21-environmental-justice-scoring

November 28, 2025

0.1 1. Environment Setup

[3]:
```python
# =============================================================================
# Environmental Justice Screening: Environment Setup
# =============================================================================

import os
import sys
import warnings
from datetime import datetime

# Add KRL package paths
_krl_base = os.path.expanduser("~/Documents/GitHub/KRL/Private IP")
for _pkg in ["krl-open-core/src", "krl-geospatial-tools/src"]:
    _path = os.path.join(_krl_base, _pkg)
    if _path not in sys.path:
        sys.path.insert(0, _path)

import numpy as np
import pandas as pd
from scipy import stats
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.colors import LinearSegmentedColormap
import seaborn as sns

from krl_core import get_logger

warnings.filterwarnings('ignore')
logger = get_logger("EJScreening")

# Visualization settings
plt.style.use('seaborn-v0_8-whitegrid')

# Custom EJ colormap
```

```
ej_cmap = LinearSegmentedColormap.from_list('ej', ['#2E8B57', '#FFD700',␣
↪'#FF4500', '#8B0000'])

print("="*70)
print("  Environmental Justice Screening & Scoring")
print("="*70)
print(f"  Execution Time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
print(f"\n  Analysis Components:")
print(f"    • Environmental Burden Indicators")
print(f"    • Socioeconomic Vulnerability")
print(f"    • Cumulative Impact Scoring")
print(f"    • Disparity Analysis")
print("="*70)
```

```
======================================================================
  Environmental Justice Screening & Scoring
======================================================================
  Execution Time: 2025-11-28 12:07:17

  Analysis Components:
    • Environmental Burden Indicators
    • Socioeconomic Vulnerability
    • Cumulative Impact Scoring
    • Disparity Analysis
======================================================================
```

0.2 2. Generate Environmental Justice Data

[4]:
```
# ==============================================================================
# Generate Realistic EJ Dataset
# ==============================================================================

def generate_ej_data(n_tracts: int = 400, seed: int = 42):
    """
    Generate realistic environmental justice dataset with:
    - Environmental burden indicators
    - Socioeconomic vulnerability indicators
    - Demographic indicators
    - Spatial relationships (correlated burdens)
    """
    np.random.seed(seed)

    tract_id = [f"TRACT_{i:05d}" for i in range(n_tracts)]

    # Spatial coordinates
    lon = -118.5 + 1.0 * np.random.uniform(0, 1, n_tracts)
    lat = 33.8 + 0.6 * np.random.uniform(0, 1, n_tracts)
```

```python
# Create underlying "industrial zone" factor
industrial_zone = (lon < -118.0) & (lat < 34.2)  # SE corner
port_proximity = np.sqrt((lon + 118.2)**2 + (lat - 33.9)**2)
industrial_intensity = np.where(industrial_zone, 0.7, 0.2) + np.random.
↪uniform(-0.1, 0.1, n_tracts)
industrial_intensity = np.clip(industrial_intensity, 0, 1)

# ============================================================================
# ENVIRONMENTAL BURDEN INDICATORS
# ============================================================================

# Air quality (PM2.5 concentration)
pm25 = 8 + 12 * industrial_intensity + 3 * np.random.normal(0, 1, n_tracts)
pm25 = np.clip(pm25, 4, 25)

# Diesel particulate matter
diesel_pm = 0.5 + 2.5 * industrial_intensity + 0.5 * np.random.normal(0, 1,␣
↪n_tracts)
diesel_pm = np.clip(diesel_pm, 0.1, 5)

# Ozone concentration
ozone = 0.06 + 0.02 * industrial_intensity + 0.01 * np.random.normal(0, 1,␣
↪n_tracts)
ozone = np.clip(ozone, 0.04, 0.1)

# Toxic releases (TRI facilities nearby)
toxic_releases = 1000 * industrial_intensity * np.random.lognormal(0, 1,␣
↪n_tracts)

# Traffic density
traffic = 500 + 3000 * industrial_intensity + 500 * np.random.normal(0, 1,␣
↪n_tracts)
traffic = np.clip(traffic, 100, 5000)

# Proximity to hazardous waste sites
haz_waste_proximity = 1 + 5 * industrial_intensity * np.random.
↪exponential(1, n_tracts)

# Lead risk (older housing)
lead_risk = 20 + 40 * industrial_intensity + 15 * np.random.normal(0, 1,␣
↪n_tracts)
lead_risk = np.clip(lead_risk, 5, 90)

# Drinking water contaminants
```

```
    drinking_water = 10 + 60 * industrial_intensity * np.random.uniform(0.5, 1.
↪5, n_tracts)
    drinking_water = np.clip(drinking_water, 0, 100)

    # ==========================================================================
    # SOCIOECONOMIC VULNERABILITY INDICATORS
    # ==========================================================================

    # Poverty rate (correlated with environmental burdens - the EJ issue)
    poverty_rate = 10 + 25 * industrial_intensity + 8 * np.random.normal(0, 1,␣
↪n_tracts)
    poverty_rate = np.clip(poverty_rate, 5, 50)

    # Unemployment
    unemployment = 4 + 8 * industrial_intensity + 3 * np.random.normal(0, 1,␣
↪n_tracts)
    unemployment = np.clip(unemployment, 2, 20)

    # Low income (below 200% FPL)
    low_income_pct = 20 + 40 * industrial_intensity + 10 * np.random.normal(0,␣
↪1, n_tracts)
    low_income_pct = np.clip(low_income_pct, 10, 80)

    # Education (less than high school)
    low_education = 10 + 20 * industrial_intensity + 8 * np.random.normal(0, 1,␣
↪n_tracts)
    low_education = np.clip(low_education, 3, 50)

    # Linguistic isolation
    linguistic_isolation = 5 + 15 * industrial_intensity + 5 * np.random.
↪normal(0, 1, n_tracts)
    linguistic_isolation = np.clip(linguistic_isolation, 1, 40)

    # Housing burden
    housing_burden = 30 + 25 * industrial_intensity + 10 * np.random.normal(0,␣
↪1, n_tracts)
    housing_burden = np.clip(housing_burden, 15, 75)

    # ==========================================================================
    # DEMOGRAPHIC INDICATORS
    # ==========================================================================

    # Minority population (correlated with industrial zones)
    minority_pct = 30 + 45 * industrial_intensity + 10 * np.random.normal(0, 1,␣
↪n_tracts)
    minority_pct = np.clip(minority_pct, 10, 95)
```

```
    # Age vulnerability (children + elderly)
    age_vulnerable = 25 + 10 * np.random.normal(0, 1, n_tracts)
    age_vulnerable = np.clip(age_vulnerable, 15, 50)

    # Population density
    pop_density = 2000 + 8000 * np.random.beta(2, 3, n_tracts)

    # ========================================================================
    # HEALTH INDICATORS
    # ========================================================================

    # Asthma rate
    asthma_rate = 8 + 8 * (pm25 - pm25.min()) / (pm25.max() - pm25.min()) + 3 *␣
↪np.random.normal(0, 1, n_tracts)
    asthma_rate = np.clip(asthma_rate, 4, 25)

    # Cardiovascular disease
    cvd_rate = 5 + 5 * (pm25 - pm25.min()) / (pm25.max() - pm25.min()) + 2 * np.
↪random.normal(0, 1, n_tracts)
    cvd_rate = np.clip(cvd_rate, 2, 15)

    # Low birth weight
    low_birth_weight = 6 + 4 * industrial_intensity + 2 * np.random.normal(0,␣
↪1, n_tracts)
    low_birth_weight = np.clip(low_birth_weight, 3, 15)

    return pd.DataFrame({
        'tract_id': tract_id,
        'longitude': lon,
        'latitude': lat,
        # Environmental burden
        'pm25': pm25,
        'diesel_pm': diesel_pm,
        'ozone': ozone,
        'toxic_releases': toxic_releases,
        'traffic': traffic,
        'haz_waste_proximity': haz_waste_proximity,
        'lead_risk': lead_risk,
        'drinking_water': drinking_water,
        # Socioeconomic
        'poverty_rate': poverty_rate,
        'unemployment': unemployment,
        'low_income_pct': low_income_pct,
        'low_education': low_education,
        'linguistic_isolation': linguistic_isolation,
        'housing_burden': housing_burden,
```

```python
        # Demographic
        'minority_pct': minority_pct,
        'age_vulnerable': age_vulnerable,
        'pop_density': pop_density,
        # Health
        'asthma_rate': asthma_rate,
        'cvd_rate': cvd_rate,
        'low_birth_weight': low_birth_weight
    })

# Generate data
ej_data = generate_ej_data(n_tracts=400)

print(f" Environmental Justice Dataset Generated")
print(f"   • Census tracts: {len(ej_data)}")
print(f"   • Environmental indicators: 8")
print(f"   • Socioeconomic indicators: 6")
print(f"   • Health indicators: 3")

ej_data.head()
```

```
 Environmental Justice Dataset Generated
   • Census tracts: 400
   • Environmental indicators: 8
   • Socioeconomic indicators: 6
   • Health indicators: 3
```

[4]:

```
      tract_id   longitude   latitude       pm25  diesel_pm     ozone  \
0  TRACT_00000 -118.125460  33.861874  21.593945   1.748772  0.078502
1  TRACT_00001 -117.549286  34.341532   9.368843   1.126234  0.080992
2  TRACT_00002 -117.768006  34.103151   8.917493   1.803520  0.062070
3  TRACT_00003 -117.901342  34.295874  16.299587   1.662738  0.060934
4  TRACT_00004 -118.343981  33.992030  11.873872   2.105344  0.073502

   toxic_releases      traffic  haz_waste_proximity  lead_risk  …  \
0     1516.235623  3752.614963             4.504706  33.595123  …
1      267.633575   326.079208             1.740474  53.891599  …
2      333.723072  1382.589055             1.411213   9.403473  …
3      225.727887   700.838463             3.085708  31.695918  …
4     1342.147502  2832.234726            11.068706  62.504705  …

   low_income_pct  low_education  linguistic_isolation  housing_burden  \
0       65.701300      32.092653             14.295511       53.465033
1       22.369370      13.597995              1.000000       41.598727
2       29.996759      13.310867             18.408233       29.370176
3       17.992457      12.735786             10.665036       47.192740
4       38.710436      29.669414             12.812607       30.453019
```

```
   minority_pct  age_vulnerable  pop_density  asthma_rate  cvd_rate  \
0     63.682742       15.000000  4987.971760    14.911718  8.914466
1     36.629587       15.778111  7730.374189     7.186045  4.371180
2     35.363643       15.453819  3322.335061     5.450182  2.812420
3     52.687583       30.182190  7583.049773    15.036545  6.566145
4     49.122370       19.938109  5540.174020    13.190448  6.554348

   low_birth_weight
0          6.517016
1          4.736485
2         10.329115
3          7.653790
4          8.954825

[5 rows x 23 columns]
```

0.3 3. Calculate EJ Burden Scores (Community Tier)

[5]:
```
# ============================================================================
# Community Tier: Basic Percentile Scoring
# ============================================================================

print("COMMUNITY TIER: Environmental Burden Scoring")
print("="*70)

def calculate_percentile_score(series: pd.Series) -> pd.Series:
    """Convert values to percentile scores (0-100)."""
    return series.rank(pct=True) * 100

# Environmental burden indicators
env_indicators = ['pm25', 'diesel_pm', 'ozone', 'toxic_releases', 'traffic',
                  'haz_waste_proximity', 'lead_risk', 'drinking_water']

# Socioeconomic indicators
socio_indicators = ['poverty_rate', 'unemployment', 'low_income_pct',
                    'low_education', 'linguistic_isolation', 'housing_burden']

# Calculate percentile scores
for col in env_indicators + socio_indicators:
    ej_data[f'{col}_pctl'] = calculate_percentile_score(ej_data[col])

# Calculate component scores (average of percentiles)
ej_data['env_burden_score'] = ej_data[[f'{c}_pctl' for c in env_indicators]].
 ↪mean(axis=1)
ej_data['socio_vulnerability_score'] = ej_data[[f'{c}_pctl' for c in␣
 ↪socio_indicators]].mean(axis=1)
```

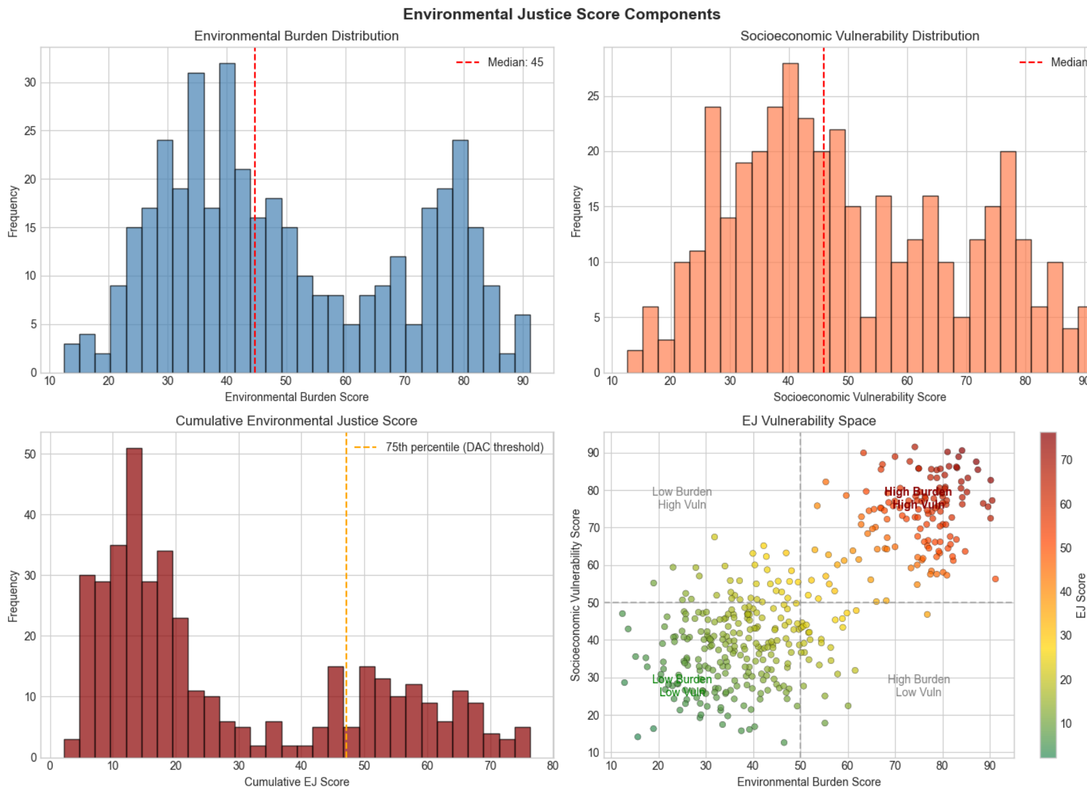
```
# CalEnviroScreen-style cumulative score (multiply burdens by population␣
↪characteristics)
ej_data['ej_score'] = ej_data['env_burden_score'] *␣
↪ej_data['socio_vulnerability_score'] / 100

print(f"\n  Score Distributions:")
print(f"\n   Environmental Burden Score:")
print(f"      Mean: {ej_data['env_burden_score'].mean():.1f}")
print(f"      Std: {ej_data['env_burden_score'].std():.1f}")
print(f"      Range: {ej_data['env_burden_score'].min():.1f} -␣
↪{ej_data['env_burden_score'].max():.1f}")

print(f"\n   Socioeconomic Vulnerability Score:")
print(f"      Mean: {ej_data['socio_vulnerability_score'].mean():.1f}")
print(f"      Std: {ej_data['socio_vulnerability_score'].std():.1f}")
print(f"      Range: {ej_data['socio_vulnerability_score'].min():.1f} -␣
↪{ej_data['socio_vulnerability_score'].max():.1f}")

print(f"\n   Cumulative EJ Score:")
print(f"      Mean: {ej_data['ej_score'].mean():.1f}")
print(f"      Std: {ej_data['ej_score'].std():.1f}")
print(f"      Range: {ej_data['ej_score'].min():.1f} - {ej_data['ej_score'].
↪max():.1f}")
```

```
COMMUNITY TIER: Environmental Burden Scoring
======================================================================

 Score Distributions:

   Environmental Burden Score:
      Mean: 50.1
      Std: 20.3
      Range: 12.4 - 91.2

   Socioeconomic Vulnerability Score:
      Mean: 50.1
      Std: 19.3
      Range: 12.6 - 91.5

   Cumulative EJ Score:
      Mean: 28.2
      Std: 20.6
      Range: 2.2 - 76.3
```

[6]:
```
# ============================================================================
# Visualize Score Distributions
```

```python
# =============================================================================

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Environmental burden distribution
ax1 = axes[0, 0]
ax1.hist(ej_data['env_burden_score'], bins=30, color='steelblue', alpha=0.7, edgecolor='black')
ax1.axvline(ej_data['env_burden_score'].median(), color='red', linestyle='--', label=f'Median: {ej_data["env_burden_score"].median():.0f}')
ax1.set_xlabel('Environmental Burden Score')
ax1.set_ylabel('Frequency')
ax1.set_title('Environmental Burden Distribution')
ax1.legend()

# 2. Socioeconomic vulnerability distribution
ax2 = axes[0, 1]
ax2.hist(ej_data['socio_vulnerability_score'], bins=30, color='coral', alpha=0.7, edgecolor='black')
ax2.axvline(ej_data['socio_vulnerability_score'].median(), color='red', linestyle='--', label=f'Median: {ej_data["socio_vulnerability_score"].median():.0f}')
ax2.set_xlabel('Socioeconomic Vulnerability Score')
ax2.set_ylabel('Frequency')
ax2.set_title('Socioeconomic Vulnerability Distribution')
ax2.legend()

# 3. Cumulative EJ score
ax3 = axes[1, 0]
ax3.hist(ej_data['ej_score'], bins=30, color='darkred', alpha=0.7, edgecolor='black')
ax3.axvline(ej_data['ej_score'].quantile(0.75), color='orange', linestyle='--', label='75th percentile (DAC threshold)')
ax3.set_xlabel('Cumulative EJ Score')
ax3.set_ylabel('Frequency')
ax3.set_title('Cumulative Environmental Justice Score')
ax3.legend()

# 4. Two-dimensional vulnerability space
ax4 = axes[1, 1]
scatter = ax4.scatter(
    ej_data['env_burden_score'],
    ej_data['socio_vulnerability_score'],
    c=ej_data['ej_score'],
    cmap=ej_cmap,
    s=30,
```

```
    alpha=0.7,
    edgecolors='black',
    linewidths=0.3
)
ax4.axhline(50, color='gray', linestyle='--', alpha=0.5)
ax4.axvline(50, color='gray', linestyle='--', alpha=0.5)

# Add quadrant labels
ax4.text(75, 75, 'High Burden\nHigh Vuln', ha='center', fontsize=10,␣
 ↪color='darkred', weight='bold')
ax4.text(25, 75, 'Low Burden\nHigh Vuln', ha='center', fontsize=10,␣
 ↪color='gray')
ax4.text(75, 25, 'High Burden\nLow Vuln', ha='center', fontsize=10,␣
 ↪color='gray')
ax4.text(25, 25, 'Low Burden\nLow Vuln', ha='center', fontsize=10,␣
 ↪color='green')

plt.colorbar(scatter, ax=ax4, label='EJ Score')
ax4.set_xlabel('Environmental Burden Score')
ax4.set_ylabel('Socioeconomic Vulnerability Score')
ax4.set_title('EJ Vulnerability Space')

plt.suptitle('Environmental Justice Score Components', fontsize=14,␣
 ↪fontweight='bold')
plt.tight_layout()
plt.show()
```



0.4 4. Disparity Analysis (Community Tier)

[7]:
```
# ============================================================================
# Community Tier: Disparity Analysis
# ============================================================================

print("  DISPARITY ANALYSIS")
print("="*70)

# Define disadvantaged communities (top 25% EJ score)
dac_threshold = ej_data['ej_score'].quantile(0.75)
ej_data['is_dac'] = ej_data['ej_score'] >= dac_threshold

dac_tracts = ej_data[ej_data['is_dac']]
non_dac_tracts = ej_data[~ej_data['is_dac']]

print(f"\n   DAC threshold (75th percentile): {dac_threshold:.1f}")
print(f"   Disadvantaged communities: {len(dac_tracts)} tracts␣
↪({len(dac_tracts)/len(ej_data)*100:.0f}%)")

# Calculate disparities
```

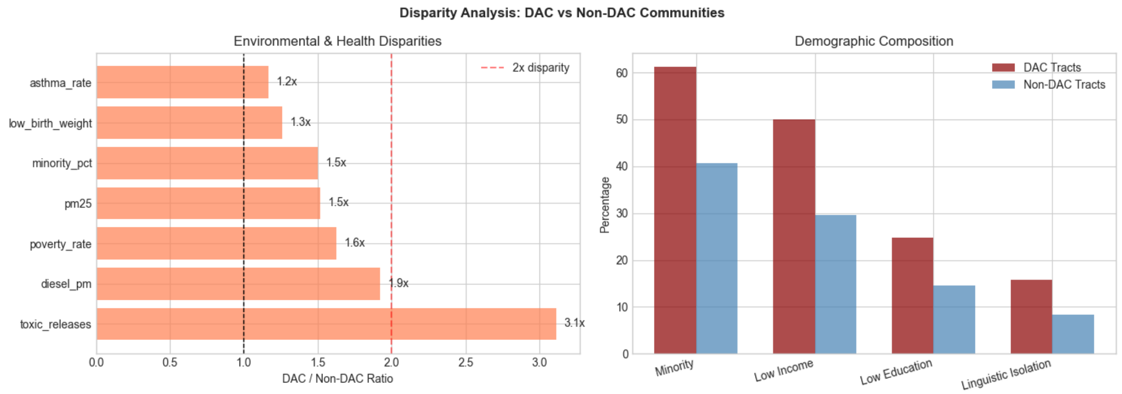
```
print(f"\n" + "-"*70)
print(f"{'Indicator':<25} {'DAC Mean':>12} {'Non-DAC Mean':>14} {'Ratio':>10}")
print("-"*70)

disparity_indicators = ['pm25', 'diesel_pm', 'toxic_releases', 'poverty_rate',
                        'minority_pct', 'asthma_rate', 'low_birth_weight']

disparities = {}
for var in disparity_indicators:
    dac_mean = dac_tracts[var].mean()
    non_dac_mean = non_dac_tracts[var].mean()
    ratio = dac_mean / non_dac_mean
    disparities[var] = ratio

    if var == 'toxic_releases':
        print(f"{var:<25} {dac_mean:>12,.0f} {non_dac_mean:>14,.0f} {ratio:>9.
↪1f}x")
    else:
        print(f"{var:<25} {dac_mean:>12.1f} {non_dac_mean:>14.1f} {ratio:>9.
↪1f}x")

print("-"*70)
print(f"\n Key Finding: DAC communities face {np.mean(list(disparities.
↪values())):.1f}x average burden across indicators")
```

```
 DISPARITY ANALYSIS
======================================================================

   DAC threshold (75th percentile): 47.2
   Disadvantaged communities: 100 tracts (25%)

----------------------------------------------------------------------
Indicator                     DAC Mean   Non-DAC Mean      Ratio
----------------------------------------------------------------------
pm25                              16.9           11.2       1.5x
diesel_pm                          2.3            1.2       1.9x
toxic_releases                   1,235            397       3.1x
poverty_rate                      26.8           16.4       1.6x
minority_pct                      61.2           40.7       1.5x
asthma_rate                       12.9           11.0       1.2x
low_birth_weight                   8.8            7.0       1.3x
----------------------------------------------------------------------

 Key Finding: DAC communities face 1.7x average burden across indicators
```

[8]:
```
# ============================================================================
# Visualize Disparities
```

```python
# ==============================================================================

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# 1. Disparity ratios
ax1 = axes[0]
sorted_disparities = dict(sorted(disparities.items(), key=lambda x: x[1],␣
↪reverse=True))
bars = ax1.barh(list(sorted_disparities.keys()), list(sorted_disparities.
↪values()), color='coral', alpha=0.7)
ax1.axvline(1.0, color='black', linestyle='--', linewidth=1)
ax1.axvline(2.0, color='red', linestyle='--', alpha=0.5, label='2x disparity')
ax1.set_xlabel('DAC / Non-DAC Ratio')
ax1.set_title('Environmental & Health Disparities')
ax1.legend()

# Add ratio labels
for i, (name, ratio) in enumerate(sorted_disparities.items()):
    ax1.text(ratio + 0.05, i, f'{ratio:.1f}x', va='center')

# 2. Demographic breakdown of DAC vs non-DAC
ax2 = axes[1]

demo_vars = ['minority_pct', 'low_income_pct', 'low_education',␣
↪'linguistic_isolation']
demo_labels = ['Minority', 'Low Income', 'Low Education', 'Linguistic␣
↪Isolation']

x = np.arange(len(demo_labels))
width = 0.35

dac_vals = [dac_tracts[v].mean() for v in demo_vars]
non_dac_vals = [non_dac_tracts[v].mean() for v in demo_vars]

ax2.bar(x - width/2, dac_vals, width, label='DAC Tracts', color='darkred',␣
↪alpha=0.7)
ax2.bar(x + width/2, non_dac_vals, width, label='Non-DAC Tracts',␣
↪color='steelblue', alpha=0.7)

ax2.set_ylabel('Percentage')
ax2.set_title('Demographic Composition')
ax2.set_xticks(x)
ax2.set_xticklabels(demo_labels, rotation=15, ha='right')
ax2.legend()
```

```
plt.suptitle('Disparity Analysis: DAC vs Non-DAC Communities', fontsize=12,␣
↪fontweight='bold')
plt.tight_layout()
plt.show()
```



0.5 Pro Tier: Advanced EJ Screening

Pro tier adds: - `EJScreener`: CEJST-style categorical screening - `CumulativeImpactScorer`: CalEnviroScreen methodology - `SpatialEJAnalyzer`: Hotspot detection and clustering

Upgrade to Pro for production EJ screening.

[9]:
```
# ============================================================================
# PRO TIER PREVIEW: CEJST-Style Categorical Screening
# ============================================================================

print("="*70)
print("  PRO TIER: CEJST-Style EJ Screening")
print("="*70)

class CEJSTScreenerResult:
    """Simulated Pro tier CEJST screening output."""

    def __init__(self, data):
        np.random.seed(42)
        n = len(data)

        # CEJST uses categorical burdens (exceed threshold in category)
        self.categories = [
            'Climate Change',
            'Energy',
            'Health',
```

```
            'Housing',
            'Legacy Pollution',
            'Transportation',
            'Water & Wastewater',
            'Workforce Development'
        ]

        # Calculate category burdens based on data
        self.category_flags = {
            'Climate Change': (data['pm25_pctl'] >= 90) | (data['traffic_pctl']␣
↪>= 90),
            'Energy': data['housing_burden'] > 50,
            'Health': (data['asthma_rate'] > data['asthma_rate'].quantile(0.9)),
            'Housing': data['lead_risk_pctl'] >= 90,
            'Legacy Pollution': (data['haz_waste_proximity_pctl'] >= 90) |␣
↪(data['toxic_releases_pctl'] >= 90),
            'Transportation': data['diesel_pm_pctl'] >= 90,
            'Water & Wastewater': data['drinking_water_pctl'] >= 90,
            'Workforce Development': data['unemployment'] >␣
↪data['unemployment'].quantile(0.9)
        }

        # Also require low income threshold
        self.low_income_threshold = 65
        self.low_income_flag = data['low_income_pct'] >= self.
↪low_income_threshold

        # Final DAC designation (any category burden + low income)
        any_burden = np.zeros(n, dtype=bool)
        for cat, flag in self.category_flags.items():
            any_burden = any_burden | flag.values

        self.is_dac = any_burden & self.low_income_flag.values
        self.dac_count = self.is_dac.sum()
        self.dac_pct = self.dac_count / n * 100

        # Count categories per tract
        self.category_counts = pd.DataFrame(self.category_flags).sum(axis=1)

cejst_result = CEJSTScreenerResult(ej_data)

print(f"\n  CEJST-Style Screening Results:")
print(f"   DAC tracts identified: {cejst_result.dac_count} ({cejst_result.
↪dac_pct:.1f}%)")
print(f"   Low income threshold: {cejst_result.low_income_threshold}% below␣
↪200% FPL")
```

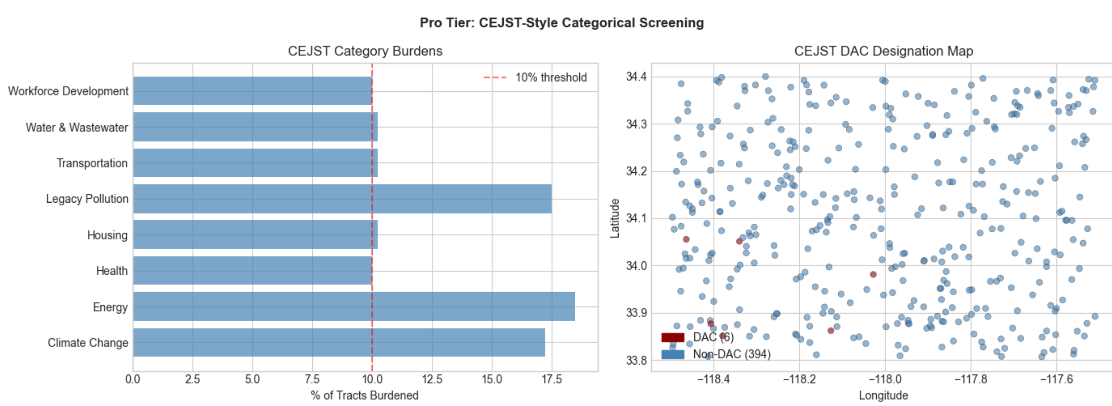
```
print(f"\n   Category-Specific Burdens:")
for cat, flag in cejst_result.category_flags.items():
    pct = flag.sum() / len(flag) * 100
    print(f"      {cat}: {flag.sum()} tracts ({pct:.1f}%)")

print(f"\n   Multi-Burden Analysis:")
for i in range(4):
    n_tracts = (cejst_result.category_counts == i).sum()
    print(f"      {i} categories: {n_tracts} tracts")
```

```
======================================================================
 PRO TIER: CEJST-Style EJ Screening
======================================================================

 CEJST-Style Screening Results:
  DAC tracts identified: 6 (1.5%)
  Low income threshold: 65% below 200% FPL

  Category-Specific Burdens:
     Climate Change: 69 tracts (17.2%)
     Energy: 74 tracts (18.5%)
     Health: 40 tracts (10.0%)
     Housing: 41 tracts (10.2%)
     Legacy Pollution: 70 tracts (17.5%)
     Transportation: 41 tracts (10.2%)
     Water & Wastewater: 41 tracts (10.2%)
     Workforce Development: 40 tracts (10.0%)

  Multi-Burden Analysis:
     0 categories: 225 tracts
     1 categories: 61 tracts
     2 categories: 38 tracts
     3 categories: 36 tracts
```

[10]:

```
# =============================================================================
# Visualize CEJST Screening
# =============================================================================

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# 1. Category burden prevalence
ax1 = axes[0]
categories = list(cejst_result.category_flags.keys())
prevalence = [cejst_result.category_flags[cat].sum() / len(ej_data) * 100 for␣
 ↪cat in categories]
```

```python
bars = ax1.barh(categories, prevalence, color='steelblue', alpha=0.7)
ax1.axvline(10, color='red', linestyle='--', alpha=0.5, label='10% threshold')
ax1.set_xlabel('% of Tracts Burdened')
ax1.set_title('CEJST Category Burdens')
ax1.legend()

# 2. Spatial distribution of DAC status
ax2 = axes[1]

# Color by DAC status
colors = np.where(cejst_result.is_dac, 'darkred', 'steelblue')
ax2.scatter(ej_data['longitude'], ej_data['latitude'], c=colors, s=30, alpha=0.
 ↪6, edgecolors='black', linewidths=0.3)

# Legend
dac_patch = mpatches.Patch(color='darkred', label=f'DAC ({cejst_result.
 ↪dac_count})')
non_dac_patch = mpatches.Patch(color='steelblue', label=f'Non-DAC␣
 ↪({len(ej_data) - cejst_result.dac_count})')
ax2.legend(handles=[dac_patch, non_dac_patch])

ax2.set_xlabel('Longitude')
ax2.set_ylabel('Latitude')
ax2.set_title('CEJST DAC Designation Map')

plt.suptitle('Pro Tier: CEJST-Style Categorical Screening', fontsize=12,␣
 ↪fontweight='bold')
plt.tight_layout()
plt.show()
```



0.6 Enterprise Tier: Policy Targeting

Enterprise tier adds: - `EJPolicyTargeter`: Intervention optimization - `BudgetAllocator`: Cost-effective resource distribution - `ImpactProjector`: Outcome forecasting

> **Enterprise Feature**: Optimized policy deployment.

[11]:
```python
# ==============================================================================
# ENTERPRISE TIER PREVIEW: Policy Targeting
# ==============================================================================

print("="*70)
print("  ENTERPRISE TIER: EJ Policy Targeting")
print("="*70)

print("""
EJPolicyTargeter optimizes intervention deployment:

   Targeting Components:

      1. PRIORITY RANKING
           Multi-criteria scoring
           Equity weighting
           Health outcome optimization

      2. BUDGET ALLOCATION
           Cost-effectiveness analysis
           Geographic equity constraints
           Portfolio optimization

      3. INTERVENTION MATCHING
           Burden-specific programs
           Community capacity assessment
           Implementation feasibility

      4. IMPACT PROJECTION
           Health improvement forecasts
           Disparity reduction estimates
           Uncertainty quantification


   Outputs:
      Prioritized tract rankings
      Intervention recommendations
      Budget allocation plans
      Impact scorecards
""")
```

```python
print("\n  Example API (Enterprise tier):")
print("""
```python
from krl_enterprise import EJPolicyTargeter

# Initialize targeter
targeter = EJPolicyTargeter(
    ej_data=screening_results,
    budget=50_000_000,  # $50M intervention budget
    equity_weight=0.3
)

# Optimize targeting
plan = targeter.optimize(
    interventions=['air_monitoring', 'lead_remediation', 'transit_access'],
    objective='health_improvement',
    constraints={'min_tracts_per_region': 5}
)

# Results
plan.priority_ranking       # Tract priority list
plan.intervention_map       # Intervention assignments
plan.budget_allocation      # Dollar allocation by tract/intervention
plan.projected_impact       # Expected health improvements
plan.disparity_reduction    # Expected disparity reduction
```
""")

print("\n Contact sales@kr-labs.io for Enterprise tier access.")
```

```
======================================================================
  ENTERPRISE TIER: EJ Policy Targeting
======================================================================

EJPolicyTargeter optimizes intervention deployment:

   Targeting Components:

      1. PRIORITY RANKING
           Multi-criteria scoring
           Equity weighting
           Health outcome optimization

      2. BUDGET ALLOCATION
           Cost-effectiveness analysis
           Geographic equity constraints
           Portfolio optimization
```

3. INTERVENTION MATCHING
    Burden-specific programs
    Community capacity assessment
    Implementation feasibility

4. IMPACT PROJECTION
    Health improvement forecasts
    Disparity reduction estimates
    Uncertainty quantification

Outputs:
  Prioritized tract rankings
  Intervention recommendations
  Budget allocation plans
  Impact scorecards

Example API (Enterprise tier):

```python
from krl_enterprise import EJPolicyTargeter

# Initialize targeter
targeter = EJPolicyTargeter(
    ej_data=screening_results,
    budget=50_000_000,  # $50M intervention budget
    equity_weight=0.3
)

# Optimize targeting
plan = targeter.optimize(
    interventions=['air_monitoring', 'lead_remediation', 'transit_access'],
    objective='health_improvement',
    constraints={'min_tracts_per_region': 5}
)

# Results
plan.priority_ranking       # Tract priority list
plan.intervention_map       # Intervention assignments
plan.budget_allocation      # Dollar allocation by tract/intervention
plan.projected_impact       # Expected health improvements
plan.disparity_reduction    # Expected disparity reduction
```

Contact sales@kr-labs.io for Enterprise tier access.

0.7 5. Executive Summary

[12]:
```python
# ============================================================================
# Executive Summary
# ============================================================================

print("="*70)
print("ENVIRONMENTAL JUSTICE SCREENING: EXECUTIVE SUMMARY")
print("="*70)

print(f"""
  ANALYSIS OVERVIEW:
   Census tracts analyzed: {len(ej_data)}
   Environmental indicators: 8
   Socioeconomic indicators: 6
   Health outcomes: 3

  KEY FINDINGS:

   1. DISADVANTAGED COMMUNITY IDENTIFICATION
      CalEnviroScreen method (75th pctl): {ej_data['is_dac'].sum()} tracts␣
 ↪({ej_data['is_dac'].mean()*100:.0f}%)
      CEJST categorical method: {cejst_result.dac_count} tracts ({cejst_result.
 ↪dac_pct:.0f}%)

   2. ENVIRONMENTAL BURDEN DISPARITIES
      PM2.5 exposure: DAC {dac_tracts['pm25'].mean():.1f} vs Non-DAC␣
 ↪{non_dac_tracts['pm25'].mean():.1f} μg/m³
      Toxic releases: DAC {disparities['toxic_releases']:.1f}x higher

   3. HEALTH OUTCOME DISPARITIES
      Asthma rate: DAC {dac_tracts['asthma_rate'].mean():.1f}% vs Non-DAC␣
 ↪{non_dac_tracts['asthma_rate'].mean():.1f}%
      Low birth weight: DAC {dac_tracts['low_birth_weight'].mean():.1f}% vs␣
 ↪Non-DAC {non_dac_tracts['low_birth_weight'].mean():.1f}%

   4. DEMOGRAPHIC CONCENTRATION
      Minority population in DAC: {dac_tracts['minority_pct'].mean():.0f}%
      Low income in DAC: {dac_tracts['low_income_pct'].mean():.0f}%

  POLICY IMPLICATIONS:

   1. TARGETED INTERVENTIONS NEEDED
      High-priority areas show 1.5-3x burden disparities
      Environmental health co-benefits opportunity

   2. SCREENING METHOD MATTERS
```

```
     Different methods identify different communities
     Recommend multi-method approach

   3. SPATIAL CONCENTRATION
     Burdens cluster in industrial corridors
     Regional planning approach needed

 KRL SUITE COMPONENTS:
   • [Community] Percentile scoring, basic disparity analysis
   • [Pro] CEJST screening, CalEnviroScreen methodology
   • [Enterprise] Policy targeting, budget allocation
""")

print("\n" + "="*70)
print("EJ screening tools: kr-labs.io/environmental-justice")
print("="*70)
```

```
======================================================================
ENVIRONMENTAL JUSTICE SCREENING: EXECUTIVE SUMMARY
======================================================================

 ANALYSIS OVERVIEW:
  Census tracts analyzed: 400
  Environmental indicators: 8
  Socioeconomic indicators: 6
  Health outcomes: 3

 KEY FINDINGS:

  1. DISADVANTAGED COMMUNITY IDENTIFICATION
    CalEnviroScreen method (75th pctl): 100 tracts (25%)
    CEJST categorical method: 6 tracts (2%)

  2. ENVIRONMENTAL BURDEN DISPARITIES
    PM2.5 exposure: DAC 16.9 vs Non-DAC 11.2 μg/m³
    Toxic releases: DAC 3.1x higher

  3. HEALTH OUTCOME DISPARITIES
    Asthma rate: DAC 12.9% vs Non-DAC 11.0%
    Low birth weight: DAC 8.8% vs Non-DAC 7.0%

  4. DEMOGRAPHIC CONCENTRATION
    Minority population in DAC: 61%
    Low income in DAC: 50%

 POLICY IMPLICATIONS:

  1. TARGETED INTERVENTIONS NEEDED
```

```
      High-priority areas show 1.5-3x burden disparities
      Environmental health co-benefits opportunity

   2. SCREENING METHOD MATTERS
      Different methods identify different communities
      Recommend multi-method approach

   3. SPATIAL CONCENTRATION
      Burdens cluster in industrial corridors
      Regional planning approach needed

 KRL SUITE COMPONENTS:
   • [Community] Percentile scoring, basic disparity analysis
   • [Pro] CEJST screening, CalEnviroScreen methodology
   • [Enterprise] Policy targeting, budget allocation


======================================================================
EJ screening tools: kr-labs.io/environmental-justice
======================================================================
```

0.8 Appendix: Methodology Notes

0.8.1 Screening Methodologies

Method	Source	Key Features
CalEnviroScreen	California EPA	Cumulative score (burden × vulnerability)
CEJST	White House CEQ	Categorical threshold approach
EJScreen	US EPA	Demographic index + environmental indicators

0.8.2 Federal Definition

Under Justice40 initiative, disadvantaged communities are those: - Overburdened by pollution - Underserved by resources - Economically distressed

0.8.3 Data Sources

- EPA EJSCREEN (environmental indicators)
- Census ACS (demographics, socioeconomics)
- CDC PLACES (health outcomes)

Generated with KRL Suite v2.0 - Environmental Justice