeric(df["Ballots Cast"],
                                            errors="coerce")

    # Turnout rate - drop impossible values ( 0 or > 1)
    df["turnout_rate"] = df["ballots_cast"] / df["registered_voters"]
    df = df[(df["turnout_rate"] > 0) & (df["turnout_rate"] <= 1.0)].copy()

    df["election_year"] = year
    return df[["election_year", "precinct_id", "Ward", "Precinct",
               "registered_voters", "ballots_cast", "turnout_rate"]]
```

---

### Part 7 – Build Master Panel Dataset

We run the full pipeline for all five election years and stack the results into a single master dataset. Each row represents one precinct in one election year.

```
os.makedirs(ROOT / "data", exist_ok=True)

all_years = []

for year in ELECTION_YEARS:
    print(f"\n Processing {year} ")

    # Step 4 - Precinct geometry
    prec_gdf = load_precinct_gdf(year)
    print(f"  Precincts loaded : {len(prec_gdf):,}")

    # Step 3 - Tract demographics for this year
    tract_gdf = make_tract_geodf(year, tracts_chicago)

    # Step 5 - Spatial overlay (sjoin + area-weighted interpolation)
    demo_prec = area_weighted_interpolate(prec_gdf, tract_gdf,
                                          DEMO_COLS, vap_col="vap")
```

```
    print(f"  Demo interpolated: {len(demo_prec):,} precincts")

    # Step 6 - Turnout data
    turnout_df = load_turnout(year)
    print(f"  Turnout rows     : {len(turnout_df):,}")

    # Merge turnout + demographics
    merged = turnout_df.merge(demo_prec, on="precinct_id", how="inner")
    print(f"  Merged (inner)   : {len(merged):,}")

    # Registration rate = registered voters / VAP
    merged["registration_rate"] = (
        merged["registered_voters"] / merged["vap"]
    )
    merged["registration_rate"] = merged["registration_rate"].clip(upper=1.0)

    n_inc = merged["median_hh_income"].notna().sum()
    print(f"  Income coverage  : {n_inc} / {len(merged)}")

    all_years.append(merged)

master = pd.concat(all_years, ignore_index=True)

# Save to disk
master_path = ROOT / "data" / "master_panel.csv"
master.to_csv(master_path, index=False)

print(f"\n  Master dataset saved to {master_path}")
print(f"  Shape: {master.shape[0]:,} rows × {master.shape[1]} columns")
master.head(3)
```

```
 Processing 2008
  CRS: 26916
  Precincts loaded : 2,574
  Tracts not in ACS: 12
  ACS tracts not in Chicago: 465
    sjoin pairs: 8,675
  Demo interpolated: 2,574 precincts
  Turnout rows     : 2,559
  Merged (inner)   : 2,557
  Income coverage  : 2547 / 2557
```

```
Processing 2012
 CRS: 26916
 Precincts loaded : 2,574
 Tracts not in ACS: 12
 ACS tracts not in Chicago: 465
   sjoin pairs: 8,675
 Demo interpolated: 2,574 precincts
 Turnout rows     : 2,034
 Merged (inner)   : 2,031
 Income coverage  : 2022 / 2031

Processing 2016
 CRS: 26916
 Precincts loaded : 2,069
 Tracts not in ACS: 12
 ACS tracts not in Chicago: 465
   sjoin pairs: 7,632
 Demo interpolated: 2,069 precincts
 Turnout rows     : 2,068
 Merged (inner)   : 2,068
 Income coverage  : 2058 / 2068

Processing 2020
 CRS: 26916
 Precincts loaded : 2,069
 Tracts not in ACS: 0
 ACS tracts not in Chicago: 466
   sjoin pairs: 7,632
 Demo interpolated: 2,069 precincts
 Turnout rows     : 2,068
 Merged (inner)   : 2,068
 Income coverage  : 2068 / 2068

Processing 2024
 CRS: 26916
 Precincts loaded : 1,291
 Tracts not in ACS: 0
 ACS tracts not in Chicago: 466
   sjoin pairs: 5,893
 Demo interpolated: 1,291 precincts
 Turnout rows     : 1,284
 Merged (inner)   : 1,284
```

```
  Income coverage  : 1284 / 1284

 Master dataset saved to data/master_panel.csv
  Shape: 10,008 rows × 15 columns
```

| | election_year | precinct_id | Ward | Precinct | registered_voters | ballots_cast | turnout_rate | median_ |
|---|---|---|---|---|---|---|---|---|
| 0 | 2008 | 01001 | 1 | 1 | 835 | 545 | 0.652695 | 43042.9 |
| 1 | 2008 | 01002 | 1 | 2 | 825 | 585 | 0.709091 | 65354.0 |
| 2 | 2008 | 01003 | 1 | 3 | 587 | 465 | 0.792164 | 69750.4 |

---

## Part 8 – Analysis

**Utilities:** We define a helper to compute Spearman rank correlations and set up demographic column labels used throughout this section.

```python
def spearman_corr(x, y):
    """Compute Spearman rank correlation using pandas rank method."""
    valid = pd.DataFrame({"x": x, "y": y}).dropna()
    if len(valid) < 10:
        return np.nan
    return valid["x"].rank().corr(valid["y"].rank())


demo_labels = {
    "median_hh_income": "Income",
    "pct_college":      "% College",
    "pct_black":        "% Black",
    "pct_hispanic":     "% Hispanic",
    "pct_renter":       "% Renter",
    "pct_18_29":        "% Age 18-29",
}
```

---

### 8.1 Summary Statistics by Election Year

```
summary = master.groupby("election_year").agg(
    n_precincts        = ("precinct_id",       "count"),
    avg_turnout        = ("turnout_rate",       "mean"),
    median_turnout     = ("turnout_rate",       "median"),
    avg_reg_rate       = ("registration_rate",  lambda x: x.dropna().mean()),
    avg_income_k       = ("median_hh_income",   lambda x: x.dropna().mean() / 1000),
    avg_pct_college    = ("pct_college",        "mean"),
    avg_pct_black      = ("pct_black",          "mean"),
    avg_pct_hispanic   = ("pct_hispanic",       "mean"),
    avg_pct_renter     = ("pct_renter",         "mean"),
    avg_pct_18_29      = ("pct_18_29",          "mean"),
).round(3)

summary.columns = [
    "N Precincts", "Avg Turnout", "Median Turnout", "Avg Reg Rate",
    "Avg Income ($K)", "Avg % College", "Avg % Black",
    "Avg % Hispanic", "Avg % Renter", "Avg % 18-29"
]
summary
```

| election_year | N Precincts | Avg Turnout | Median Turnout | Avg Reg Rate | Avg Income ($K) | Avg % Co |
|---|---|---|---|---|---|---|
| 2008 | 2557 | 0.736 | 0.738 | 0.239 | 48.201 | 0.300 |
| 2012 | 2031 | 0.753 | 0.765 | 0.276 | 49.815 | 0.319 |
| 2016 | 2068 | 0.707 | 0.702 | 0.298 | 59.417 | 0.360 |
| 2020 | 2068 | 0.724 | 0.721 | 0.283 | 66.349 | 0.385 |
| 2024 | 1284 | 0.671 | 0.662 | 0.413 | 83.480 | 0.423 |

---

### 8.2 Turnout Rate Distribution by Election Year

```
chart_dist = (
    alt.Chart(master)
    .mark_area(opacity=0.45, interpolate="monotone")
    .transform_density(
        density="turnout_rate",
        as_=["turnout_rate", "density"],
```

```
        groupby=["election_year"],
        extent=[0, 1],
    )
    .encode(
        x=alt.X("turnout_rate:Q",
                title="Turnout Rate (Ballots / Registered Voters)"),
        y=alt.Y("density:Q", title="Density"),
        color=alt.Color("election_year:N",
                        scale=alt.Scale(scheme="set2"),
                        title="Election Year"),
    )
    .properties(width=600, height=350,
                title="Precinct Turnout Rate Distribution by Election Year")
)
display_altair_png(chart_dist)
```

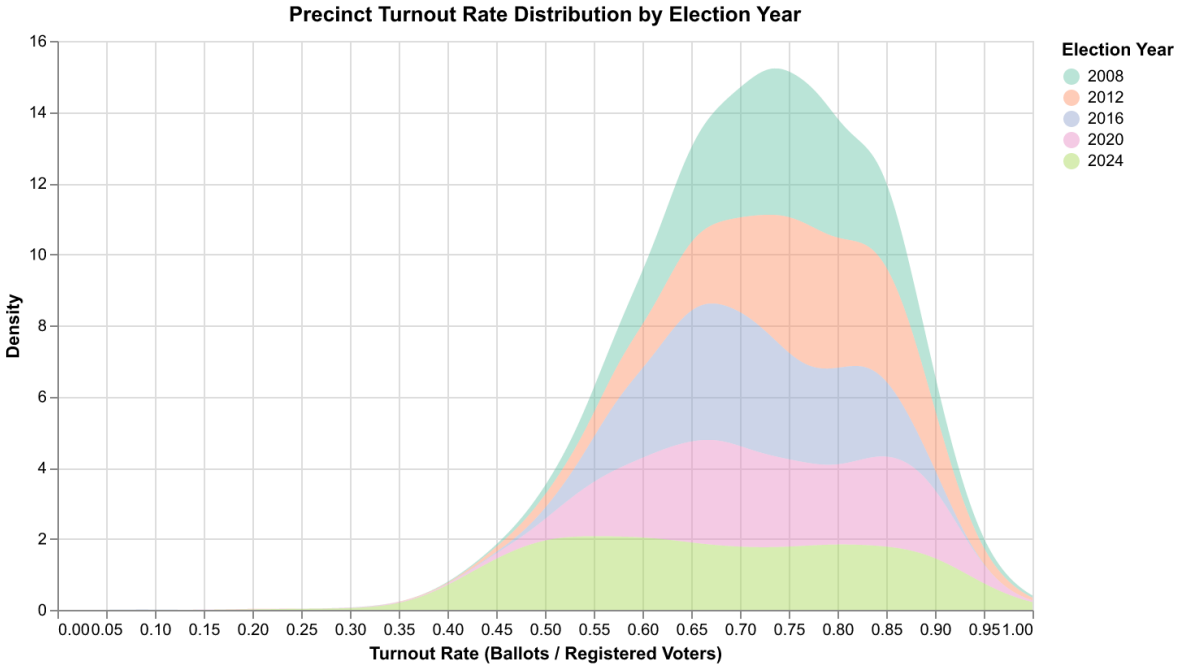


Figure 1: Distribution of precinct-level turnout rates across election years

### 8.3 Average Turnout by Year

```
avg_t = (
    master.groupby("election_year")["turnout_rate"]
    .mean()
    .reset_index()
    .rename(columns={"turnout_rate": "avg_turnout"})
)
avg_t["avg_turnout_pct"] = (avg_t["avg_turnout"] * 100).round(1)
avg_t["election_year_str"] = avg_t["election_year"].astype(str)

bars = (
    alt.Chart(avg_t)
    .mark_bar(cornerRadiusTopLeft=4, cornerRadiusTopRight=4)
    .encode(
        x=alt.X("election_year_str:N", title="Election Year",
                sort=None),
        y=alt.Y("avg_turnout_pct:Q", title="Average Turnout Rate (%)",
                scale=alt.Scale(domain=[0, 105])),
        color=alt.Color("election_year_str:N",
                        scale=alt.Scale(scheme="set2"),
                        legend=None),
        tooltip=[
            alt.Tooltip("election_year:O", title="Year"),
            alt.Tooltip("avg_turnout_pct:Q", title="Avg Turnout %",
                        format=".1f"),
        ],
    )
)

text = bars.mark_text(dy=-10, fontWeight="bold", fontSize=12).encode(
    text=alt.Text("avg_turnout_pct:Q", format=".1f")
)

chart_avg = (
    (bars + text)
    .properties(width=450, height=300,
                title="Mean Precinct-Level Voter Turnout by Election Year")
)
chart_avg
```

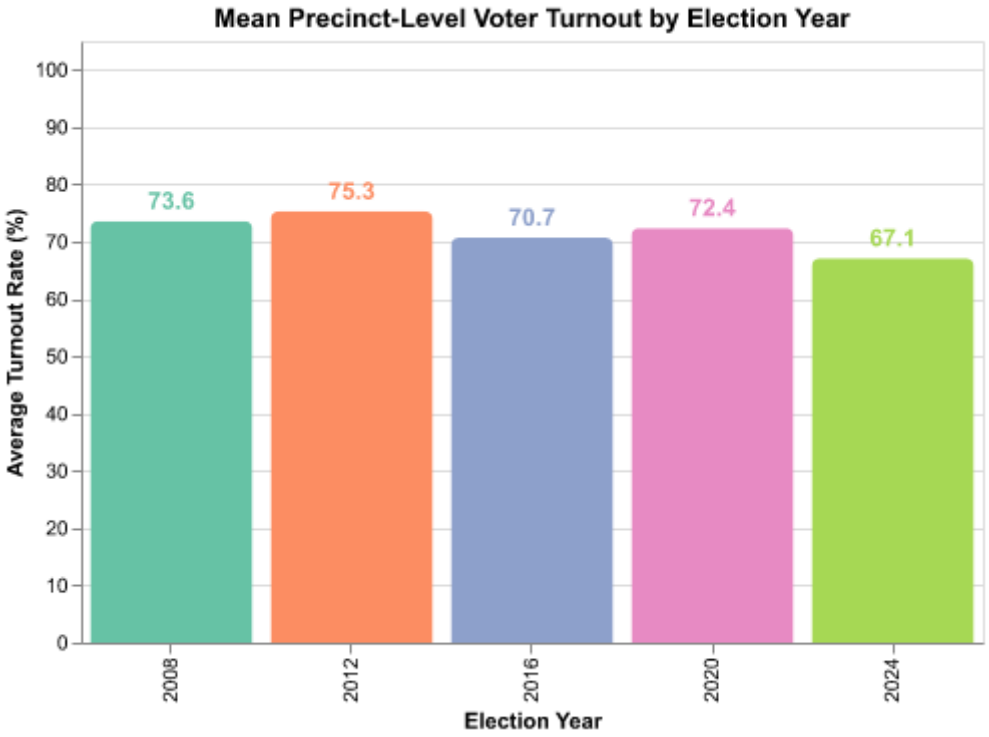


Figure 2: Mean precinct-level turnout rate by election year

---

### 8.4 Correlation Heatmap: Demographics vs Turnout

Spearman rank correlations (robust to outliers and non-linearity) between precinct turnout rate and each demographic variable, computed separately for each election year.

```
corr_rows = []
for year in ELECTION_YEARS:
    sub = master[master["election_year"] == year].copy()
    row = {"Year": year}
    for col in DEMO_COLS:
        valid = sub[["turnout_rate", col]].dropna()
        if len(valid) >= 30:
            r = spearman_corr(valid["turnout_rate"], valid[col])
            row[col] = round(r, 3)
        else:
            row[col] = np.nan
```

```
    corr_rows.append(row)

corr_df = (
    pd.DataFrame(corr_rows)
      .set_index("Year")
      .rename(columns=demo_labels)
)

# Reshape to long form for Altair
corr_long = corr_df.reset_index().melt(
    id_vars="Year", var_name="Variable", value_name="Spearman_r"
)

chart_heatmap = (
    alt.Chart(corr_long)
    .mark_rect()
    .encode(
        x=alt.X("Variable:N", title=None,
                sort=list(demo_labels.values())),
        y=alt.Y("Year:O", title="Election Year"),
        color=alt.Color("Spearman_r:Q",
                        scale=alt.Scale(scheme="redblue",
                                        domain=[-0.75, 0.75]),
                        title="Spearman r"),
        tooltip=[
            alt.Tooltip("Year:O"),
            alt.Tooltip("Variable:N"),
            alt.Tooltip("Spearman_r:Q", format=".3f"),
        ],
    )
)

text_hm = chart_heatmap.mark_text(fontSize=12).encode(
    text=alt.Text("Spearman_r:Q", format=".2f"),
    color=alt.condition(
        "abs(datum.Spearman_r) > 0.4",
        alt.value("white"),
        alt.value("black"),
    ),
)

chart_corr = (
```

```
    (chart_heatmap + text_hm)
    .properties(width=500, height=220,
                title="Spearman Correlation: Precinct Turnout Rate vs Demographic Variables")
)
display_altair_png(chart_corr)
```

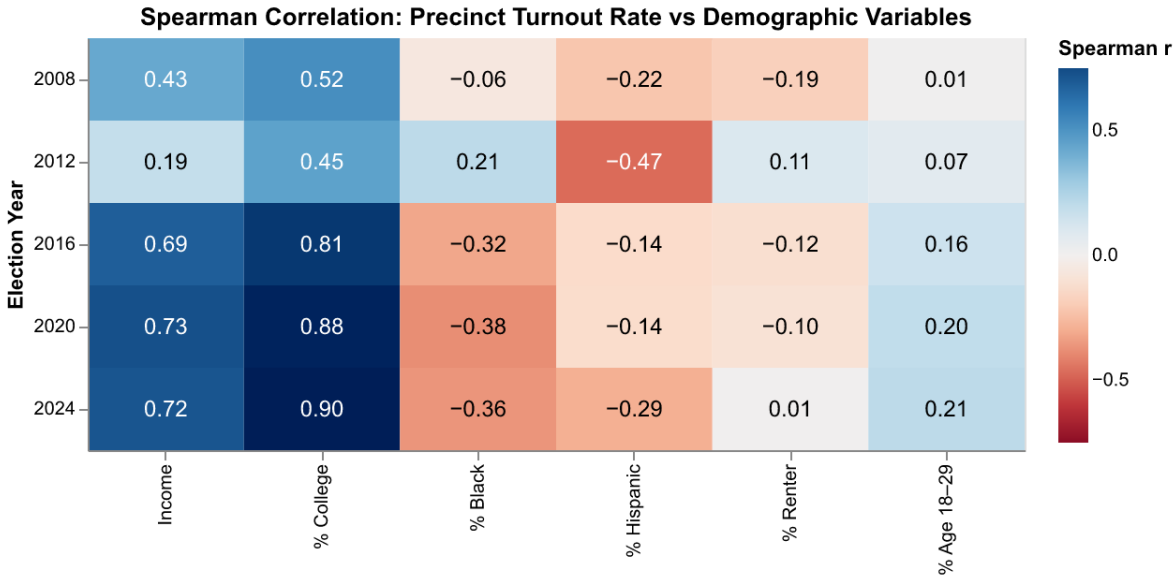


Figure 3: Spearman correlations between precinct turnout rate and demographic variables by year

---

### 8.5 Correlation Trends Over Time

```
# Selection for hover highlight
highlight = alt.selection_point(on="pointerover", fields=["Variable"],
                                nearest=True)

chart_trends = (
    alt.Chart(corr_long)
    .mark_line(point=True, strokeWidth=2.5)
    .encode(
        x=alt.X("Year:O", title="Election Year"),
```

```
        y=alt.Y("Spearman_r:Q", title="Spearman r",
                 scale=alt.Scale(domain=[-0.65, 0.65])),
        color=alt.Color("Variable:N", title="Demographic",
                        scale=alt.Scale(scheme="tableau10")),
        opacity=alt.condition(highlight, alt.value(1), alt.value(0.3)),
        tooltip=[
            alt.Tooltip("Year:O"),
            alt.Tooltip("Variable:N"),
            alt.Tooltip("Spearman_r:Q", format=".3f"),
        ],
    )
    .add_params(highlight)
)

rule_zero = (
    alt.Chart(pd.DataFrame({"y": [0]}))
    .mark_rule(strokeDash=[5, 5], color="black", opacity=0.5)
    .encode(y="y:Q")
)

chart_trends_final = (
    (chart_trends + rule_zero)
    .properties(width=550, height=350,
                title="Correlation of Precinct Turnout with Demographic Variables Over Time")
)
chart_trends_final
```

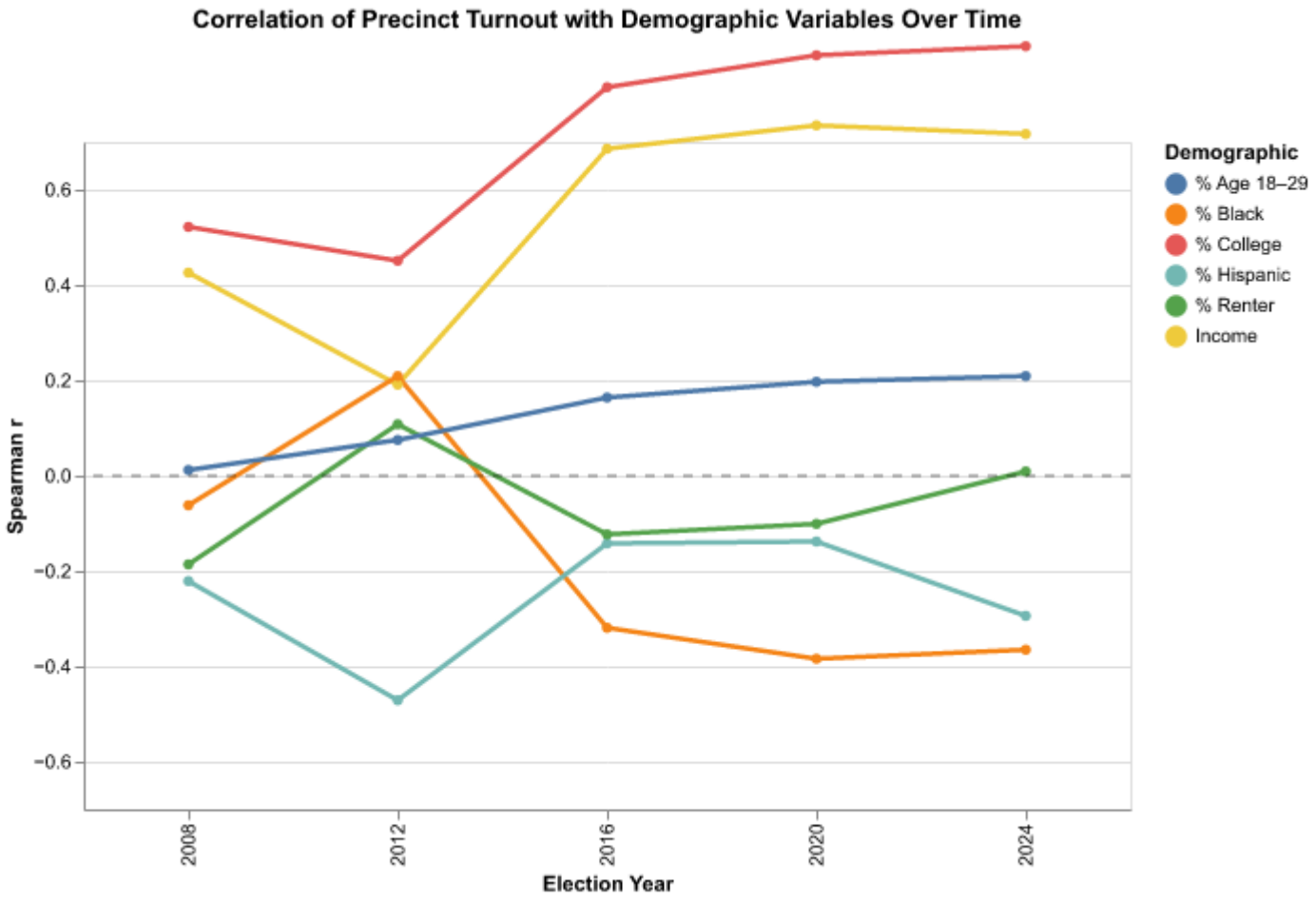


Figure 4: How turnout–demographic correlations change across election years (interactive)

---

### 8.6 Turnout vs Median Household Income

```
plot_inc = master[["election_year", "turnout_rate", "median_hh_income"]].dropna().copy()
plot_inc["income_k"]       = plot_inc["median_hh_income"] / 1000
plot_inc["turnout_pct"]    = plot_inc["turnout_rate"] * 100
plot_inc["election_year_str"] = plot_inc["election_year"].astype(str)

base_inc = alt.Chart(plot_inc)
```

```
scatter_inc = base_inc.mark_circle(size=8, opacity=0.25, color="#1565C0").encode(
    x=alt.X("income_k:Q", title="Income ($K)"),
    y=alt.Y("turnout_pct:Q", title="Turnout Rate (%)"),
)

reg_inc = base_inc.transform_regression(
    "income_k", "turnout_pct", groupby=["election_year_str"]
).mark_line(color="red", strokeWidth=2).encode(
    x="income_k:Q",
    y="turnout_pct:Q",
)

# Layer first, then facet
chart_inc_final = (
    (scatter_inc + reg_inc)
    .facet(facet=alt.Facet("election_year_str:N", title="Election Year",
                           sort=None),
           columns=5)
    .properties(title="Precinct Turnout Rate vs Median Household Income")
)
display_altair_png(chart_inc_final, scale=3)
```

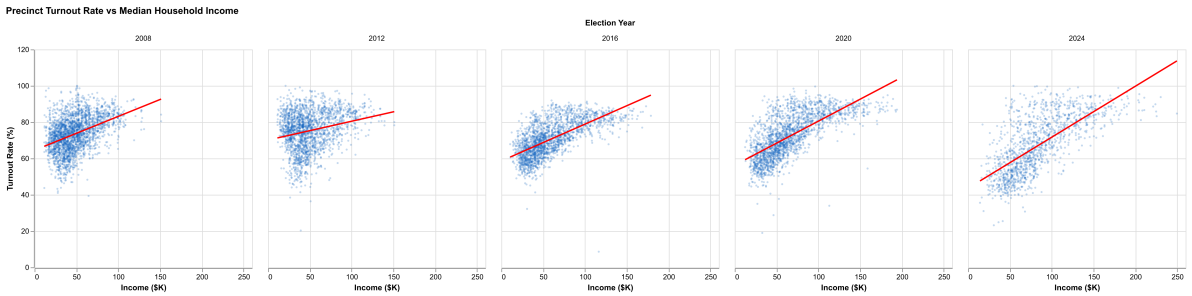


Figure 5: Precinct turnout rate vs median household income (each panel = one election year)

---

### 8.7 Turnout vs College Education

```
plot_edu = master[["election_year", "turnout_rate", "pct_college"]].dropna().copy()
plot_edu["pct_college_100"]   = plot_edu["pct_college"] * 100
```

```
plot_edu["turnout_pct"]       = plot_edu["turnout_rate"] * 100
plot_edu["election_year_str"] = plot_edu["election_year"].astype(str)

base_edu = alt.Chart(plot_edu)

scatter_edu = base_edu.mark_circle(size=8, opacity=0.25, color="#2E7D32").encode(
    x=alt.X("pct_college_100:Q", title="% College"),
    y=alt.Y("turnout_pct:Q", title="Turnout Rate (%)"),
)

reg_edu = base_edu.transform_regression(
    "pct_college_100", "turnout_pct", groupby=["election_year_str"]
).mark_line(color="red", strokeWidth=2).encode(
    x="pct_college_100:Q",
    y="turnout_pct:Q",
)

display_altair_png(
    (scatter_edu + reg_edu)
    .facet(facet=alt.Facet("election_year_str:N", title="Election Year",
                           sort=None),
           columns=5)
    .properties(title="Precinct Turnout Rate vs Share with Bachelor's Degree or Higher"),
    scale=3,
)
```

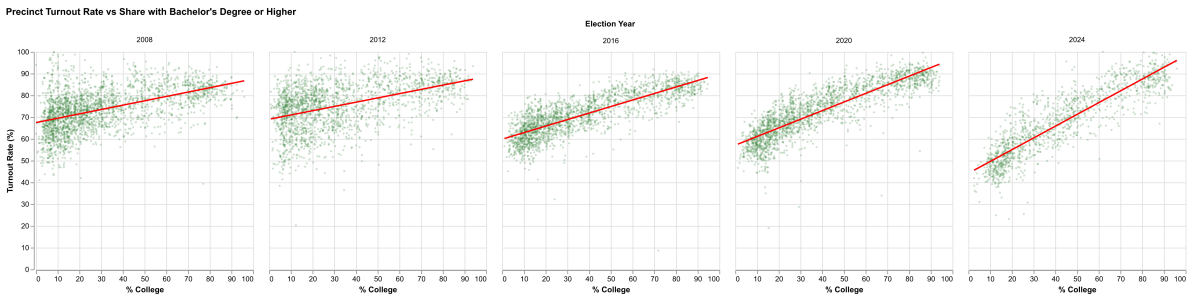


Figure 6: Precinct turnout rate vs share with bachelor's degree or higher

### 8.8 Turnout by Income Quintile

```
q_rows = []
for year in ELECTION_YEARS:
    sub = master[master["election_year"] == year][
        ["turnout_rate", "median_hh_income"]
    ].dropna().copy()
    sub["income_q"] = pd.qcut(sub["median_hh_income"], q=5, labels=False) + 1
    qmeans = sub.groupby("income_q")["turnout_rate"].mean() * 100
    for q, val in qmeans.items():
        q_rows.append({"Year": str(year), "Quintile": int(q), "Avg Turnout %": round(val, 2)}

q_df = pd.DataFrame(q_rows)
q_labels = {1: "Q1 (Lowest)", 2: "Q2", 3: "Q3", 4: "Q4", 5: "Q5 (Highest)"}
q_df["Quintile Label"] = q_df["Quintile"].map(q_labels)

chart_q_inc = (
    alt.Chart(q_df)
    .mark_line(point=True, strokeWidth=2.5)
    .encode(
        x=alt.X("Quintile Label:N", title="Income Quintile",
                sort=list(q_labels.values())),
        y=alt.Y("Avg Turnout %:Q", title="Average Turnout Rate (%)"),
        color=alt.Color("Year:N", scale=alt.Scale(scheme="set1"),
                        title="Election Year"),
        tooltip=[
            alt.Tooltip("Year:N"),
            alt.Tooltip("Quintile Label:N", title="Quintile"),
            alt.Tooltip("Avg Turnout %:Q", format=".1f"),
        ],
    )
    .properties(width=500, height=350,
                title="Voter Turnout by Income Quintile Across Election Years")
)
chart_q_inc
```

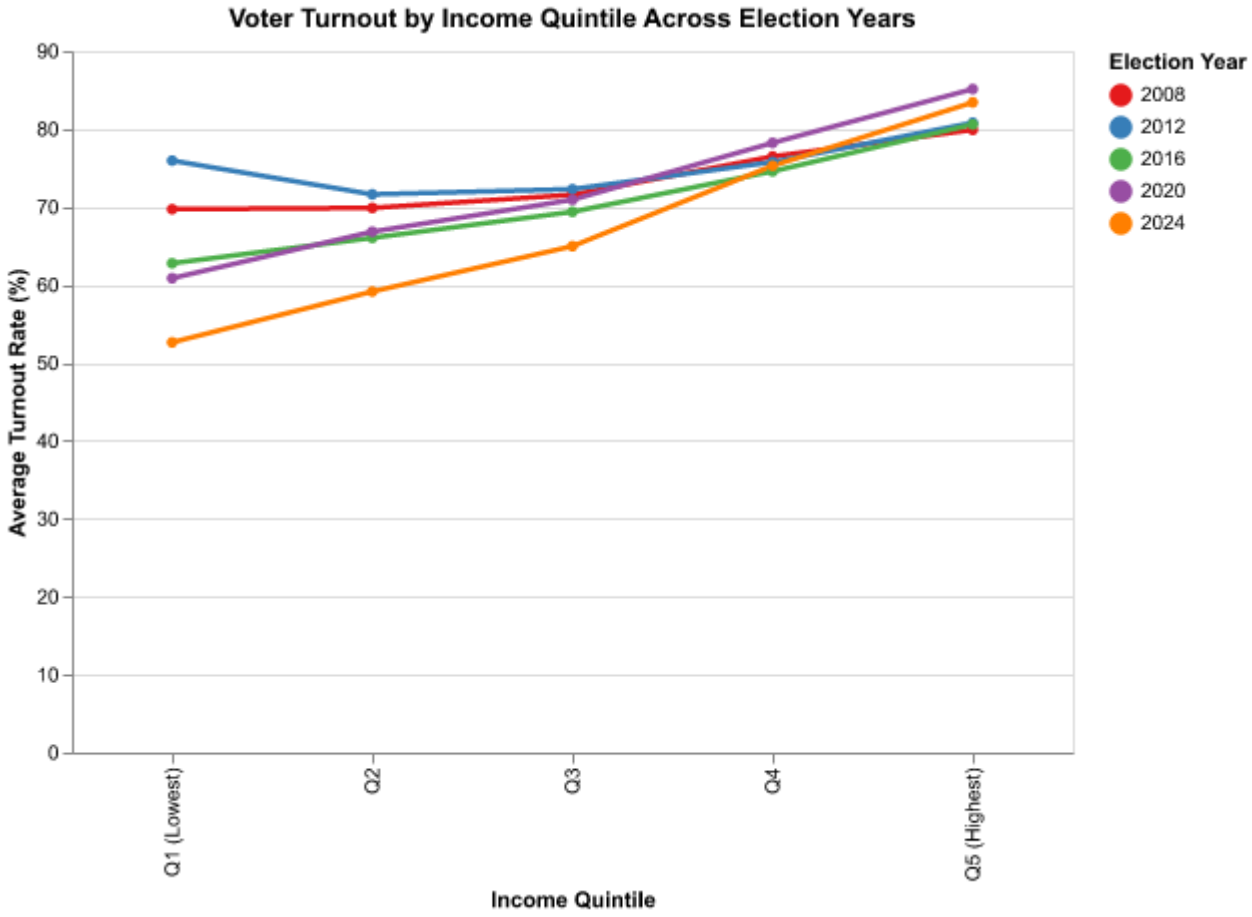


Figure 7: Average turnout rate by income quintile and election year (interactive)

---

### 8.9 Turnout by Education Quintile

```
eq_rows = []
for year in ELECTION_YEARS:
    sub = master[master["election_year"] == year][
        ["turnout_rate", "pct_college"]
    ].dropna().copy()
    sub["edu_q"] = pd.qcut(sub["pct_college"], q=5, labels=False) + 1
    qmeans = sub.groupby("edu_q")["turnout_rate"].mean() * 100
    for q, val in qmeans.items():
```

```
        eq_rows.append({"Year": str(year), "Quintile": int(q), "Avg Turnout %": round(val, 2)

eq_df = pd.DataFrame(eq_rows)
eq_df["Quintile Label"] = eq_df["Quintile"].map(q_labels)

chart_q_edu = (
    alt.Chart(eq_df)
    .mark_line(point=True, strokeWidth=2.5)
    .encode(
        x=alt.X("Quintile Label:N",
                title="Education Quintile (% BA+)",
                sort=list(q_labels.values())),
        y=alt.Y("Avg Turnout %:Q", title="Average Turnout Rate (%)"),
        color=alt.Color("Year:N", scale=alt.Scale(scheme="set1"),
                        title="Election Year"),
        tooltip=[
            alt.Tooltip("Year:N"),
            alt.Tooltip("Quintile Label:N", title="Quintile"),
            alt.Tooltip("Avg Turnout %:Q", format=".1f"),
        ],
    )
    .properties(width=500, height=350,
                title="Voter Turnout by Education Quintile Across Election Years")
)
chart_q_edu
```

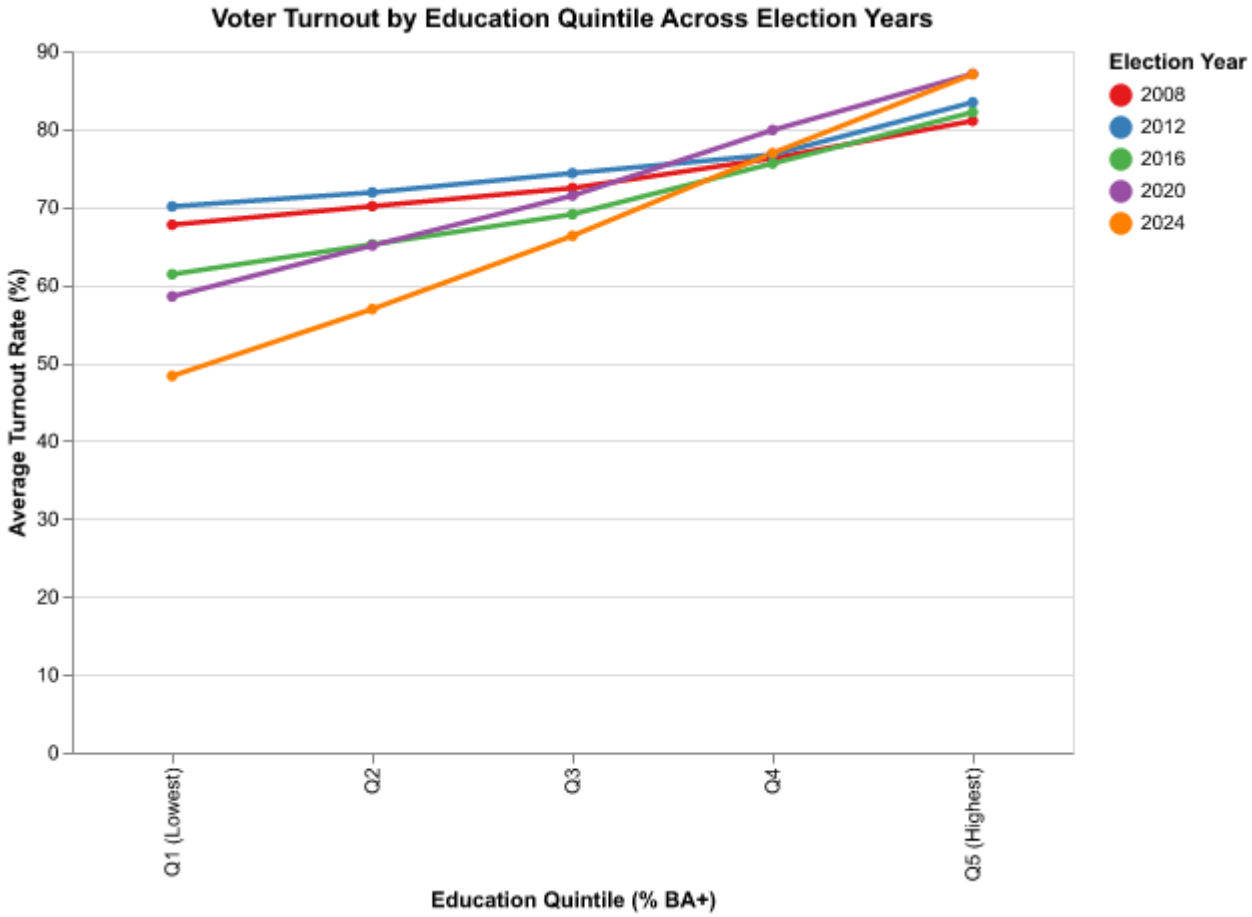


Figure 8: Average turnout rate by education quintile and election year (interactive)

---

### 8.10 Turnout vs Racial Composition

```
# % Black
plot_blk = master[["election_year", "turnout_rate", "pct_black"]].dropna().copy()
plot_blk["pct_black_100"]     = plot_blk["pct_black"] * 100
plot_blk["turnout_pct"]       = plot_blk["turnout_rate"] * 100
plot_blk["election_year_str"] = plot_blk["election_year"].astype(str)

base_blk = alt.Chart(plot_blk)
blk_pts = base_blk.mark_circle(size=8, opacity=0.25, color="#7B1FA2").encode(
```

```
    x=alt.X("pct_black_100:Q", title="% Black Population"),
    y=alt.Y("turnout_pct:Q",   title="Turnout Rate (%)"),
)
blk_reg = base_blk.transform_regression(
    "pct_black_100", "turnout_pct", groupby=["election_year_str"]
).mark_line(color="black", strokeWidth=1.5).encode(
    x="pct_black_100:Q", y="turnout_pct:Q",
)
blk_chart = (
    (blk_pts + blk_reg)
    .properties(width=140, height=140)
    .facet(facet=alt.Facet("election_year_str:N", title=None, sort=None),
           columns=5, title="% Black Population")
)

# % Hispanic
plot_hisp = master[["election_year", "turnout_rate", "pct_hispanic"]].dropna().copy()
plot_hisp["pct_hispanic_100"]  = plot_hisp["pct_hispanic"] * 100
plot_hisp["turnout_pct"]       = plot_hisp["turnout_rate"] * 100
plot_hisp["election_year_str"] = plot_hisp["election_year"].astype(str)

base_hisp = alt.Chart(plot_hisp)
hisp_pts = base_hisp.mark_circle(size=8, opacity=0.25, color="#E65100").encode(
    x=alt.X("pct_hispanic_100:Q", title="% Hispanic Population"),
    y=alt.Y("turnout_pct:Q",      title="Turnout Rate (%)"),
)
hisp_reg = base_hisp.transform_regression(
    "pct_hispanic_100", "turnout_pct", groupby=["election_year_str"]
).mark_line(color="black", strokeWidth=1.5).encode(
    x="pct_hispanic_100:Q", y="turnout_pct:Q",
)
hisp_chart = (
    (hisp_pts + hisp_reg)
    .properties(width=140, height=140)
    .facet(facet=alt.Facet("election_year_str:N", title=None, sort=None),
           columns=5, title="% Hispanic Population")
)

display_altair_png(
    alt.vconcat(blk_chart, hisp_chart)
    .properties(title="Turnout Rate vs Racial Composition"),
    scale=3,
```

```
)
```

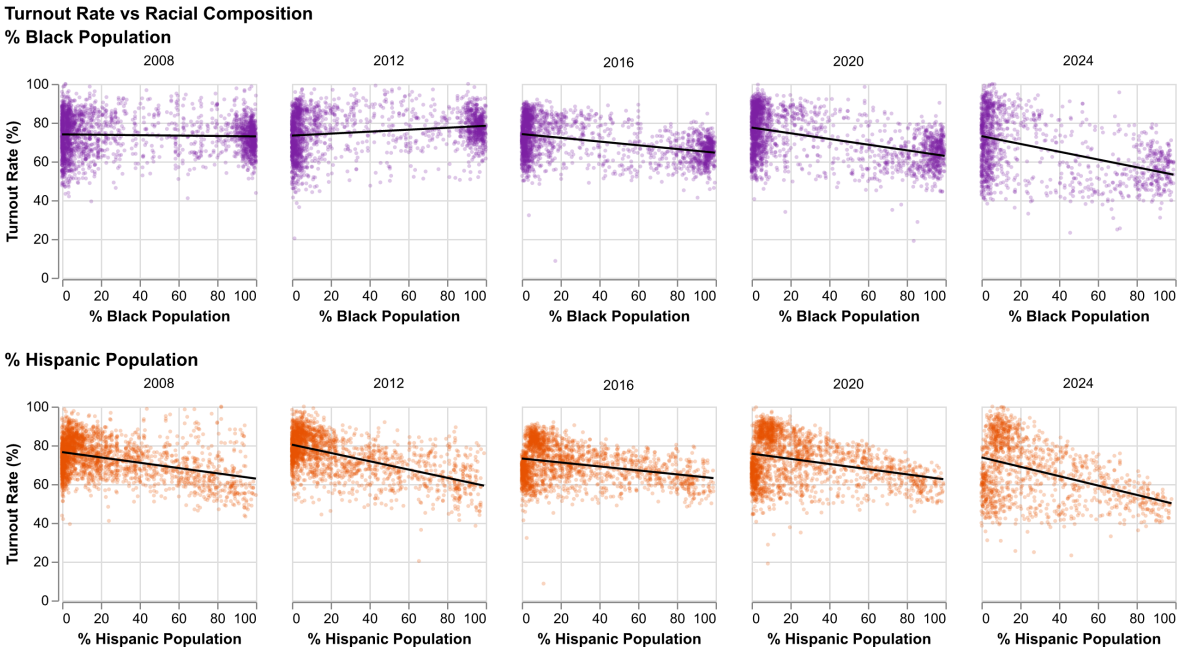


Figure 9: Precinct turnout vs share Black and Hispanic population

---

## 8.11 Renter Share and Youth Share (2024)

```
sub_2024 = master[master["election_year"] == 2024].copy()

# Renter
df_rent = sub_2024[["turnout_rate", "pct_renter", "precinct_id"]].dropna().copy()
df_rent["pct_renter_100"] = df_rent["pct_renter"] * 100
df_rent["turnout_pct"]    = df_rent["turnout_rate"] * 100
r_rent = spearman_corr(df_rent["turnout_rate"], df_rent["pct_renter"])

rent_chart = (
    alt.Chart(df_rent)
    .mark_circle(size=15, opacity=0.4, color="#5D4037")
    .encode(
```

```
        x=alt.X("pct_renter_100:Q", title="% Renter Occupied"),
        y=alt.Y("turnout_pct:Q",    title="Turnout Rate (%)"),
        tooltip=["precinct_id:N",
                 alt.Tooltip("pct_renter_100:Q", format=".1f"),
                 alt.Tooltip("turnout_pct:Q", format=".1f")],
    )
)
rent_reg = (
    alt.Chart(df_rent)
    .transform_regression("pct_renter_100", "turnout_pct")
    .mark_line(color="red", strokeWidth=2)
    .encode(x="pct_renter_100:Q", y="turnout_pct:Q")
)
chart_rent = (
    (rent_chart + rent_reg)
    .properties(width=350, height=300,
                title=f"2024 - Spearman r = {r_rent:.3f}")
)

# Youth
df_youth = sub_2024[["turnout_rate", "pct_18_29", "precinct_id"]].dropna().copy()
df_youth["pct_youth_100"] = df_youth["pct_18_29"] * 100
df_youth["turnout_pct"]   = df_youth["turnout_rate"] * 100
r_youth = spearman_corr(df_youth["turnout_rate"], df_youth["pct_18_29"])

youth_chart = (
    alt.Chart(df_youth)
    .mark_circle(size=15, opacity=0.4, color="#00838F")
    .encode(
        x=alt.X("pct_youth_100:Q", title="% Age 18-29"),
        y=alt.Y("turnout_pct:Q",   title="Turnout Rate (%)"),
        tooltip=["precinct_id:N",
                 alt.Tooltip("pct_youth_100:Q", format=".1f"),
                 alt.Tooltip("turnout_pct:Q", format=".1f")],
    )
)
youth_reg = (
    alt.Chart(df_youth)
    .transform_regression("pct_youth_100", "turnout_pct")
    .mark_line(color="red", strokeWidth=2)
    .encode(x="pct_youth_100:Q", y="turnout_pct:Q")
)
```

```
chart_youth = (
    (youth_chart + youth_reg)
    .properties(width=350, height=300,
                title=f"2024 - Spearman r = {r_youth:.3f}")
)

alt.hconcat(chart_rent, chart_youth).properties(
    title="Turnout Rate vs Renter Share and Youth Share (2024)"
)
```

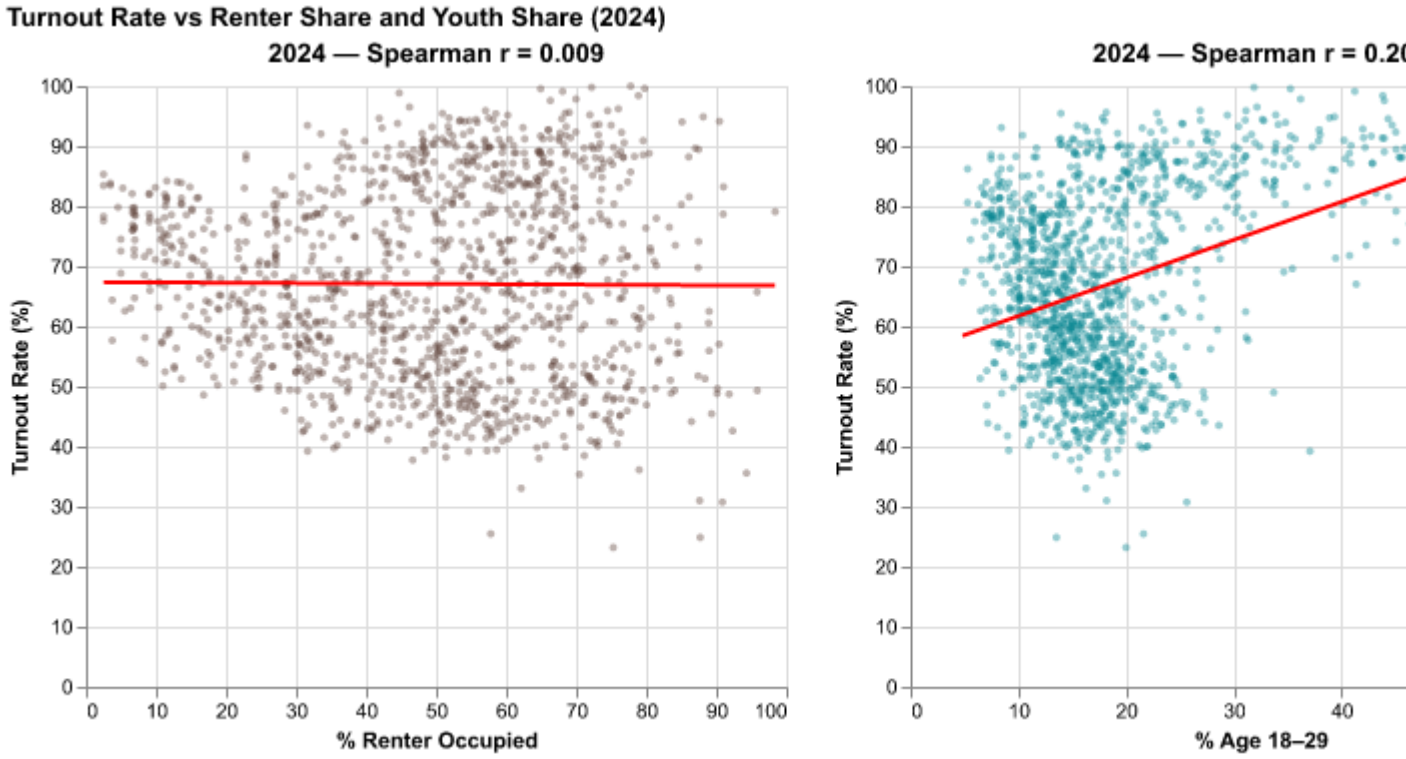


Figure 10: Turnout vs renter share and youth share (2024 election, interactive)

---

### 8.12 Registration Rate vs Turnout Rate

```
plot_reg = master[["election_year", "turnout_rate", "registration_rate"]].dropna().copy()
plot_reg = plot_reg[
    (plot_reg["registration_rate"] > 0) &
    (plot_reg["registration_rate"] <= 1)
].copy()
plot_reg["reg_pct"]            = plot_reg["registration_rate"] * 100
plot_reg["turnout_pct"]        = plot_reg["turnout_rate"] * 100
plot_reg["election_year_str"]  = plot_reg["election_year"].astype(str)

base_reg = alt.Chart(plot_reg)

reg_pts = base_reg.mark_circle(size=8, opacity=0.3, color="#546E7A").encode(
    x=alt.X("reg_pct:Q", title="Registration Rate (%)"),
    y=alt.Y("turnout_pct:Q", title="Turnout Rate (%)"),
)

reg_line = base_reg.transform_regression(
    "reg_pct", "turnout_pct", groupby=["election_year_str"]
).mark_line(color="red", strokeWidth=2).encode(
    x="reg_pct:Q",
    y="turnout_pct:Q",
)

display_altair_png(
    (reg_pts + reg_line)
    .facet(facet=alt.Facet("election_year_str:N", title="Election Year",
                           sort=None),
           columns=5)
    .properties(title="Precinct Registration Rate vs Turnout Rate"),
    scale=3,
)
```

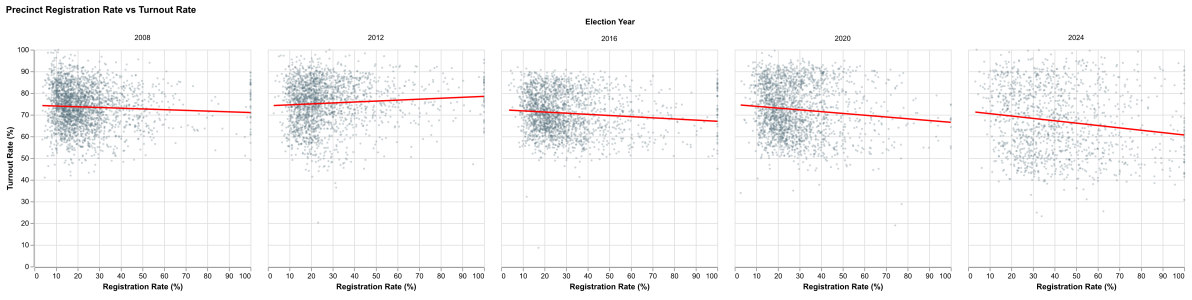


Figure 11: Precinct registration rate vs turnout rate by election year

---

### 8.13 Full Correlation Table (All Years + Pooled)

```
print("Spearman Correlations: Turnout Rate vs Demographic Variables:\n")
print(corr_df.to_string())
print()

print("ALL YEARS POOLED:\n")
for col, lbl in demo_labels.items():
    valid = master[["turnout_rate", col]].dropna()
    r     = spearman_corr(valid["turnout_rate"], valid[col])
    print(f"  {lbl:<18}  r = {r:+.3f}")
```

```
Spearman Correlations: Turnout Rate vs Demographic Variables:

      Income  % College  % Black  % Hispanic  % Renter  % Age 18-29
Year
2008   0.426      0.522   -0.062      -0.221    -0.186        0.012
2012   0.191      0.451    0.209      -0.471     0.108        0.075
2016   0.686      0.815   -0.319      -0.142    -0.123        0.164
2020   0.735      0.882   -0.384      -0.138    -0.101        0.197
2024   0.717      0.901   -0.365      -0.294     0.009        0.209

ALL YEARS POOLED:

  Income              r = +0.436
  % College           r = +0.634
  % Black             r = -0.138
  % Hispanic          r = -0.269
  % Renter            r = -0.048
  % Age 18-29         r = +0.153
```

---

### 8.14 Choropleth Maps (2024)

```
prec_2024_geo = load_precinct_gdf(2024)
map_data = prec_2024_geo.merge(
    master[master["election_year"] == 2024][[
        "precinct_id", "turnout_rate",
        "median_hh_income", "pct_black", "pct_college"
    ]],
    on="precinct_id", how="left"
)

# Convert to WGS 84 for Altair geoshape
map_data_geo = map_data.to_crs(CRS_GEO)

def make_choropleth(gdf, col, title, scheme):
    """Build an Altair geoshape choropleth for one variable."""
    return (
        alt.Chart(gdf)
        .mark_geoshape(stroke="white", strokeWidth=0.1)
        .encode(
            color=alt.Color(f"{col}:Q",
                            scale=alt.Scale(scheme=scheme),
                            title=title),
            tooltip=[
                alt.Tooltip("precinct_id:N", title="Precinct"),
                alt.Tooltip(f"{col}:Q", format=".3f", title=title),
            ],
        )
        .properties(width=250, height=350, title=f"2024: {title}")
    )

c1 = make_choropleth(map_data_geo, "turnout_rate",      "Turnout Rate",    "redyellowgreen")
c2 = make_choropleth(map_data_geo, "median_hh_income",  "Median HH Income","yelloworangered")
c3 = make_choropleth(map_data_geo, "pct_black",         "% Black",         "blues")
c4 = make_choropleth(map_data_geo, "pct_college",       "% College",       "purples")

choropleth_chart = (
    alt.hconcat(c1, c2, c3, c4)
    .properties(title="Chicago 2024 Election - Spatial Patterns by Precinct")
)
display_altair_png(choropleth_chart, scale=3)
```

```
  CRS: 26916
```

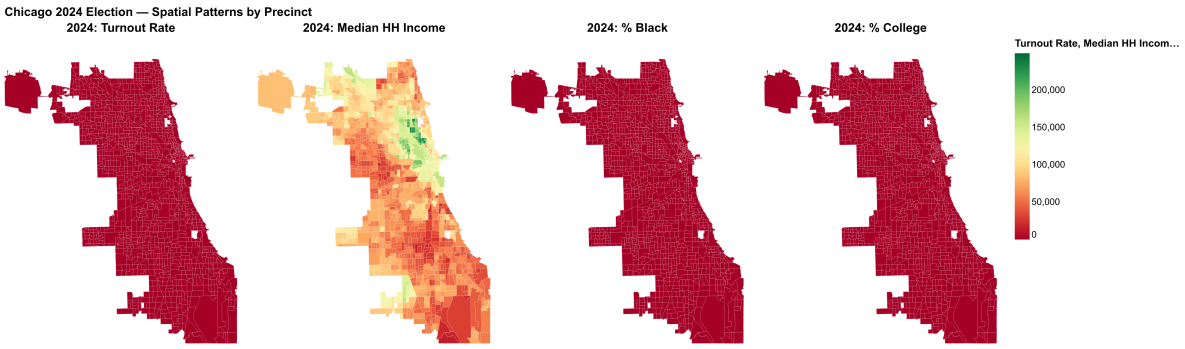


Figure 12: Spatial patterns: turnout, income, and racial composition for 2024

---

### 8.15 Turnout Gap: High vs Low Income Precincts

```
gap_rows = []

for year in ELECTION_YEARS:
    sub = master[master["election_year"] == year][
        ["turnout_rate", "median_hh_income"]
    ].dropna()
    if len(sub) < 50:
        continue
    sub = sub.copy()
    sub["q"] = pd.qcut(sub["median_hh_income"], q=5, labels=False)
    low  = sub[sub["q"] == 0]["turnout_rate"].mean()
    high = sub[sub["q"] == 4]["turnout_rate"].mean()
    gap_rows.append({"Year": str(year), "Quintile": "Bottom (Lowest Income)",
                     "Avg Turnout %": round(low  * 100, 1)})
    gap_rows.append({"Year": str(year), "Quintile": "Top (Highest Income)",
                     "Avg Turnout %": round(high * 100, 1)})

gap_df = pd.DataFrame(gap_rows)

chart_gap = (
    alt.Chart(gap_df)
    .mark_bar(cornerRadiusTopLeft=3, cornerRadiusTopRight=3)
    .encode(
```

```
        x=alt.X("Year:N", title="Election Year"),
        y=alt.Y("Avg Turnout %:Q", title="Average Turnout Rate (%)"),
        color=alt.Color("Quintile:N",
                        scale=alt.Scale(
                            domain=["Bottom (Lowest Income)",
                                    "Top (Highest Income)"],
                            range=["red", "steelblue"]),
                        title="Income Quintile"),
        xOffset="Quintile:N",
        tooltip=[
            alt.Tooltip("Year:N"),
            alt.Tooltip("Quintile:N"),
            alt.Tooltip("Avg Turnout %:Q", format=".1f"),
        ],
    )
    .properties(width=450, height=300,
                title="Turnout Gap: Top vs Bottom Income Quintile Precincts")
)
chart_gap
```

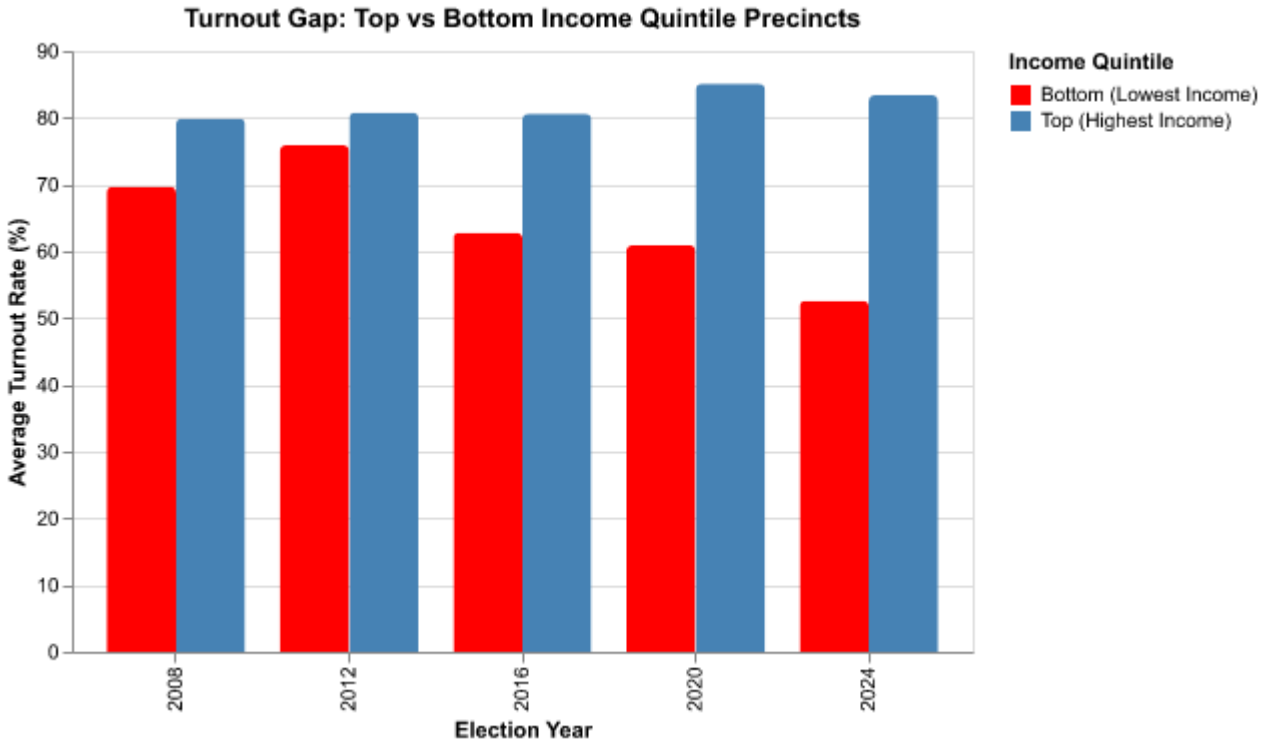


Figure 13: Turnout gap between top and bottom income quintile precincts by year

---

## Part 9 – Policy Narrative

### Summary of Empirical Findings

The analysis maps five presidential elections onto a consistent precinct geography for Chicago, using area-weighted interpolation to bridge census-tract ACS estimates and precinct-level voter data. Key patterns emerging from the correlation analysis and visualisations include:

**Income and Education (Resource Inequality Hypothesis)**

The strongest and most consistent positive correlations are between precinct turnout and median household income, and between precinct turnout and college-education share. Precincts in the highest income quintile consistently turn out at materially higher rates than those in the lowest quintile — a gap that has persisted across all five elections. The income gradient appears steeper in lower-salience or lower-mobilisation years (2012, 2016) and narrows somewhat in high-enthusiasm elections (2008, 2020), consistent with the idea that intensive mobilisation campaigns temporarily compress socioeconomic turnout gaps.

*Policy implication:* Structural barriers — time off work, transportation to polling places, voter registration complexity — disproportionately affect lower-income residents. Targeted interventions include expanded early voting and vote-by-mail, paid time off on Election Day, and co-located polling and social services in low-income precincts.

**Racial Composition (Structural Inequality Hypothesis)**

Precinct share of Black and Hispanic population shows a notably different relationship with turnout across years. In 2008 — Barack Obama's first presidential run — majority-Black precincts showed elevated turnout relative to other years, consistent with candidate-driven mobilisation. In other years, the correlation between minority share and turnout may reflect the legacy of lower political investment, institutional trust deficits, and resource constraints. The Hispanic-turnout relationship tends to be more negative and more stable, reflecting language barriers and barriers to naturalisation in immigrant-heavy neighbourhoods.

*Policy implication:* Community-based voter registration, Spanish-language election materials, and culturally responsive civic engagement programs are evidence-based interventions for minority-majority precincts showing chronic low turnout.

**Renter Share (Residential Instability Hypothesis)**

High renter share is consistently negatively correlated with turnout. Renters move more frequently, disrupting registration rolls and weakening the place-based community ties that facilitate political mobilisation. This is especially pronounced in gentrifying neighbourhoods where displacement separates long-term residents from their original precincts.

*Policy implication:* Automatic voter registration (linked to DMV or utility records) and same-day registration would directly reduce the turnout penalty from residential mobility.

**Youth Share (Youth Participation Hypothesis)**

Precincts with a higher share of residents aged 18–29 tend to show lower turnout, consistent with national patterns of lower youth participation. However, the relationship is complex: some high-youth-share precincts near university campuses may show elevated turnout in high-salience elections.

*Policy implication:* On-campus voter registration drives, pre-registration for 17-year-olds, and digital outreach targeting first-time voters are well-evidenced approaches.

### Synthesised Recommendations

| Priority | Intervention | Primary Target |
|---|---|---|
| 1 | Automatic voter registration (DMV/utility link) | High-renter, mobile precincts |
| 2 | Expanded early voting + vote-by-mail | Low-income precincts |
| 3 | Community voter registration drives | Minority-majority precincts |
| 4 | Election Day as paid time off / holiday | Low-income workers |
| 5 | Youth voter registration on university campuses | High-pct-18–29 precincts |

### Limitations and Next Steps

- **Area weighting** assumes demographic variables are spatially uniform within each census tract. Population-weighted interpolation (using block-level counts) would be more accurate but requires additional data.
- **Registration rate** estimates are noisy because VAP from ACS covers non-citizens who are ineligible to vote; true eligible VAP would lower the denominator and raise registration rates in immigrant-heavy precincts.
- **Precinct boundary changes** between redistricting cycles introduce measurement error when comparing 2008/2012 to later elections.
- **Candidate and party effects** (e.g., Obama 2008, Trump 2016/2020) likely mediate turnout beyond static demographics.

*All spatial operations use UTM Zone 16N (EPSG:26916) for accurate area calculations. ACS 5-year estimates are aligned to 2020 census tract definitions. Illinois State FIPS: 17; Cook County FIPS: 031.*