In [1]:
```
!pip install -q folium mapclassify

import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd

# For testing purposes
import json
import numpy as np
from geopandas.plotting import _plot_polygon_collection
from matplotlib.collections import PatchCollection

with open("expected_idx.json") as f:
    rural_idx, rural_la_idx, urban_ha_idx, lalowi_idx = json.load(f)

class MockAxes:
    def add_collection(self, *args, **kwargs):
        pass

    def autoscale_view(self, *args, **kwargs):
        pass

def assert_patches_allclose(actual_patches, ax=MockAxes(), num_colors=None, **kwargs):
    if isinstance(kwargs.get("geoms"), str):
        kwargs["geoms"] = gpd.read_file(kwargs["geoms"]).geometry
    expected_patches = _plot_polygon_collection(ax=ax, **kwargs)
    for expected, actual in zip(expected_patches.get_paths(), actual_patches.get_paths()):
        try:
            np.testing.assert_allclose(expected.vertices, actual.vertices)
        except AssertionError as e:
            e.args = "wrong selection",
            raise e
    if "color" in kwargs:
        try:
            np.testing.assert_allclose(expected_patches.get_fc(), actual_patches.get_fc())
        except AssertionError as e:
            e.args = "wrong color",
            raise e
    elif isinstance(num_colors, int):
        num_unique = len(np.unique(actual_patches.get_fc(), axis=0))
        assert num_unique == num_colors, f"expected {num_colors} colors, got {num_unique} colors"
```

Collaboration and Conduct

Students are expected to follow Washington state law on the Student Conduct Code for the University of Washington. In this course, students must:

- Indicate on your submission any assistance received, including materials distributed in this course.
- Not receive, generate, or otherwise acquire any substantial portion or walkthrough to an assessment.
- Not aid, assist, attempt, or tolerate prohibited academic conduct in others.

Update the following code cell to include your name and list your sources. If you used any kind of computer technology to help prepare your assessment submission, include the queries and/or prompts. Submitted work that is not consistent with sources may be subject to the student conduct process.

In [2]:
```
your_name = "Enrico Pratama"
sources = [
    "Lecture Notes",
    "Lecture Recordings",
    "Office Hours and TAs",
    "Quiz Sections",
    "GeoPandas"
]

assert your_name != "", "your_name cannot be empty"
assert ... not in sources, "sources should not include the placeholder ellipsis"
assert len(sources) >= 2, "must include at least 2 sources, inclusive of lectures and sections"
```

Task: Load in data

Write a function `load_data` that takes path for census dataset and the path for the food access dataset and returns the `GeoDataFrame` resulting from merging the two datasets on the census tract identifiers `CTIDFP00` / `CensusTract`. Assume the census tract identifier columns exist: use only these two column names. Not all census tracts have food access data.

In [3]:
```
def load_data(shp_path, csv_path):
    """
    Loads and merges census and food access data.
    Takes path for census dataset and the path for the food access dataset and returns
    the GeoDataFrame resulting from merging the two datasets on the census tract identifiers
    CTIDFP00 / CensusTract. Assumes the census tract identifier columns exist:
    use only these two column names. Not all census tracts have food access data.

    Parameters:
    shp_path (str): Path to the shapefile containing census data.
    csv_path (str): Path to the CSV file containing food access data.
    Returns:
    gpd.GeoDataFrame: Merged GeoDataFrame containing both census and food access data.
    """


    census_data = gpd.read_file(shp_path)
    food_access_data = pd.read_csv(csv_path)
    # Left since Assume census tract identifier col exist
    merged_data = census_data.merge(food_access_data, left_on="CTIDFP00", right_on='CensusTract', how='left')
    return merged_data


state_data = load_data("tl_2010_53_tract00.shp", "food_access.csv")
display(state_data)
assert type(state_data) == gpd.GeoDataFrame
assert list(state_data.columns) == [
    "STATEFP00", "COUNTYFP00", "TRACTCE00", "CTIDFP00", "NAME00", "NAMELSAD00", "MTFCC00",
    "FUNCSTAT00", "ALAND00", "AWATER00", "INTPTLAT00", "INTPTLON00", "geometry", "CensusTract",
    "State", "County", "Urban", "Rural", "LATracts_half", "LATracts10", "GroupQuartersFlag",
    "OHU2010", "NUMGQTRS", "PCTGQTRS", "LowIncomeTracts", "POP2010", "lapophalf", "lalowihalf",
    "lapop10", "lalowi10",
]
assert len(state_data) == 1318
```

	STATEFP00	COUNTYFP00	TRACTCE00	CTIDFP00	NAME00	NAMELSAD00	MTFCC00	FUNCSTAT00	ALAND00	AWATER00
0	53	077	001400	53077001400	14	Census Tract 14	G5020	S	5539748	0
1	53	077	001600	53077001600	16	Census Tract 16	G5020	S	97657363	1509774
2	53	077	000700	53077000700	7	Census Tract 7	G5020	S	2930010	0
3	53	077	002400	53077002400	24	Census Tract 24	G5020	S	232557960	69748
4	53	077	002200	53077002200	22	Census Tract 22	G5020	S	207645882	0
...	...	...	...	...	...	...	...	...	...	...
1313	53	063	010202	53063010202	102.02	Census Tract 102.02	G5020	S	184070644	0
1314	53	063	010301	53063010301	103.01	Census Tract 103.01	G5020	S	21667422	0
1315	53	063	010504	53063010504	105.04	Census Tract 105.04	G5020	S	9371197	0
1316	53	063	010303	53063010303	103.03	Census Tract 103.03	G5020	S	107033392	0
1317	53	063	001600	53063001600	16	Census Tract 16	G5020	S	2104249	0

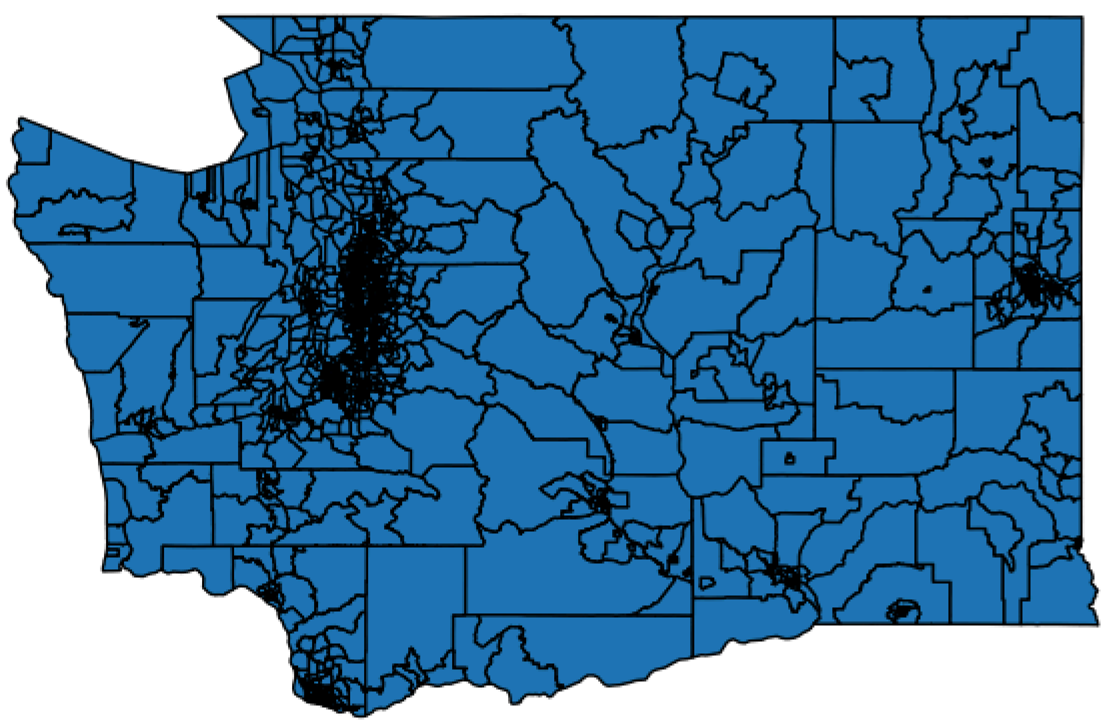
1318 rows × 30 columns

Task: Plot census tracts

Write a function `plot_census_map` that takes the merged data and returns the `Axes` that contains shapes of all the census tracts in Washington. Title the plot "Washington Census Tracts" and turn off axis labels.

In [4]:
```
def plot_census_map(state_data):
    """
    Plots the census tracts of Washington state.
    Takes the merged data and returns the Axes that contains shapes
    of all the census tracts in Washington.

    Parameters:
    state_data (gpd.GeoDataFrame): GeoDataFrame containing merged census and food access data.

    Returns:
    matplotlib.axes._subplots.AxesSubplot: The plot axis object.
    """

    fig, ax = plt.subplots(1, figsize=(13, 5))
    state_data.plot(ax=ax, edgecolor='black')
    ax.set_title("Washington Census Tracts")
    ax.axis('off')
    return ax

ax = plot_census_map(state_data)
layers = ax.findobj(PatchCollection)
assert_patches_allclose(layers[0], geoms=state_data.geometry)
assert len(layers) == 1, "unexpected extra plot layers"
assert ax.get_title() == "Washington Census Tracts", "title does not match expected"
assert not ax.axison, "borders and labels must be hidden"
```

Washington Census Tracts



When given no arguments, the `dissolve` method considers the entire `GeoDataFrame` as a single group. This will be useful for plotting backgrounds later.

```
In [5]: entire_state = state_data[["geometry"]].dissolve()
        display(entire_state)
        ax = entire_state.plot(color="#EEE")
        ax.set_axis_off()
```

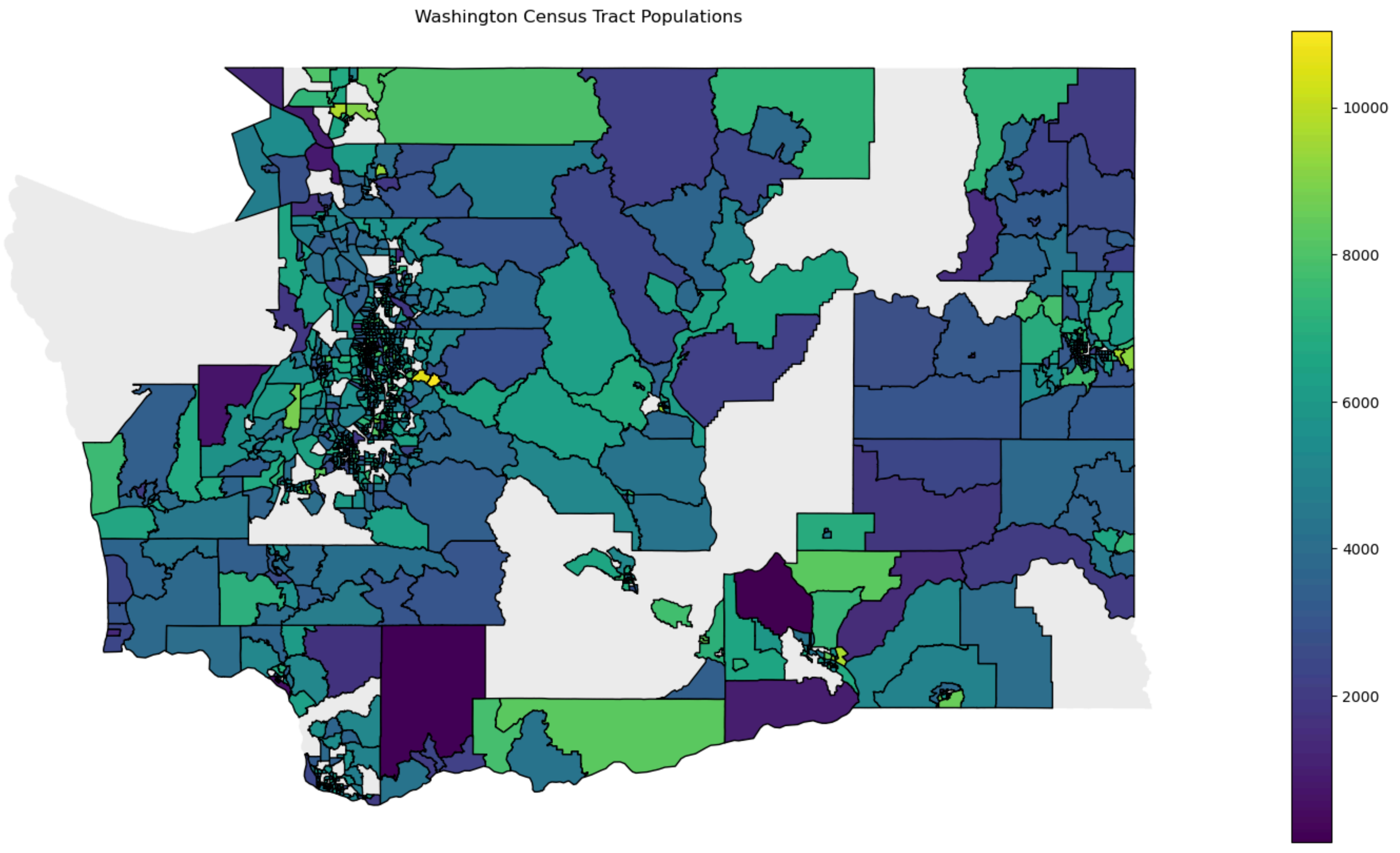
```
                                           geometry
0   POLYGON ((-122.88260 46.05175, -122.88261 46.0...
```

Task: Plot census tract populations

Write a function `plot_census_population_map` that takes the merged data and return the `Axes` that plots all the census tracts in Washington where each census tract is colored according to the `POP2010` column. There will be some missing census tracts. Underneath, plot the entire state of Washington in the background color `#EEE`. Title the plot "Washington Census Tract Populations", turn off axis labels, include a legend, and increase the figure size so that the map is the same height as the legend.

```
In [6]: def plot_census_population_map(state_data):
            """
            Plots the population distribution of census tracts in Washington state.
            Takes the merged data and return the Axes that plots all the census tracts
            in Washington where each census tract is colored according to the POP2010 column.

            Parameters:
            state_data (gpd.GeoDataFrame): GeoDataFrame containing merged census and food access data.

            Returns:
            matplotlib.axes._subplots.AxesSubplot: The plot axis object.
            """

            fig, ax = plt.subplots(figsize=(20, 10))
            entire_state.plot(ax=ax, color="#EEE")

            population_plot = state_data.plot(column='POP2010', ax=ax, legend=True, edgecolor='black')

            ax.set_title("Washington Census Tract Populations")
            ax.set_axis_off()
            return ax



        ax = plot_census_population_map(state_data)
        layers = ax.findobj(PatchCollection)
```

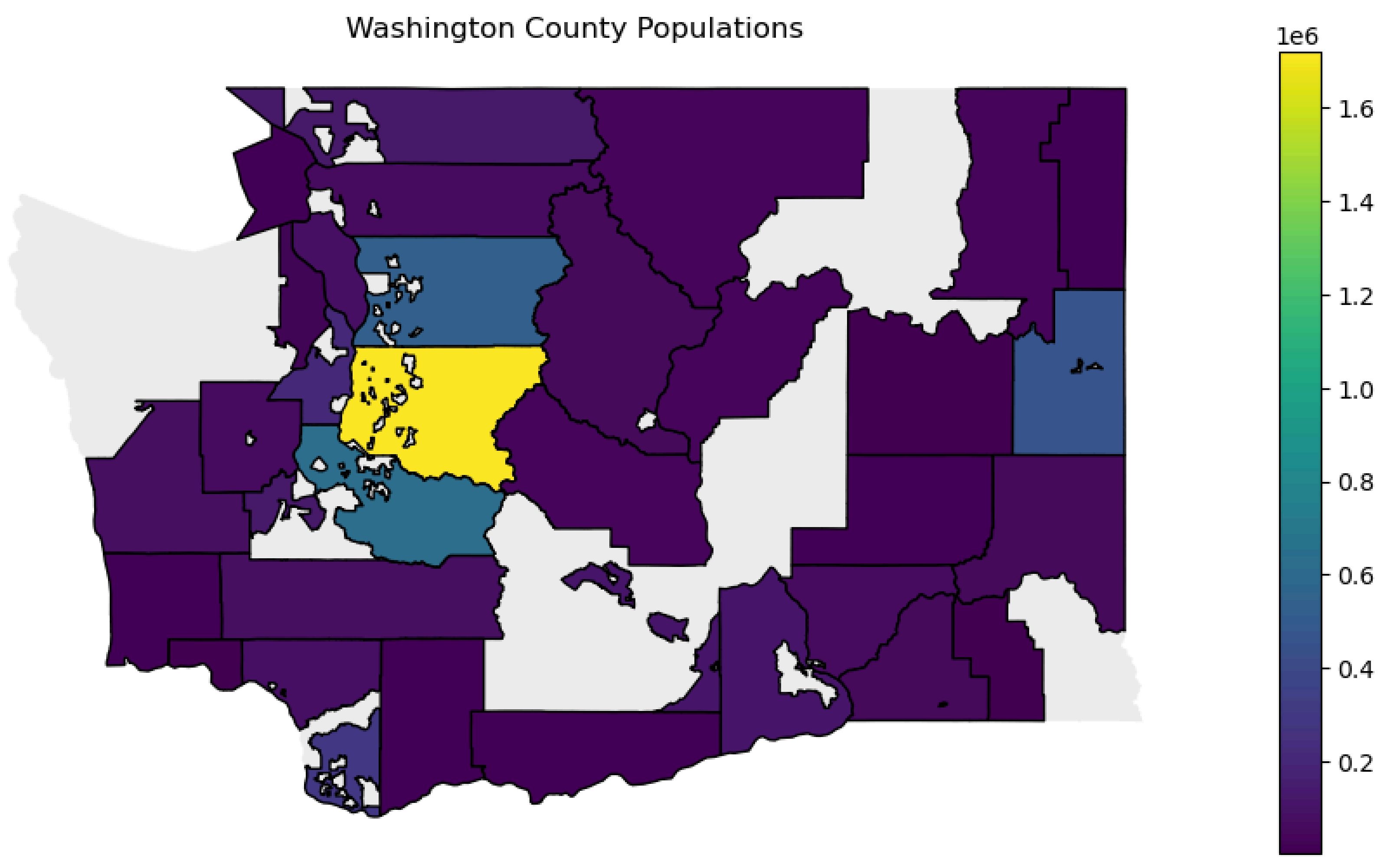
```
assert_patches_allclose(layers[0], geoms=entire_state.geometry, color="#EEE")
assert_patches_allclose(layers[1], geoms=state_data.dropna().geometry, num_colors=183)
assert len(layers) == 2, "unexpected extra plot layers"
assert ax.get_title() == "Washington Census Tract Populations", "title does not match expected"
assert not ax.axison, "borders and labels must be hidden"
cbar = ax.get_figure().get_axes()[-1]
assert cbar.get_label() == "<colorbar>", "missing legend"
assert ax.bbox.height == cbar.bbox.height, "map can be enlarged"
```



Task: Plot county populations

Write a function `plot_county_populations_map` that takes the merged data and returns the `Axes` that plots all the counties in Washington where each county is colored according to the `POP2010` column. This will require combining all the census tract data and geometries for each county, though there will be missing data for some counties. Underneath, plot the entire state of Washington in the background color `#EEE`. Title the plot "Washington County Populations", turn off axis labels, include a legend, and increase the figure size so that the map is the same height as the legend.

```
In [7]: def plot_county_population_map(state_data):
            """
            Plots the population distribution of counties in Washington state.
            Takes the merged data and returns the Axes that plots all the counties
            in Washington where each county is colored according to the POP2010 column

            Parameters:
            state_data (gpd.GeoDataFrame): GeoDataFrame containing merged census and food access data.

            Returns:
            matplotlib.axes._subplots.AxesSubplot: The plot axis object.
            """

            county_data = state_data.dissolve(by='County', aggfunc='sum')

            fig, ax = plt.subplots(figsize=(12, 6))

            entire_state = state_data[["geometry"]].dissolve()
            entire_state.plot(ax=ax, color="#EEE")
            ax.set_axis_off()

            county_data.plot(column='POP2010', ax=ax, cmap='viridis', legend=True, edgecolor='black')

            ax.set_title("Washington County Populations")

            return ax



        ax = plot_county_population_map(state_data)
        layers = ax.findobj(PatchCollection)
        assert_patches_allclose(layers[0], geoms=entire_state.geometry, color="#EEE")
        assert_patches_allclose(layers[1], geoms="counties.geojson", num_colors=20)
        assert len(layers) == 2, "unexpected extra plot layers"
        assert ax.get_title() == "Washington County Populations", "title does not match expected"
        assert not ax.axison, "borders and labels must be hidden"
        cbar = ax.get_figure().get_axes()[-1]
        assert cbar.get_label() == "<colorbar>", "missing legend"
        assert ax.bbox.height == cbar.bbox.height, "map can be enlarged"
```



Task: Plot food access by county

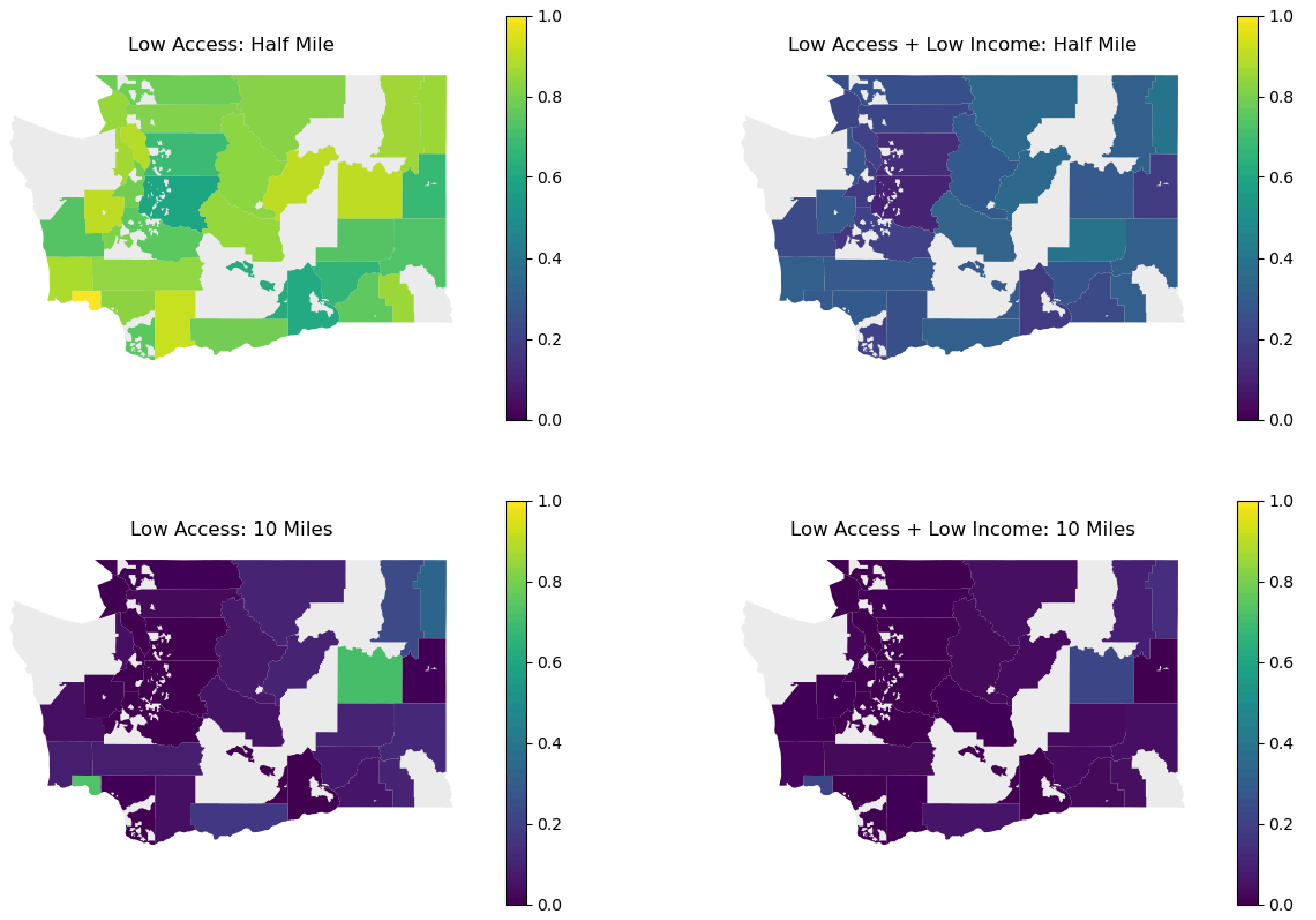
Write a function `plot_food_access_by_county_map` that takes the merged data and returns a 4-tuple of `Axes` that represent the subplots in a 2-by-2 figure consisting of 4 choropleth maps:

- Top left plot titled "Low Access: Half Mile" showing the proportion of people per county who have low access to food within a half mile `lapophalf` out of the total population `POP2010`.
- Top right plot titled "Low Access + Low Income: Half Mile" showing the proportion of people per county considered low income who also have low access to food within a half mile `lalowihalf` out of the total population `POP2010`.
- Bottom left plot titled "Low Access: 10 Miles" showing the proportion of people per county who have low access to food within 10 miles `lapop10` out of the total population `POP2010`.
- Bottom right plot titled "Low Access + Low Income: 10 Miles" showing the proportion of people per county considered low income who also have low access to food within 10 miles `lalowi10` out of the total population `POP2010`.

When calling `plot`, specify the keyword arguments `vmin=0` and `vmax=1` so that the subplots all share the same scale. Underneath, plot the entire state of Washington in the background color `#EEE`. We recommend preparing subplots with `figsize=(15, 10)`. Turn off axis labels on each subplot.

In [8]:
```
def plot_food_access_by_county_map(state_data):
    """
    Plots the food access metrics by county in Washington state.
    Top left plot titled "Low Access: Half Mile" showing the proportion of people per county who have
    low access to food within a half mile lapophalf out of the total population POP2010.

    Top right plot titled "Low Access + Low Income: Half Mile" showing the proportion of people
    per county considered low income who also have low access to food within a half mile lalowihalf
    out of the total population POP2010.

    Bottom left plot titled "Low Access: 10 Miles" showing the proportion of people per county
    who have low access to food within 10 miles lapop10 out of the total population POP2010.

    Bottom right plot titled "Low Access + Low Income: 10 Miles" showing the proportion of
    people per county considered low income who also have low access to food within 10 miles
    lalowi10 out of the total population POP2010.

    Parameters:
    state_data (gpd.GeoDataFrame): GeoDataFrame containing merged census and food access data.

    Returns:
    list: List of matplotlib.axes._subplots.AxesSubplot objects.
    """
    fig, [[ax1, ax2], [ax3, ax4]] = plt.subplots(2, 2, figsize=(15, 10))
    axes = [ax1, ax2, ax3, ax4]
    metrics = ['lapophalf', 'lalowihalf', 'lapop10', 'lalowi10', 'POP2010']
    counties_data = state_data.dissolve(by='County', aggfunc='sum')
    plot_titles = [
        "Low Access: Half Mile",
        "Low Access + Low Income: Half Mile",
        "Low Access: 10 Miles",
        "Low Access + Low Income: 10 Miles"
    ]

    for metric, ax, plot_title in zip(metrics, axes, plot_titles):
        entire_state.plot(ax=ax, color="#EEE")
        column = counties_data[metric] / counties_data["POP2010"]
        counties_data.plot(ax = ax, column = column, vmin = 0, vmax = 1, legend = True)
        ax.set_title(plot_title)
        ax.set_axis_off()

    return axes
```

```
axs = plot_food_access_by_county_map(state_data)
expected_titles = ["Low Access: Half Mile", "Low Access + Low Income: Half Mile",
                   "Low Access: 10 Miles", "Low Access + Low Income: 10 Miles"]
for ax, expected_num_colors, expected_title in zip(axs, [31, 23, 19, 16], expected_titles):
    layers = ax.findobj(PatchCollection)
    assert_patches_allclose(layers[0], geoms=entire_state.geometry, color="#EEE")
    assert_patches_allclose(layers[1], geoms="counties.geojson", num_colors=expected_num_colors)
    assert len(layers) == 2, "unexpected extra plot layers"
    assert ax.get_title() == expected_title, f"title {ax.get_title()} does not match expected"
    assert not ax.axison, "borders and labels must be hidden"
```



Low Access: Half Mile
Low Access + Low Income: Half Mile
Low Access: 10 Miles
Low Access + Low Income: 10 Miles
1.0
0.8
0.6
0.4
0.2
0.0

Writeup: Food access by county

Setting aside the lack of a legend in `plot_food_access_by_county_map`, is it an effective visualization? Why or why not?

I feel like this it can be more effective by having more colors that are more different to ensure that people can differentiate between the colors better since the current colors that are used may feel clustered. Also, this map assumes that people are failiar with the Washington state map which leads to lack of understanding for people who do not.

Task: Plot low access census tracts

Write a function `plot_census_low_access_map` that takes the merged data and returns the `Axes` that plots all census tracts (not counties) considered low access using 5 `plot` layers for the following definitions for "low access" in urban and rural tracts. For this task, do not use the `LATracts_half` or `LATracts10` columns in the merged data; the procedure described below intentionally results in different values.

1. Plot the map of Washington in the background with color `#EEE`.
2. Plot all the `Urban` and `Rural` census tracts for which we have food access data in the color `#AAA`.
3. Plot all the `Urban` and `Rural` census tracts considered low access in the default (blue) color.

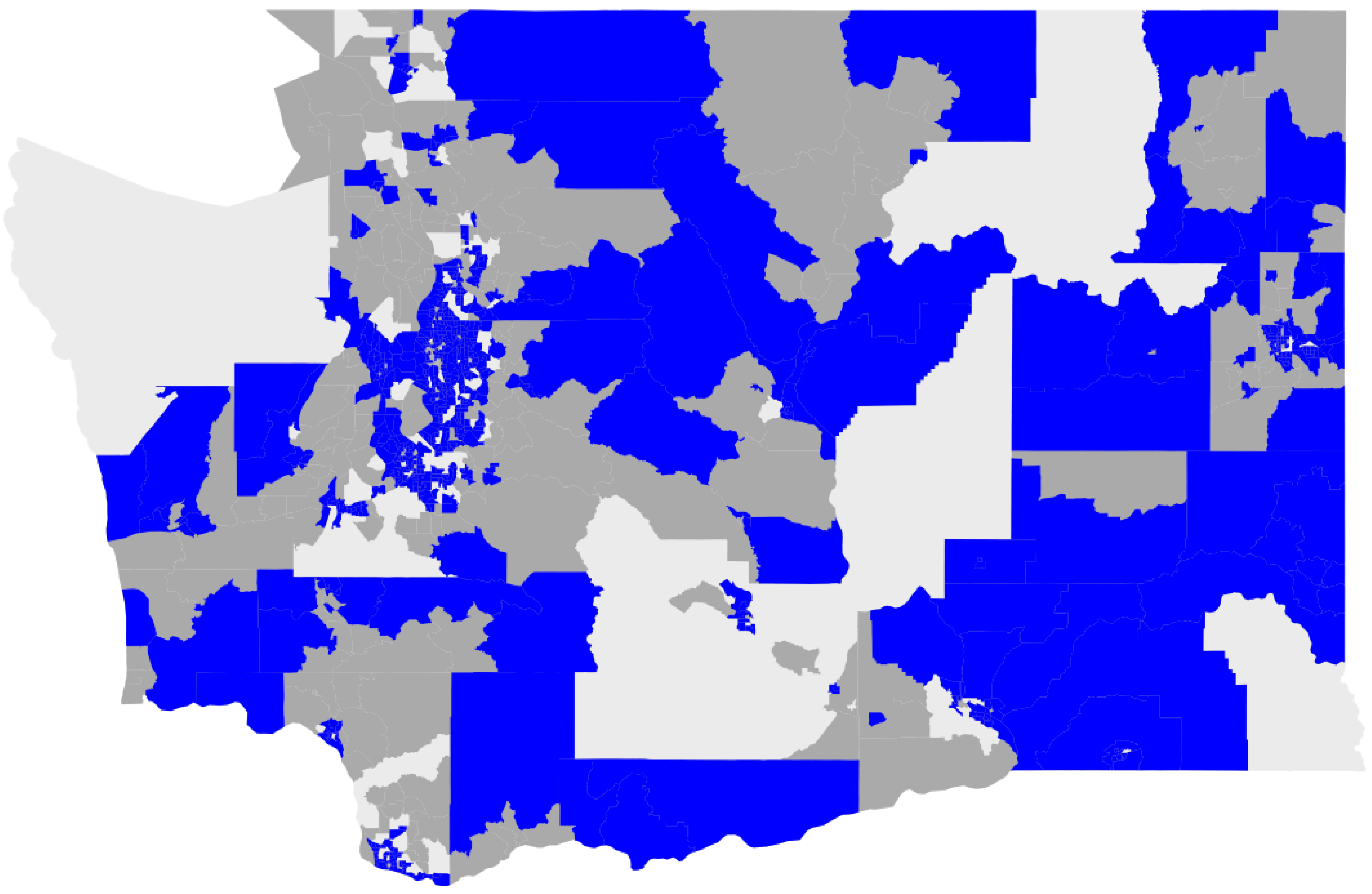
   Low access in an urban census tract is defined by a `lapophalf` value that is at least 500 people or at least 33% of the census tract population.

   Low access in a rural census tract is defined by a `lapop10` value that is at least 500 people or at least 33% of the census tract population.

Finally, title the plot "Low Access Census Tracts" and turn off axis labels.

In [9]:
```
def plot_census_low_access_map(state_data):
    """
    Plots the low access census tracts in Washington state where:
    Low access in an urban census tract is defined by a lapophalf value that is at least 500
    people or at least 33% of the census tract population.
    Low access in a rural census tract is defined by a lapop10 value that is at
    least 500 people or at least 33% of the census tract population.

    Parameters:
    state_data (gpd.GeoDataFrame): GeoDataFrame containing merged census and food access data.

    Returns:
    matplotlib.axes._subplots.AxesSubplot: The plot axis object.
    """

    fig, ax = plt.subplots(figsize=(15, 10)) # Set up the plot

    entire_state.plot(ax=ax, color="#EEE") # Plot the map of WA in the background color #EEE

    urban_data = state_data[(state_data['Urban'] == 1)]
    rural_data = state_data[(state_data['Rural'] == 1)]
    urban_plot = urban_data.plot(ax=ax, color='#AAA')
    rural_plot = rural_data.plot(ax=ax, color='#AAA')


    urban_low_access = state_data[
        (state_data['Urban'] == 1) &
        ((state_data['lapophalf'] >= 500) | (state_data['lapophalf'] / state_data['POP2010'] >= 0.33))
    ]

    rural_low_access = state_data[
        (state_data['Rural'] == 1) &
        ((state_data['lapop10'] >= 500) | (state_data['lapop10'] / state_data['POP2010'] >= 0.33))
    ]

    urban_low_access.plot(ax=ax, color='blue', legend=True)
    rural_low_access.plot(ax=ax, color='blue', legend=True)

    ax.set_title("Low Access Census Tracts")
    ax.set_axis_off()

    return ax


ax = plot_census_low_access_map(state_data)
layers = ax.findobj(PatchCollection)
assert_patches_allclose(layers[0], geoms=entire_state.geometry, color="#EEE")
urban_idx = state_data["Urban"].notna() & ~state_data.index.isin(rural_idx)
urban_la_idx = urban_idx & ~state_data.index.isin(urban_ha_idx)
error = None
for i, j, k, l in [
    [1, 2, 3, 4], # urban, rural, urban low access, rural low access
    [2, 1, 3, 4], # rural, urban, urban low access, rural low access
    [1, 2, 4, 3], # urban, rural, rural low access, urban low access
    [2, 1, 4, 3], # rural, urban, rural low access, urban low access
    [1, 3, 2, 4], # urban, urban low access, rural, rural low access
    [3, 1, 4, 2], # rural, rural low access, urban, urban low access
]:
    try:
        assert_patches_allclose(layers[i], geoms=state_data.loc[urban_idx, "geometry"], color="#AAA")
        assert_patches_allclose(layers[j], geoms=state_data.loc[rural_idx, "geometry"], color="#AAA")
        assert_patches_allclose(layers[k], geoms=state_data.loc[urban_la_idx, "geometry"])
        assert_patches_allclose(layers[l], geoms=state_data.loc[rural_la_idx, "geometry"])
        break
    except AssertionError as e:
        if error is None: # Store only the first error encountered during testing
            error = e
else: # Only raise an error if none of the possible ways to layer the plot worked
    raise error
assert len(layers) == 5, "unexpected extra plot layers"
assert ax.get_title() == "Low Access Census Tracts", "title does not match expected"
assert not ax.axison, "borders and labels must be hidden"
```

Low Access Census Tracts



Writeup: Data-driven decision-making

What is one way that government or food access organizations could use `plot_food_access_by_county` or `plot_low_access_tracts` to shape their decisions or policies? Then, explain one limitation or concern with using the plots in the way you suggested.

The government agencies or food access organizations can use the visualizations like plot_food_access_by_county or plot_low_access_tracts to strategically target target interventions aimed at improving food access at certain regions, as determined by the plots above. This is because by identifying areas with significant populations of low-income individuals who have limited access to food, policymakers can better allocate resources such as mobile food markets, subsidies for local grocery stores, or improvements in public transportation systems to ensure these areas receive adequate support.

However, the concern is that this plot might be inaccurate or represent an oversimplification of the issues tha affects food access. The map might summarise or combine lack of food access purely based on distance to the nearest supermarket, but it might not account for other factors such as the affordability of food, the quality of available food, or cultural appropriateness of the food items stocked by nearby stores.

Task: Interactive map

Although the initial report to Congress was completed in June 2009, the Economic Research Service has since then maintained an interactive map for their **Food Access Research Atlas**. Open this interactive map, turn off the default layer "LI and LA and 1 and 10 miles", and turn on the layer "LI and LA at 1/2 and 10 miles". This layer displays:

> Low-income census tracts where a significant number or share of residents is more than 1/2 mile (urban) or 10 miles (rural) from the nearest supermarket.

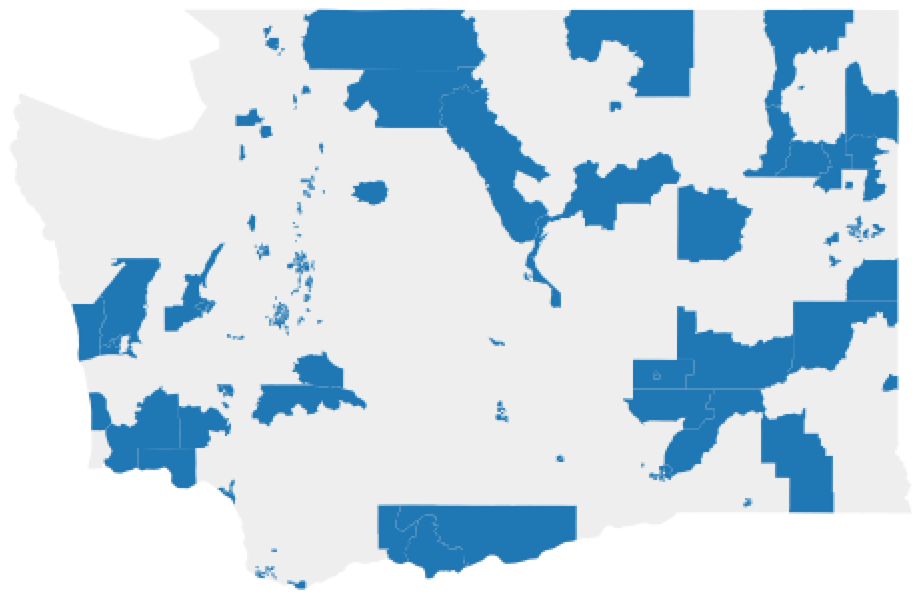
Write a function `interactive_map` that returns a Folium `Map` of low income and low access census tracts in Washington. Include only `LowIncomeTracts` that are either low access at half a mile `LATracts_half` or low access at 10 miles `LATracts10` depending on whether the census tract is `Urban` or `Rural`, respectively. This dataset does not match the Food Access Research Atlas, so some differences are to be expected. It is also OK if your interactive map does not appear in the PDF printout as PDF files cannot embed interactive maps.

In [12]:
```
def interactive_map(state_data):
    """
    Creates an interactive map of low-income tracts in Washington state.
    Returns a Folium Map of low income and low access census tracts in Washington.
    Includes only LowIncomeTracts that are either low access at half a mile LATracts_half or low
    access at 10 miles LATracts10 depending on whether the census tract is Urban or Rural, respectively.

    Parameters:
    state_data (gpd.GeoDataFrame): GeoDataFrame containing merged census and food access data.

    Returns:
    folium.Map: Interactive map highlighting low-income tracts with food access issues.
    """
    return state_data[
        (state_data['LowIncomeTracts'] == 1) &
        ((state_data['Urban'] == 1) & (state_data['LATracts_half'] == 1) |
         (state_data['Rural'] == 1) & (state_data['LATracts10'] == 1))
    ].explore()


map = interactive_map(state_data)
display(map)
last_child = next(reversed(map._children.values()))
from folium.features import GeoJson
assert type(last_child) == GeoJson, "last child should be GeoJson; do not specify column"
geojson = last_child.data
assert set(int(d["id"]) for d in geojson["features"]) == set(lalowi_idx), "wrong selection"
```

Make this Notebook Trusted to load map: File -> Trust Notebook

The following cell plots a preview of your interactive map for the PDF printout.

In [11]:
```
ax = entire_state.plot(color="#EEE")
gpd.GeoDataFrame.from_features(geojson, crs="EPSG:4326").plot(ax=ax)
ax.set_axis_off()
```



Writeup: Build a new supermarket

Using the interactive map above, locate the low-income low-access census tract closest to your favorite place in Washington. Then, identify a location (a specific street intersection, such as "University Way NE & NE 45th St") to add a new supermarket that would serve the people living in that census tract. Finally, explain the considerations that factored into your choice of location.

This dataset is outdated, so assume there are no supermarkets in any low-income low-access census tract even if supermarkets are present today.

The location that I identified as closest to my favorite place in Washington that is in need is 3800 Montlake Blvd NE, Seattle, WA 98195 which is actually near the Husky Stadium area as I believe that there is indeed no supermarket as far as I know near that area. Hence, I believe that a supermarket situated in the center of that low-access census tract near UW Husky Stadium Area will be beneficial.