# 21-environmental-justice-scoring

November 29, 2025

### 0.1 1. Environment Setup

[1]:
```
# ==============================================================================
# Environmental Justice Screening: Environment Setup
# ==============================================================================

import os
import sys
import warnings
from datetime import datetime
from dotenv import load_dotenv

# Load environment variables
_env_path = os.path.expanduser("~/Documents/GitHub/KRL/Private IP/krl-tutorials/
↪.env")
load_dotenv(_env_path)

# 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",␣
↪"krl-data-connectors/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
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```python
from krl_core import get_logger

# Import Professional FRED connector
from krl_data_connectors.professional.fred_full import FREDFullConnector
from krl_data_connectors import skip_license_check

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 (Real FRED Data)")
print(f"   • Cumulative Impact Scoring")
print(f"   • Disparity Analysis")
print(f"\n Data Source: FRED Professional (County Economics)")
print("="*70)
```

```
======================================================================
 Environmental Justice Screening & Scoring
======================================================================
 Execution Time: 2025-11-29 12:19:11

 Analysis Components:
   • Environmental Burden Indicators
   • Socioeconomic Vulnerability (Real FRED Data)
   • Cumulative Impact Scoring
   • Disparity Analysis

 Data Source: FRED Professional (County Economics)
======================================================================
```

## 0.2 2. Fetch Real Socioeconomic Data from FRED

We use real county-level unemployment data as a vulnerability indicator for environmental justice analysis. Environmental burden indicators are simulated based on real economic vulnerability patterns.

[2]:

```python
# ==============================================================================
# Fetch Real Socioeconomic Data and Build EJ Dataset
# ==============================================================================

# Initialize FRED connector with Professional tier license skip
fred = FREDFullConnector(api_key="SHOWCASE-KEY")
skip_license_check(fred)
fred.fred_api_key = os.getenv('FRED_API_KEY')
fred._init_session()

# California county FIPS codes for EJ analysis (LA Basin area)
ca_counties = {
    '037': ('Los Angeles', -118.24, 34.05),
    '059': ('Orange', -117.83, 33.71),
    '065': ('Riverside', -116.97, 33.95),
    '071': ('San Bernardino', -117.29, 34.84),
    '111': ('Ventura', -119.22, 34.35),
    '029': ('Kern', -119.02, 35.35),
    '025': ('Imperial', -115.36, 32.79),
    '073': ('San Diego', -117.16, 32.72),
    '079': ('San Luis Obispo', -120.45, 35.34),
    '083': ('Santa Barbara', -119.85, 34.71),
    '019': ('Fresno', -119.77, 36.75),
    '099': ('Stanislaus', -120.99, 37.56),
    '077': ('San Joaquin', -121.27, 37.93),
    '047': ('Merced', -120.48, 37.19),
    '039': ('Madera', -119.76, 37.22),
    '107': ('Tulare', -118.80, 36.23),
    '031': ('Kings', -119.82, 36.07)
}

# Fetch unemployment data for 2022-2023
print(" Fetching California county unemployment data from FRED...")
records = []

for county_code, (county_name, lon, lat) in ca_counties.items():
    try:
        series_id = f'LAUCN06{county_code}0000000003A'
        series_data = fred.get_series(series_id, start_date='2022-01-01',␣
 ↪end_date='2023-12-31')

        if series_data is not None and not series_data.empty:
            series_data.index = pd.to_datetime(series_data.index)
            ur_2023 = series_data[series_data.index.year == 2023]['value'].
 ↪mean()

            if not pd.isna(ur_2023):
```

```python
                records.append({
                    'tract_id': f'CA_{county_code}',
                    'county_name': county_name,
                    'longitude': lon,
                    'latitude': lat,
                    'unemployment_rate': float(ur_2023)
                })
    except Exception as e:
        pass

print(f"     Retrieved data for {len(records)} California counties")

# Create base dataset
ej_data = pd.DataFrame(records)

# Use unemployment as base vulnerability indicator
# Higher unemployment correlates with higher vulnerability
np.random.seed(42)

# Scale unemployment to create vulnerability index
ej_data['vulnerability_base'] = (ej_data['unemployment_rate'] -
 ↪ej_data['unemployment_rate'].min()) / \
                                  (ej_data['unemployment_rate'].max() -
 ↪ej_data['unemployment_rate'].min())

# ============================================================================
# ENVIRONMENTAL BURDEN INDICATORS (simulated based on vulnerability patterns)
# Industrial areas tend to be in lower-income communities
# ============================================================================

industrial_intensity = ej_data['vulnerability_base'] * 0.7 + np.random.
 ↪uniform(-0.1, 0.1, len(ej_data))
industrial_intensity = np.clip(industrial_intensity, 0, 1)

# Air quality (PM2.5 concentration)
ej_data['pm25'] = 8 + 12 * industrial_intensity + 3 * np.random.normal(0, 1,
 ↪len(ej_data))
ej_data['pm25'] = np.clip(ej_data['pm25'], 4, 25)

# Diesel particulate matter
ej_data['diesel_pm'] = 0.5 + 2.5 * industrial_intensity + 0.5 * np.random.
 ↪normal(0, 1, len(ej_data))
ej_data['diesel_pm'] = np.clip(ej_data['diesel_pm'], 0.1, 5)

# Ozone concentration
ej_data['ozone'] = 0.06 + 0.02 * industrial_intensity + 0.01 * np.random.
 ↪normal(0, 1, len(ej_data))
```

```python
ej_data['ozone'] = np.clip(ej_data['ozone'], 0.04, 0.1)

# Toxic releases
ej_data['toxic_releases'] = 1000 * industrial_intensity * np.random.
↪lognormal(0, 1, len(ej_data))

# Traffic density
ej_data['traffic'] = 500 + 3000 * industrial_intensity + 500 * np.random.
↪normal(0, 1, len(ej_data))
ej_data['traffic'] = np.clip(ej_data['traffic'], 100, 5000)

# Hazardous waste proximity
ej_data['haz_waste_proximity'] = 1 + 5 * industrial_intensity * np.random.
↪exponential(1, len(ej_data))

# Lead risk
ej_data['lead_risk'] = 20 + 40 * industrial_intensity + 15 * np.random.
↪normal(0, 1, len(ej_data))
ej_data['lead_risk'] = np.clip(ej_data['lead_risk'], 5, 90)

# Drinking water contaminants
ej_data['drinking_water'] = 10 + 60 * industrial_intensity * np.random.
↪uniform(0.5, 1.5, len(ej_data))
ej_data['drinking_water'] = np.clip(ej_data['drinking_water'], 0, 100)

# ==========================================================================
# SOCIOECONOMIC VULNERABILITY INDICATORS (based on real unemployment)
# ==========================================================================

# Poverty rate (correlated with unemployment)
ej_data['poverty_rate'] = 10 + 25 * ej_data['vulnerability_base'] + 8 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['poverty_rate'] = np.clip(ej_data['poverty_rate'], 5, 50)

# Keep real unemployment
ej_data['unemployment'] = ej_data['unemployment_rate']

# Low income
ej_data['low_income_pct'] = 20 + 40 * ej_data['vulnerability_base'] + 10 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['low_income_pct'] = np.clip(ej_data['low_income_pct'], 10, 80)

# Education
ej_data['low_education'] = 10 + 20 * ej_data['vulnerability_base'] + 8 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['low_education'] = np.clip(ej_data['low_education'], 3, 50)
```

```
# Linguistic isolation
ej_data['linguistic_isolation'] = 5 + 15 * ej_data['vulnerability_base'] + 5 *␣
↪np.random.normal(0, 1, len(ej_data))
ej_data['linguistic_isolation'] = np.clip(ej_data['linguistic_isolation'], 1,␣
↪40)

# Housing burden
ej_data['housing_burden'] = 30 + 25 * ej_data['vulnerability_base'] + 10 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['housing_burden'] = np.clip(ej_data['housing_burden'], 15, 75)

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

# Minority population percentage
ej_data['minority_pct'] = 30 + 40 * ej_data['vulnerability_base'] + 15 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['minority_pct'] = np.clip(ej_data['minority_pct'], 10, 95)

# Age groups
ej_data['children_pct'] = 20 + 10 * ej_data['vulnerability_base'] + 5 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['children_pct'] = np.clip(ej_data['children_pct'], 10, 40)

ej_data['elderly_pct'] = 12 + 8 * ej_data['vulnerability_base'] + 4 * np.random.
↪normal(0, 1, len(ej_data))
ej_data['elderly_pct'] = np.clip(ej_data['elderly_pct'], 5, 30)

# Health indicators
ej_data['asthma_rate'] = 8 + 7 * ej_data['vulnerability_base'] + 2 * np.random.
↪normal(0, 1, len(ej_data))
ej_data['asthma_rate'] = np.clip(ej_data['asthma_rate'], 5, 20)

ej_data['heart_disease_rate'] = 5 + 5 * ej_data['vulnerability_base'] + 2 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['heart_disease_rate'] = np.clip(ej_data['heart_disease_rate'], 3, 15)

# Low birth weight rate
ej_data['low_birth_weight'] = 6 + 5 * ej_data['vulnerability_base'] + 1.5 * np.
↪random.normal(0, 1, len(ej_data))
ej_data['low_birth_weight'] = np.clip(ej_data['low_birth_weight'], 4, 15)

# Population
ej_data['population'] = np.random.lognormal(11, 1.5, len(ej_data)).astype(int)
```

```python
print(f"\n Environmental Justice Dataset (Real FRED + Simulated)")
print(f"   • Counties: {len(ej_data)}")
print(f"   • Region: Southern California / Central Valley")
print(f"   • Avg unemployment (real): {ej_data['unemployment'].mean():.1f}%")
print(f"   • Avg PM2.5 (simulated): {ej_data['pm25'].mean():.1f} g/m³")
print(f"   • Avg poverty rate: {ej_data['poverty_rate'].mean():.1f}%")

ej_data.head()
```

{"timestamp": "2025-11-29T17:19:11.545276Z", "level": "INFO", "name": "FREDFullConnector", "message": "Connector initialized", "source": {"file": "base_connector.py", "line": 81, "function": "__init__"}, "levelname": "INFO", "taskName": "Task-3", "connector": "FREDFullConnector", "cache_dir": "/Users/bcdelo/.krl_cache/fredfullconnector", "cache_ttl": 3600, "has_api_key": true}

{"timestamp": "2025-11-29T17:19:11.546013Z", "level": "INFO", "name": "FREDFullConnector", "message": "Connector initialized", "source": {"file": "base_connector.py", "line": 81, "function": "__init__"}, "levelname": "INFO", "taskName": "Task-3", "connector": "FREDFullConnector", "cache_dir": "/Users/bcdelo/.krl_cache/fredfullconnector", "cache_ttl": 3600, "has_api_key": true}

{"timestamp": "2025-11-29T17:19:11.546172Z", "level": "INFO", "name": "krl_data_connectors.licensed_connector_mixin", "message": "Licensed connector initialized: FRED_Full", "source": {"file": "licensed_connector_mixin.py", "line": 198, "function": "__init__"}, "levelname": "INFO", "taskName": "Task-3", "connector": "FRED_Full", "required_tier": "PROFESSIONAL", "has_api_key": true}

{"timestamp": "2025-11-29T17:19:11.546329Z", "level": "INFO", "name": "FREDFullConnector", "message": "Initialized FRED Full connector (Professional tier)", "source": {"file": "fred_full.py", "line": 102, "function": "__init__"}, "levelname": "INFO", "taskName": "Task-3", "connector": "FRED_Full"}

{"timestamp": "2025-11-29T17:19:11.546516Z", "level": "WARNING", "name": "krl_data_connectors.licensed_connector_mixin", "message": "License checking DISABLED for FREDFullConnector. This should ONLY be used in testing!", "source": {"file": "licensed_connector_mixin.py", "line": 386, "function": "skip_license_check"}, "levelname": "WARNING", "taskName": "Task-3"}

 Fetching California county unemployment data from FRED…

{"timestamp": "2025-11-29T17:19:11.546923Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060370000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060370000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:11.926176Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for

LAUCN060370000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060370000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:11.927941Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060590000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060590000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:12.296567Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060590000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060590000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:12.298863Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060650000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060650000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:12.802511Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060650000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060650000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:12.804012Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060710000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060710000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:13.311003Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060710000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060710000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:13.312196Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN061110000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN061110000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:13.383701Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for

LAUCN061110000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN061110000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:13.384706Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060290000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060290000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:13.517423Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060290000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060290000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:13.518785Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060250000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060250000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:13.616929Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060250000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060250000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:13.618062Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060730000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060730000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:14.441258Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060730000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060730000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:14.442111Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060790000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060790000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:14.605439Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for

LAUCN060790000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060790000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:14.606042Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060830000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060830000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:14.711635Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060830000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060830000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:14.712264Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060190000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060190000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:15.112949Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060190000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060190000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:15.113521Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060990000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060990000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:15.205384Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060990000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060990000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:15.205933Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060770000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060770000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:15.664323Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for

LAUCN060770000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060770000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:15.665016Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060470000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060470000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:15.828943Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060470000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060470000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:15.829567Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060390000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060390000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:15.950968Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN060390000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060390000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:15.951667Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN061070000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN061070000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:16.268677Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for LAUCN061070000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN061070000000003A", "rows": 2}

{"timestamp": "2025-11-29T17:19:16.269788Z", "level": "INFO", "name": "FREDFullConnector", "message": "Fetching FRED series: LAUCN060310000000003A", "source": {"file": "fred_full.py", "line": 168, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060310000000003A", "start_date": "2022-01-01", "end_date": "2023-12-31", "units": "lin", "frequency": null}

{"timestamp": "2025-11-29T17:19:16.389870Z", "level": "INFO", "name": "FREDFullConnector", "message": "Retrieved 2 observations for

LAUCN060310000000003A", "source": {"file": "fred_full.py", "line": 211, "function": "get_series"}, "levelname": "INFO", "taskName": "Task-3", "series_id": "LAUCN060310000000003A", "rows": 2}

```
    Retrieved data for 17 California counties

 Environmental Justice Dataset (Real FRED + Simulated)
   • Counties: 17
   • Region: Southern California / Central Valley
   • Avg unemployment (real): 6.7%
   • Avg PM2.5 (simulated): 9.5 g/m³
   • Avg poverty rate: 20.1%
```

[2]:

| | tract_id | county_name | longitude | latitude | unemployment_rate |
|---|---|---|---|---|---|
| 0 | CA_037 | Los Angeles | -118.24 | 34.05 | 5.1 |
| 1 | CA_059 | Orange | -117.83 | 33.71 | 3.5 |
| 2 | CA_065 | Riverside | -116.97 | 33.95 | 4.8 |
| 3 | CA_071 | San Bernardino | -117.29 | 34.84 | 4.6 |
| 4 | CA_111 | Ventura | -119.22 | 34.35 | 4.2 |

| | vulnerability_base | pm25 | diesel_pm | ozone | toxic_releases | … |
|---|---|---|---|---|---|---|
| 0 | 0.113475 | 4.000000 | 0.318072 | 0.047620 | 73.100405 | … |
| 1 | 0.000000 | 11.932823 | 0.214581 | 0.052997 | 31.930635 | … |
| 2 | 0.092199 | 11.780589 | 0.696467 | 0.050913 | 102.839127 | … |
| 3 | 0.078014 | 4.320472 | 0.419030 | 0.062831 | 196.691118 | … |
| 4 | 0.049645 | 6.715862 | 0.497236 | 0.065821 | 0.000000 | … |

| | low_education | linguistic_isolation | housing_burden | minority_pct |
|---|---|---|---|---|
| 0 | 3.000000 | 15.516815 | 46.145693 | 36.467272 |
| 1 | 9.723763 | 6.154769 | 39.882026 | 38.243645 |
| 2 | 4.139847 | 2.338294 | 34.627926 | 14.624866 |
| 3 | 19.377724 | 11.457330 | 33.712164 | 68.956107 |
| 4 | 11.328748 | 6.001485 | 19.715481 | 59.637022 |

| | children_pct | elderly_pct | asthma_rate | heart_disease_rate |
|---|---|---|---|---|
| 0 | 22.723587 | 7.707030 | 10.168023 | 4.147120 |
| 1 | 11.938863 | 18.172271 | 8.141945 | 3.000000 |
| 2 | 25.648894 | 18.030644 | 9.524044 | 3.019853 |
| 3 | 27.112176 | 12.110465 | 8.329104 | 4.051665 |
| 4 | 22.382935 | 10.534504 | 7.374652 | 7.572102 |

| | low_birth_weight | population |
|---|---|---|
| 0 | 9.435852 | 81286 |
| 1 | 7.597841 | 138505 |
| 2 | 4.000000 | 221640 |
| 3 | 5.719911 | 338246 |
| 4 | 8.055611 | 29258 |

[5 rows x 27 columns]

## 0.3 3. Calculate EJ Burden Scores (Community Tier)

[3]:
```
# ============================================================================
# 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: 52.9
      Std: 19.1
      Range: 23.9 - 97.1

   Socioeconomic Vulnerability Score:
      Mean: 52.9
      Std: 21.9
      Range: 20.1 - 100.0

   Cumulative EJ Score:
      Mean: 30.6
      Std: 23.1
      Range: 6.7 - 97.1
```

[4]:
```
# ============================================================================
# Visualize Score Distributions
# ============================================================================

COLORS = ['#0072B2', '#E69F00', '#009E73', '#CC79A7', '#56B4E9', '#D55E00']

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=(
        'Environmental Burden Distribution',
        'Socioeconomic Vulnerability Distribution',
        'Cumulative Environmental Justice Score',
        'EJ Vulnerability Space'
    ),
    vertical_spacing=0.12,
    horizontal_spacing=0.08
)
```

```python
# 1. Environmental burden distribution
fig.add_trace(
    go.Histogram(x=ej_data['env_burden_score'], nbinsx=30,␣
 ↪marker_color=COLORS[0],
                 opacity=0.7, name='Env Burden'),
    row=1, col=1
)
env_median = ej_data['env_burden_score'].median()
fig.add_vline(x=env_median, line_dash='dash', line_color='red',
              annotation_text=f'Median: {env_median:.0f}', row=1, col=1)

# 2. Socioeconomic vulnerability distribution
fig.add_trace(
    go.Histogram(x=ej_data['socio_vulnerability_score'], nbinsx=30,␣
 ↪marker_color=COLORS[1],
                 opacity=0.7, name='Socio Vuln'),
    row=1, col=2
)
socio_median = ej_data['socio_vulnerability_score'].median()
fig.add_vline(x=socio_median, line_dash='dash', line_color='red',
              annotation_text=f'Median: {socio_median:.0f}', row=1, col=2)

# 3. Cumulative EJ score
fig.add_trace(
    go.Histogram(x=ej_data['ej_score'], nbinsx=30, marker_color=COLORS[5],
                 opacity=0.7, name='EJ Score'),
    row=2, col=1
)
ej_75pctl = ej_data['ej_score'].quantile(0.75)
fig.add_vline(x=ej_75pctl, line_dash='dash', line_color='orange',
              annotation_text='75th pctl (DAC)', row=2, col=1)

# 4. Two-dimensional vulnerability space
fig.add_trace(
    go.Scatter(
        x=ej_data['env_burden_score'],
        y=ej_data['socio_vulnerability_score'],
        mode='markers',
        marker=dict(
            size=8,
            color=ej_data['ej_score'],
            colorscale=[[0, '#2E8B57'], [0.33, '#FFD700'], [0.66, '#FF4500'],␣
 ↪[1, '#8B0000']],
            colorbar=dict(title='EJ Score', x=1.02),
            opacity=0.7,
            line=dict(width=0.5, color='black')
```

```
        ),
        name='Tracts',
        showlegend=False
    ),
    row=2, col=2
)

fig.add_hline(y=50, line_dash='dash', line_color='gray', opacity=0.5, row=2,␣
 ↪col=2)
fig.add_vline(x=50, line_dash='dash', line_color='gray', opacity=0.5, row=2,␣
 ↪col=2)

# Add quadrant annotations
fig.add_annotation(x=75, y=75, text='High Burden<br>High Vuln', showarrow=False,
                   font=dict(color='darkred', size=10), row=2, col=2)
fig.add_annotation(x=25, y=75, text='Low Burden<br>High Vuln', showarrow=False,
                   font=dict(color='gray', size=10), row=2, col=2)
fig.add_annotation(x=75, y=25, text='High Burden<br>Low Vuln', showarrow=False,
                   font=dict(color='gray', size=10), row=2, col=2)
fig.add_annotation(x=25, y=25, text='Low Burden<br>Low Vuln', showarrow=False,
                   font=dict(color='green', size=10), row=2, col=2)

# Update axes labels
fig.update_xaxes(title_text='Environmental Burden Score', row=1, col=1)
fig.update_yaxes(title_text='Frequency', row=1, col=1)
fig.update_xaxes(title_text='Socioeconomic Vulnerability Score', row=1, col=2)
fig.update_yaxes(title_text='Frequency', row=1, col=2)
fig.update_xaxes(title_text='Cumulative EJ Score', row=2, col=1)
fig.update_yaxes(title_text='Frequency', row=2, col=1)
fig.update_xaxes(title_text='Environmental Burden Score', row=2, col=2)
fig.update_yaxes(title_text='Socioeconomic Vulnerability', row=2, col=2)

# Update layout
fig.update_layout(
    title_text='Environmental Justice Score Components',
    title_font_size=16,
    height=700,
    width=1000,
    showlegend=False,
    template='plotly_white'
)

fig.show()
```

## 0.4 4. Disparity Analysis (Community Tier)

[5]:
```python
# ============================================================================
# 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): 40.5
Disadvantaged communities: 5 tracts (29%)

----------------------------------------------------------------------
Indicator                      DAC Mean   Non-DAC Mean      Ratio
----------------------------------------------------------------------
pm25                               11.6            8.7       1.3x
diesel_pm                           1.3            0.7       1.8x
toxic_releases                      368            347       1.1x
poverty_rate                       30.3           15.8       1.9x
minority_pct                       52.0           39.7       1.3x
asthma_rate                        12.9            8.9       1.4x
low_birth_weight                    9.0            6.8       1.3x
----------------------------------------------------------------------

Key Finding: DAC communities face 1.5x average burden across indicators
```

[6]:

```python
# ============================================================================
# Visualize Disparities
# ============================================================================

COLORS = ['#0072B2', '#E69F00', '#009E73', '#CC79A7', '#56B4E9', '#D55E00']

fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=('Environmental & Health Disparities', 'Demographic Composition'),
    horizontal_spacing=0.12
)

# 1. Disparity ratios
sorted_disparities = dict(sorted(disparities.items(), key=lambda x: x[1], reverse=True))
ratios = list(sorted_disparities.values())
names = list(sorted_disparities.keys())

fig.add_trace(
    go.Bar(
        y=names,
        x=ratios,
        orientation='h',
        marker_color=COLORS[1],
        opacity=0.7,
        text=[f'{r:.1f}x' for r in ratios],
        textposition='outside',
        name='Disparity Ratio'
```

```python
    ),
    row=1, col=1
)
fig.add_vline(x=1.0, line_dash='dash', line_color='black', line_width=1, row=1,␣
 ↪col=1)
fig.add_vline(x=2.0, line_dash='dash', line_color='red', opacity=0.5,
              annotation_text='2x disparity', row=1, col=1)

# 2. Demographic breakdown of DAC vs non-DAC
demo_vars = ['minority_pct', 'low_income_pct', 'low_education',␣
 ↪'linguistic_isolation']
demo_labels = ['Minority', 'Low Income', 'Low Education', 'Linguistic␣
 ↪Isolation']

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

fig.add_trace(
    go.Bar(
        x=demo_labels,
        y=dac_vals,
        name='DAC Tracts',
        marker_color=COLORS[5],
        opacity=0.7
    ),
    row=1, col=2
)

fig.add_trace(
    go.Bar(
        x=demo_labels,
        y=non_dac_vals,
        name='Non-DAC Tracts',
        marker_color=COLORS[0],
        opacity=0.7
    ),
    row=1, col=2
)

# Update axes labels
fig.update_xaxes(title_text='DAC / Non-DAC Ratio', row=1, col=1)
fig.update_yaxes(title_text='', row=1, col=1)
fig.update_xaxes(title_text='', tickangle=15, row=1, col=2)
fig.update_yaxes(title_text='Percentage', row=1, col=2)

# Update layout
fig.update_layout(
```

```
    title_text='Disparity Analysis: DAC vs Non-DAC Communities',
    title_font_size=14,
    height=450,
    width=1000,
    barmode='group',
    template='plotly_white',
    legend=dict(orientation='h', yanchor='bottom', y=1.02, xanchor='right', x=1)
)

fig.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.

[7]:
```
# ============================================================================
# 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
```

```python
        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: 2 (11.8%)
  Low income threshold: 65% below 200% FPL

  Category-Specific Burdens:
    Climate Change: 4 tracts (23.5%)
    Energy: 2 tracts (11.8%)
    Health: 2 tracts (11.8%)
    Housing: 2 tracts (11.8%)
    Legacy Pollution: 3 tracts (17.6%)
    Transportation: 2 tracts (11.8%)
    Water & Wastewater: 2 tracts (11.8%)
    Workforce Development: 2 tracts (11.8%)

  Multi-Burden Analysis:
    0 categories: 8 tracts
    1 categories: 7 tracts
    2 categories: 0 tracts
    3 categories: 0 tracts
```

[8]:
```
# ============================================================================
# Visualize CEJST Screening
# ============================================================================

COLORS = ['#0072B2', '#E69F00', '#009E73', '#CC79A7', '#56B4E9', '#D55E00']

fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=('CEJST Category Burdens', 'CEJST DAC Designation Map'),
    horizontal_spacing=0.1
)

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

fig.add_trace(
    go.Bar(
        y=categories,
```

```
        x=prevalence,
        orientation='h',
        marker_color=COLORS[0],
        opacity=0.7,
        name='Prevalence'
    ),
    row=1, col=1
)
fig.add_vline(x=10, line_dash='dash', line_color='red', opacity=0.5,
              annotation_text='10% threshold', row=1, col=1)

# 2. Spatial distribution of DAC status
dac_mask = cejst_result.is_dac

# DAC tracts
fig.add_trace(
    go.Scatter(
        x=ej_data.loc[dac_mask, 'longitude'],
        y=ej_data.loc[dac_mask, 'latitude'],
        mode='markers',
        marker=dict(size=8, color=COLORS[5], opacity=0.7, line=dict(width=0.5,␣
↪color='black')),
        name=f'DAC ({cejst_result.dac_count})'
    ),
    row=1, col=2
)

# Non-DAC tracts
fig.add_trace(
    go.Scatter(
        x=ej_data.loc[~dac_mask, 'longitude'],
        y=ej_data.loc[~dac_mask, 'latitude'],
        mode='markers',
        marker=dict(size=8, color=COLORS[0], opacity=0.7, line=dict(width=0.5,␣
↪color='black')),
        name=f'Non-DAC ({len(ej_data) - cejst_result.dac_count})'
    ),
    row=1, col=2
)

# Update axes labels
fig.update_xaxes(title_text='% of Tracts Burdened', row=1, col=1)
fig.update_xaxes(title_text='Longitude', row=1, col=2)
fig.update_yaxes(title_text='Latitude', row=1, col=2)

fig.update_layout(
    title_text='Pro Tier: CEJST-Style Categorical Screening',
```

```
    title_font_size=14,
    height=450,
    width=1000,
    template='plotly_white',
    legend=dict(orientation='h', yanchor='bottom', y=1.02, xanchor='right', x=1)
)

fig.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.

[9]:
```
# =============================================================================
# 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
""")

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

[10]:
```
# ============================================================================
# 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: 17
  Environmental indicators: 8
  Socioeconomic indicators: 6
  Health outcomes: 3

 KEY FINDINGS:

  1. DISADVANTAGED COMMUNITY IDENTIFICATION
     CalEnviroScreen method (75th pctl): 5 tracts (29%)
     CEJST categorical method: 2 tracts (12%)

  2. ENVIRONMENTAL BURDEN DISPARITIES
     PM2.5 exposure: DAC 11.6 vs Non-DAC 8.7 μg/m³
     Toxic releases: DAC 1.1x higher
```

```
  3. HEALTH OUTCOME DISPARITIES
     Asthma rate: DAC 12.9% vs Non-DAC 8.9%
     Low birth weight: DAC 9.0% vs Non-DAC 6.8%

  4. DEMOGRAPHIC CONCENTRATION
     Minority population in DAC: 52%
     Low income in DAC: 49%

 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*