Part 0: Initial Setup

```
import requests
import folium
import requests_cache
import pandas as pd
from retry_requests import retry
import numpy as np
from folium import plugins
import json
import shapely.geometry
import shapely.ops
import requests
import openmeteo_requests
import re  # Import the regular expression module
import jinja2
from folium import Html
from branca.element import Element, Figure
import sqlite3  # Import sqlite3

# Function to Fetch GeoJSON from CNRA API
def
get_california_geojson_from_cnra_api(url="https://gis.data.cnra.ca.gov
/api/download/v1/items/c3c10388e3b24cec8a954ba10458039d/geojson?
layers=0"):
    """
    Fetches the California GeoJSON data from the CNRA API.
    Args:
        url: The URL of the API endpoint that returns the GeoJSON
data.
    Returns:
        The GeoJSON data as a Python dictionary, or None if an error
occurred.
    """
    try:
        response = requests.get(url)
        response.raise_for_status()  # Raise an exception for bad
status codes
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error fetching GeoJSON data: {e}")
        return None

# Define the boundaries for Los Angeles County (approximate)
min_lat, max_lat = 33.28, 34.86  # Latitude range
min_lon, max_lon = -119.1, -117.3  # Longitude range

# Function to check if a point is within the Los Angeles County
boundaries
```

```
def is_point_in_la_county(latitude, longitude):
    return min_lat <= latitude <= max_lat and min_lon <= longitude <=
max_lon

def create_la_county_base_map():
    """Creates a Folium map centered on Los Angeles with LA county
boundaries."""
    california_geojson = get_california_geojson_from_cnra_api()
    if not california_geojson:
        print("Failed to fetch California GeoJSON data for base map
creation.")
        return None, None, None  # Ensure three values are returned

    # □ Create a base map without setting tiles (to manually add
later)
    m = folium.Map(location=[34.0522, -118.2437], zoom_start=8,
tiles=None) #Zoom out from 9 to 8

    # □ Add CartoDB Positron as a FeatureGroup so it behaves like LA
County Boundary
    base_map_layer = folium.FeatureGroup(name="CartoDB Positron Base
Map", control=True, overlay=True)
    folium.TileLayer("cartodbpositron").add_to(base_map_layer)
    base_map_layer.add_to(m)

    # □ Create LA County Boundary FeatureGroup
    la_county_layer = folium.FeatureGroup(name="Los Angeles County
Boundary", control=True, overlay=True)

    # Extract only LA County features from the full California GeoJSON
    la_county_features = []
    if 'features' in california_geojson:
        for feature in california_geojson['features']:
            if feature['geometry']['type'] == 'Polygon':
                coordinates = feature['geometry']['coordinates'][0]
                for lon, lat in coordinates:
                    if is_point_in_la_county(lat, lon):
                        la_county_features.append(feature)
                        break
            elif feature['geometry']['type'] == 'MultiPolygon':
                for polygon_coords in feature['geometry']
['coordinates']:
                    for coordinates in polygon_coords:
                        for lon, lat in coordinates:
                            if is_point_in_la_county(lat, lon):
                                la_county_features.append(feature)
                                break
                        else:
                            continue  # Continue to the next
polygon_coords if inner loop did not break
```

```
                        break  # Break outer loop if inner loop broke
                    else:
                        continue  # Continue to the next feature if 
middle loop did not break
                    break  # Break outermost loop if middle loop broke

    la_county_geojson_data = {
        "type": "FeatureCollection",
        "features": la_county_features
    }

    folium.GeoJson(la_county_geojson_data, name="Los Angeles County 
Data").add_to(la_county_layer)
    la_county_layer.add_to(m)

    return m, la_county_layer, base_map_layer  # Return both layers
```

Part 1: Fire Weather Index Calculation Functions

```
# Open-Meteo API for Weather Data
def get_weather_data(latitude, longitude, start_date, end_date):
    """
    Fetches weather data, including hourly Tmax and RHmin, from Open-
    Meteo API.
    """
    cache_session = requests_cache.CachedSession('.cache', 
expire_after=3600)
    retry_session = retry(cache_session, retries=5, 
backoff_factor=0.2)
    openmeteo = openmeteo_requests.Client(session=retry_session)

    url = "https://api.open-meteo.com/v1/forecast"
    params = {
        "latitude": latitude,
        "longitude": longitude,
        "current": ["temperature_2m", "relative_humidity_2m", 
"precipitation", "wind_speed_10m"],
        "hourly": ["temperature_2m", "relative_humidity_2m", 
"precipitation", "wind_speed_10m"],
        "daily": ["temperature_2m_max", "temperature_2m_min", 
"relative_humidity_2m_max", "relative_humidity_2m_min", 
"precipitation_sum", "wind_speed_10m_max"],
        "timezone": "America/Los_Angeles",
        "start_date": start_date,
        "end_date": end_date,
        "models": "best_match"
    }
    responses = openmeteo.weather_api(url, params=params)
    response = responses[0]
    # Process current data
    current = response.Current()
    current_data = {
        "temperature": current.Variables(0).Value(),
        "relative_humidity": current.Variables(1).Value(),
        "precipitation": current.Variables(2).Value(),
        "wind_speed": current.Variables(3).Value(),
        "elevation": response.Elevation(),
        "timezone": response.Timezone(),
        "timezone_abbreviation": response.TimezoneAbbreviation()
    }

    # Process daily data
    daily = response.Daily()
    daily_data = {
        "temperature_2m_max": daily.Variables(0).ValuesAsNumpy()[0],
        "temperature_2m_min": daily.Variables(1).ValuesAsNumpy()[0],
        "relative_humidity_2m_max": daily.Variables(2).ValuesAsNumpy()
[0],
        "relative_humidity_2m_min": daily.Variables(3).ValuesAsNumpy()
[0],
        "precipitation_sum": daily.Variables(4).ValuesAsNumpy()[0],
        "wind_speed_10m_max": daily.Variables(5).ValuesAsNumpy()[0]
    }

    return current_data, daily_data

# NFDRS4 Fire Danger Index Calculation
def calculate_fire_danger_nfdrs4(temperature, Tmax, relative_humidity,
RHmin, precipitation, wind_speed_mph):
    """Calculates fire danger indices based on NFDRS4."""
    ffmc = 0.0
    if precipitation > 0:
        ffmc = max(0.0, 101.0 - (0.5 * precipitation))
    else:
        ffmc = max(0.0, 101.0 - (0.25 * (101.0 - 85.0)))

    dmc = max(0.0, 0.92 * temperature * (100 - relative_humidity) /
100.0)
    dc = 250.0  # Initialize DC
    dc = dc + (0.025 * (Tmax - 10) * (100 - RHmin))

    def calculate_kbdi(Tmax, rainfall, prev_kbdi=250): # set default
prev_kbdi to 250 as in main block
        """Calculates KBDI."""
        max_kbdi = 800
        if rainfall > 0:
            new_kbdi = max(0, prev_kbdi - (0.2 * rainfall))
        else:
            new_kbdi = max(0, prev_kbdi + ((Tmax - 10) * 0.3))
        return min(new_kbdi, max_kbdi)

    keetch_byram_drought_index = calculate_kbdi(Tmax, precipitation,
prev_kbdi=250) # use prev_kbdi=250

    def kbdi_to_df(kbdi):
        """Estimates Drought Factor from KBDI."""
        df = round(kbdi / 100)
        return max(0, min(8, df))

    df = kbdi_to_df(keetch_byram_drought_index)

    spread_component = 0.0
    if ffmc >= 85.0:
        spread_component = 0.208 * wind_speed_mph * np.exp(0.05039 *
ffmc)
    spread_component *= 0.01

    buildup_index = dmc + dc
    buildup_index *= 0.05

    burning_index = 0.1 * spread_component * buildup_index
    burning_index *= 0.5

    def calculate_ffdi(Tmax, RHmin, wind_speed, fuel_moisture=0.12):
        """Calculates FFDI."""
        a = 0.027
        b = 0.075
        wind_speed_kmh = wind_speed * 1.60934
        ffdi = a * (Tmax ** 2) * (100 - RHmin) * (wind_speed_kmh **
0.5) * (fuel_moisture ** b)
        ffdi *= 0.01
        return ffdi

    forest_fire_danger_index = calculate_ffdi(Tmax, RHmin,
wind_speed_mph)

    return {
        "FFMC": round(ffmc, 2),
        "DMC": round(dmc, 2),
        "DC": round(dc, 2),
        "KBDI": round(keetch_byram_drought_index, 2),
        "Wind Speed (mph)": round(wind_speed_mph, 2),
        "Buildup Index": round(buildup_index, 2),
        "Spread Component (SC)": round(spread_component, 2),
        "Burning Index (BI)": round(burning_index, 2),
        "Forest Fire Danger Index (FFDI)":
round(forest_fire_danger_index, 2)
    }
```

```
# Canadian Forest Fire Weather Index (FWI) System
def fine_fuel_moisture_code(ffmc_yda, temp, rh, ws, prec):
    """Calculates the Fine Fuel Moisture Code (FFMC)."""
    mo = 147.2 * (101.0 - ffmc_yda) / (59.5 + ffmc_yda)
    prec_above_threshold = prec > 0.5
    rf = prec - 0.5
    mo[prec_above_threshold] = (mo + 42.5 * rf * np.exp(-100.0 /
(251.0 - mo)) * (1.0 - np.exp(-6.93 / rf)) + (
                0.00057 * rf ** 2 * (np.exp(0.0365 * temp))))
[prec_above_threshold]
    mo[prec_above_threshold & (mo > 250)] = 250

    ed = 0.942 * (rh ** 0.679) + (11.0 * np.exp((rh - 100.0) / 10.0))
+ 0.18 * (21.1 - temp) * (
                1.0 - np.exp(-0.115 * rh))
    ew = 0.618 * (rh ** 0.753) + (10.0 * np.exp((rh - 100.0) / 10.0))
+ 0.18 * (21.1 - temp) * (
                1.0 - np.exp(-0.115 * rh))
    m = mo.copy()
    m[mo <= ew] = (ew - (ew - mo) / (10.0 ** (0.424 * (1.0 - ((100.0 -
rh) / 100.0) ** 1.7) + (
                0.0694 * np.sqrt(ws)) * (1.0 - ((100.0 - rh) / 100.0)
** 8))) * (
                               0.581 * np.exp(0.0365 * temp)))[mo <=
ew]
    m[(mo > ew) & (mo < ed)] = mo[(mo > ew) & (mo < ed)]
    m[mo >= ed] = (ed + 0.00046 * (mo - ed) * (42.5 - 0.0365 * temp) *
np.exp(0.0325 * (42.5 - 0.0365 * temp)))[
        mo >= ed]
    m[(mo >= ed) & (m > 1000)] = (ed + (1000.0 - ed) / 10.0 ** 0.00018
* (m - ed) * np.exp(
        0.0685 * (42.5 - 0.0365 * temp)))[(mo >= ed) & (m > 1000)]

    ffmc = 59.5 * (250.0 - m) / (147.2 + m)
    ffmc = np.clip(ffmc, 0.0, 101.0)
    return ffmc

def duff_moisture_code(dmc_yda, temp, rh, prec, lat, mon,
lat_adjust=True):
    """Calculates the Duff Moisture Code (DMC)."""
    dmc = dmc_yda.copy().astype(np.float64)
    if lat_adjust and (mon > 2 and mon < 6):
        fl = pd.Series(0.0, index=dmc.index)
        fl[lat > 0] = 1.311 + 8.766 * np.exp(-0.0825 * (58.8 + lat))
        fl[lat <= 0] = 0.210 + 0.640 * np.exp(0.0420 * (47.0 - lat))
        dmc = dmc + (fl * (1.0 - np.exp(-0.177 * prec)))
    else:
        fl = 6.0
        rk = pd.Series(0.0, index=dmc.index)
```

```python
        rk[temp > -1.1] = 1.894 * (temp[temp > -1.1] + 1.1) * (100.0 -
rh[temp > -1.1]) * fl * 0.0001

    mr = dmc.copy()
    re = 0.92 * prec - 1.27
    mr[(dmc <= 15.0) & (prec > 1.5)] = (dmc[(dmc <= 15.0) & (prec >
1.5)] + 100.0 * re[(dmc <= 15.0) & (prec > 1.5)] * (1.0 - np.exp(-
0.058 * (2.0 + re[(dmc <= 15.0) & (prec > 1.5)])))).astype(np.float64)
    mr[(dmc > 15.0) & (prec > 1.5)] = (15.0 + 100.0 * re[(dmc > 15.0)
& (prec > 1.5)] * (1.0 - np.exp(-0.020 * (6.0 + re[(dmc > 15.0) &
(prec > 1.5)])))).astype(np.float64)

    mo = mr.copy()
    mo[mr >= 150.0] = (mr[mr >= 150.0] + ((1000 / np.exp(0.1054 *
mr[mr >= 150.0])) - 1000) / np.exp(0.1209 * mr[mr >=
150.0])).astype(np.float64)

    rd = rk.copy()
    rd[temp > -2.8] = 244.72 * np.exp(0.0913 * (temp[temp > -2.8] +
2.8)) / (17.502 + np.exp(0.0913 * (temp[temp > -2.8] + 2.8))) +
rk[temp > -2.8]
    dmc = mo + 1000.0 * (1.0 - np.exp(-rd / 100.0))
    dmc[dmc < 0] = 0
    return dmc

def drought_code(dc_yda, temp, rh, prec, lat, mon, lat_adjust=True):
    """Calculates the Drought Code (DC)."""
    dc = dc_yda.copy()
    if lat_adjust and (mon > 2 and mon < 6):
        latitude = pd.Series(0.0, index=dc.index)
        latitude[lat > 0] = 65 * (np.exp(-0.1055 * (58.9 + lat)))
        latitude[lat <= 0] = 15 + 35 * (np.exp(0.0439 * (46.4 - lat)))
        dc = dc + latitude * (1.0 - np.exp(-0.0317 * prec))
    else:
        latitude = 40
        pe = latitude / (latitude + np.exp(3.73 * 0.0684 * (58.8 +
lat)))
        pe[temp > -2.8] = (0.36 * (temp[temp > -2.8] + 2.8) +
latitude) / (latitude + np.exp(3.73 * 0.0684 * (58.8 + lat)))

    pr = dc.copy()
    rw = 0.83 * prec - 1.27
    pr[(dc <= 2) & (prec > 2.8)] = dc[(dc <= 2) & (prec > 2.8)] +
100.0 * rw[(dc <= 2) & (prec > 2.8)] * np.exp(-pe[(dc <= 2) & (prec >
2.8)]) * (2.0 + np.exp(-0.0866 * dc[(dc <= 2) & (prec > 2.8)])) * (1.0
- np.exp(-6.93 / rw[(dc <= 2) & (prec > 2.8)]))
    pr[(dc > 2) & (prec > 2.8)] = dc[(dc > 2) & (prec > 2.8)] + 100.0
* rw[(dc > 2) & (prec > 2.8)] * (1.0 - np.exp(-0.0201 * (16.0 + 0.0792
* rw[(dc > 2) & (prec > 2.8)])))
    pr[pr > 1000.0] = 1000.0
    dc = pr + 1000.0 * (1.0 - np.exp(-pe))
    return dc

def initial_spread_index(ffmc, ws, fbpMod=False):
    """Calculates the Initial Spread Index (ISI)."""
    fwind = np.exp(0.05039 * ws)
    fwind[ffmc > 84.0] = np.exp(0.05039 * ws[ffmc > 84.0]) +
(ffmc[ffmc > 84.0] - 84.0) * 0.09216537 * ((ffmc[ffmc > 84.0] - 84) **
0.5)
    ffmc_factor = 0.00803 * ffmc
    ffmc_factor[(ffmc > 80) & (ffmc <= 87)] = ffmc[(ffmc > 80) & (ffmc
<= 87)] * (0.0451 - 0.45 + 0.0556 * ffmc[(ffmc > 80) & (ffmc <= 87)])
/ 7
    ffmc_factor[ffmc > 87] = 0.0732 + 0.00818 * ffmc[ffmc > 87]
    isi = ffmc_factor * fwind
    return isi

def buildup_index(dmc, dc):
    """Calculates the Buildup Index (BUI)."""
    bui = pd.Series(0.0, index=dmc.index)
    bui[dmc > 0] = np.where(dc[dmc > 0] <= 0.4 * dmc[dmc > 0], 0.8 *
dmc[dmc > 0] / (dc[dmc > 0] + 0.4 * dmc[dmc > 0]), dc[dmc > 0] - (1.0
- 0.8 * dc[dmc > 0] / (dc[dmc > 0] + 0.4 * dmc[dmc > 0])) * (0.92 +
(0.0114 * dc[dmc > 0]) ** 1.7))
    bui[(dmc <= 0) & (dc > 0)] = dc[(dmc <= 0) & (dc > 0)]
    return bui

def fire_weather_index(isi, bui):
    """Calculates the Fire Weather Index (FWI)."""
    bb = 0.1 * isi * (0.626 * bui ** 0.5 + 1.0)
    bb[bui > 80.0] = isi[bui > 80.0] * (0.000313 * bui[bui > 80.0] +
0.0234)
    fwi = bb.copy()
    fwi[bb > 1.0] = np.exp(2.72 * (0.434 * np.log(bb[bb > 1.0])))
    return fwi

def fwi_from_dataframe(df, init={'ffmc': 85, 'dmc': 6, 'dc': 15},
mon=7, out="all", lat_adjust=True, uppercase=True):
    """Calculates FWI components from a DataFrame."""
    if 'latitude' not in df.columns:
        df['latitude'] = 55  # Default latitude
    df['ffmc_yda'] = init['ffmc']
    df['dmc_yda'] = init['dmc']
    df['dc_yda'] = init['dc']
    df['rh'] = df['relative_humidity'].clip(upper=99.9999)
    df['ffmc'] = fine_fuel_moisture_code(df['ffmc_yda'],
df['temperature'], df['rh'], df['wind_speed'], df['precipitation'])
    df['dmc'] = duff_moisture_code(df['dmc_yda'], df['temperature'],
df['rh'], df['precipitation'], df['latitude'], mon, lat_adjust)
    df['dc'] = drought_code(df['dc_yda'], df['temperature'], df['rh'],
df['precipitation'], df['latitude'], mon, lat_adjust)
    df['isi'] = initial_spread_index(df['ffmc'], df['wind_speed'])
    df['bui'] = buildup_index(df['dmc'], df['dc'])
    df['fwi'] = fire_weather_index(df['isi'], df['bui'])
    df['dsr'] = 0.0272 * (df['fwi'] ** 1.77)

    if out == "fwi":
        fwi_vars = ['ffmc', 'dmc', 'dc', 'isi', 'bui', 'fwi', 'dsr']
        new_fwi = df[fwi_vars]
    else:
        new_fwi = df

    if uppercase:
        new_fwi.columns = [col.upper() for col in new_fwi.columns]
    return new_fwi

# Modified FFWI Calculation
def calculate_kbdi(max_temp_f, precip_in, prev_kbdi=0):
    """Calculates the Keetch-Byram Drought Index (KBDI)."""
    drought_factor = 0.968 * np.exp(0.0875 * max_temp_f + 1.5552) -
8.258
    if precip_in > 0.2:
        net_precip = precip_in - 0.2
    else:
        net_precip = 0
    kbdi = max(0, min(800, prev_kbdi + drought_factor - net_precip))
    return kbdi

def calc_ffwi(temp, rh, wind, kbdi=None):
    """Calculates the Fosberg Fire Weather Index (FFWI) and the
modified FFWI (mFFWI)."""
    if rh < 10:
        m = 0.03229 + 0.281073 * rh - 0.000578 * rh * temp
    elif 10 <= rh <= 50:
        m = 2.22749 + 0.160107 * rh - 0.01478 * temp
    else:
        m = 21.0606 + 0.005565 * rh**2 - 0.00035 * rh * temp -
0.483199 * rh

    ffwi = np.exp(0.05039 * temp - 0.02016 * rh + 0.00504 * wind)

    if kbdi is not None:
        fa = 1 + (kbdi / 100)
        mffwi = ffwi * fa
    else:
        mffwi = None

    return ffwi, mffwi
```

Part 2: Map Generation with Folium

```
def create_fire_index_map(grid_data, index_combination, base_map):
    """
    Creates a Folium map with a choropleth layer representing the
    specified fire index combination, focused on Los Angeles County,
    using a provided base map.
    """
    # Create a feature group for the fire index combination
    index_layer = folium.FeatureGroup(name=index_combination)

    # Get the colormaps
    ffdi_colormap = folium.LinearColormap(
        colors=['green', 'yellow', 'orange', 'red'],
        vmin=min(data['FFDI'] for data in grid_data.values() if
data['FFDI'] is not None),
        vmax=max(data['FFDI'] for data in grid_data.values() if
data['FFDI'] is not None),
        caption='FFDI Value'
    )
    fwi_colormap = folium.LinearColormap(
        colors=['green', 'yellow', 'orange', 'red'],
        vmin=min(data['FWI'] for data in grid_data.values() if
data['FWI'] is not None),
        vmax=max(data['FWI'] for data in grid_data.values() if
data['FWI'] is not None),
        caption='FWI Value'
    )
    mffwi_colormap = folium.LinearColormap(
        colors=['green', 'yellow', 'orange', 'red'],
        vmin=min(data['mFFWI'] for data in grid_data.values() if
data['mFFWI'] is not None),
        vmax=max(data['mFFWI'] for data in grid_data.values() if
data['mFFWI'] is not None),
        caption='mFFWI Value'
    )

    # Define the mapping for index combinations to indices and radii
    combination_mapping = {
        'FWI': (['FWI'], [5]),
        'FFDI': (['FFDI'], [5]),
        'mFFWI': (['mFFWI'], [5]),
        'FWI-FFDI': (['FWI', 'FFDI'], [5, 3]),
        'FWI-mFFWI': (['FWI', 'mFFWI'], [5, 3]),
        'FFDI-mFFWI': (['FFDI', 'mFFWI'], [5, 3]),
        'FWI-FFDI-mFFWI': (['FWI', 'FFDI', 'mFFWI'], [5, 3, 1])
    }

    indices, radii = combination_mapping.get(index_combination, ([],
[]))
```

```
    num_indices = len(indices)

    # Add data points to the feature group
    for (lat, lon), data in grid_data.items():
        values = [data[index] for index in indices if index in data
and data[index] is not None]
        if not values:
            continue

        # Create concentric circles or a single circle
        circles = []
        for i, index_type in enumerate(indices):
            value = data.get(index_type)
            if value is not None:
                colormap = {
                    'FFDI': ffdi_colormap,
                    'FWI': fwi_colormap,
                    'mFFWI': mffwi_colormap
                }.get(index_type)

                color = colormap(value) if colormap else 'blue'
                circle = folium.CircleMarker(
                    location=[lat, lon],
                    radius=radii[i],
                    color=color,
                    fill=True,
                    fill_color=color,
                    fill_opacity=0.7,
                    interactive=False if i > 0 else True,  # Address
event propagation
                    zIndexOffset=1000 - i  # Address z-index
                )
                circles.append(circle)

        # Add a popup to the outermost circle
        if circles:
            # Format the popup content to avoid redundancy
            popup_content = []
            for i, index_type in enumerate(indices):
                value = data.get(index_type)
                if value is not None:
                    popup_content.append(f"{index_type}: {value:.2f}")

            # Join the values to avoid repetitive index names in the
popup
            popup_html = "<br>".join(popup_content)
            popup = folium.Popup(popup_html, max_width=300)
            circles[0].add_child(popup)

            # Add circles to the feature group
            for circle in circles:
                circle.add_to(index_layer)

    return base_map, index_layer
```

Part 3: Main Execution Block

```
if __name__ == "__main__":
    # Database setup and connection
    conn = sqlite3.connect('fire_weather.db')
    cursor = conn.cursor()

    # Create table (if it doesn't exist)
    create_table_sql = """
        CREATE TABLE IF NOT EXISTS fire_data (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            latitude REAL NOT NULL,
            longitude REAL NOT NULL,
            date TEXT NOT NULL,
            ffdi REAL,
            fwi REAL,
            mffwi REAL,
            temperature REAL,
            relative_humidity REAL,
            wind_speed REAL,
            precipitation REAL
        );
    """
    cursor.execute(create_table_sql)

    # Create the LA County base map with CartoDB Positron tiles
    combined_map, la_county_layer, base_map_layer = 
create_la_county_base_map()

    if combined_map is None or la_county_layer is None:
        print("Failed to create base map. Exiting.")
    else:
        # Ensure both CartoDB Positron and LA County Boundary are 
added before other layers
        #la_county_layer.add_to(combined_map)  # Add LA County 
boundary layer to the map
        base_map_layer.add_to(combined_map) # Add the base map layer 
(CartoDB Positron)

        # Today's date for the Open-Meteo API
        today = pd.to_datetime("today").strftime("%Y-%m-%d")
        grid_spacing = 0.1  # Define grid spacing here, same as 
original
```

```
        # Extend longitude range for additional columns
        extended_min_lon = min_lon - 6 * grid_spacing  # Six rows west
        extended_max_lon = max_lon + 2 * grid_spacing  # Two rows east

        # Generate the extended grid
        la_county_grid = [
            (lat, lon)
            for lat in np.arange(min_lat, max_lat + grid_spacing,
grid_spacing)
            for lon in np.arange(extended_min_lon, extended_max_lon +
grid_spacing, grid_spacing)
        ]

        for lat, lon in la_county_grid:
            try:
                # Fetch weather data and calculate fire indices
                current_weather_data, daily_weather_data =
get_weather_data(lat, lon, today, today)

                # Prepare data for NFDRS4
                tmax = daily_weather_data["temperature_2m_max"]
                rhmin = daily_weather_data["relative_humidity_2m_min"]
                wind_speed_mph = current_weather_data["wind_speed"] *
0.621371  # Convert km/h to mph
                precipitation_inches =
daily_weather_data["precipitation_sum"] * 0.0393701  # Convert mm to
inches

                # Calculate NFDRS4 indices
                nfdrs4_data = calculate_fire_danger_nfdrs4(
                    current_weather_data["temperature"], tmax,
current_weather_data["relative_humidity"],
                    rhmin, precipitation_inches, wind_speed_mph
                )

                # Prepare data for FWI
                fwi_df = pd.DataFrame({
                    "temperature":
[current_weather_data["temperature"]],
                    "relative_humidity":
[current_weather_data["relative_humidity"]],
                    "wind_speed":
[current_weather_data["wind_speed"]],
                    "precipitation":
[current_weather_data["precipitation"]],
                    "latitude": [lat]
                })

                # Calculate FWI
                fwi_result = fwi_from_dataframe(fwi_df,
mon=pd.to_datetime(today).month)

                # Calculate mFFWI
                mffwi_ffwi, mffwi_val = calc_ffwi(
                    tmax * 9 / 5 + 32, rhmin, wind_speed_mph,  #
Convert to Fahrenheit
                    kbdi=nfdrs4_data["KBDI"]  # Use KBDI from NFDRS4
                )

                # Store data in the database
                sql_insert = """
                    INSERT INTO fire_data (latitude, longitude, date,
ffdi, fwi, mffwi, temperature, relative_humidity, wind_speed,
precipitation)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
                """
                data_to_insert = (lat, lon, today, nfdrs4_data["Forest
Fire Danger Index (FFDI)"], fwi_result["FWI"].iloc[0], mffwi_val,
current_weather_data["temperature"],
current_weather_data["relative_humidity"],
current_weather_data["wind_speed"],
current_weather_data["precipitation"])
                cursor.execute(sql_insert, data_to_insert)
                conn.commit()

            except Exception as e:
                print(f"Error processing data for ({lat}, {lon}):
{e}")
                grid_data[(lat, lon)] = {"FFDI": None, "FWI":
None,"mFFWI": None} #Assigning default value of None


        # Retrieve data from the database for map creation
        cursor.execute("SELECT latitude, longitude, ffdi, fwi, mffwi
FROM fire_data WHERE date = ?", (today,))
        grid_data_from_db = cursor.fetchall()

        grid_data = {}
        for row in grid_data_from_db:
            grid_data[(row[0], row[1])] = {
                "FFDI": row[2],
                "FWI": row[3],
                "mFFWI": row[4]
            }

 # Define the colormaps
        ffdi_colormap = folium.LinearColormap(
        colors=['green', 'yellow', 'orange', 'red'],
        vmin=min(data['FFDI'] for data in grid_data.values() if
data['FFDI'] is not None), vmax=max(data['FFDI'] for data in
grid_data.values() if data['FFDI'] is not None),
        caption='FFDI Value'
        )
        fwi_colormap = folium.LinearColormap(
           colors=['green', 'yellow', 'orange', 'red'],
           vmin=min(data['FWI'] for data in grid_data.values() if
data['FWI'] is not None), vmax=max(data['FWI'] for data in
grid_data.values() if data['FWI'] is not None),
            caption='FWI Value'
        )
        mffwi_colormap = folium.LinearColormap(
            colors=['green', 'yellow', 'orange', 'red'],
            vmin=min(data['mFFWI'] for data in grid_data.values() if
data['mFFWI'] is not None), vmax=max(data['mFFWI'] for data in
grid_data.values() if data['mFFWI'] is not None),
            caption='mFFWI Value'
        )

        # Define the index combinations for the layers
        index_combinations = [
            'FWI', 'FFDI', 'mFFWI', 'FWI-FFDI', 'FWI-mFFWI', 'FFDI-
mFFWI', 'FWI-FFDI-mFFWI'
        ]

        # Create feature groups for each index combination
        base_layers = {}  # Use a dictionary to store base layers

        for index_combination in index_combinations:
            combined_map, index_layer =
create_fire_index_map(grid_data, index_combination, combined_map)
            feature_group =
folium.FeatureGroup(name=index_combination, control=True,
overlay=False)
            index_layer.add_to(feature_group) # Add the index layer to
the feature group
            feature_group.add_to(combined_map) #Add the feature group
to the map
            base_layers[index_combination] = feature_group # Store
FeatureGroup instead of raw layer


            # Get HTML representation of the colormaps
        ffdi_colormap_html = ffdi_colormap._repr_html_()
        fwi_colormap_html = fwi_colormap._repr_html_()
        mffwi_colormap_html = mffwi_colormap._repr_html_()

        #Format numbers in colormap HTML to two decimal places
        ffdi_colormap_html = re.sub(r"(\d+\.\d{2})\d+", r"\1",
ffdi_colormap_html)
        fwi_colormap_html = re.sub(r"(\d+\.\d{2})\d+", r"\1",
fwi_colormap_html)
        mffwi_colormap_html = re.sub(r"(\d+\.\d{2})\d+", r"\1",
mffwi_colormap_html)

        #Remove the min and max value lines from colormap HTML
        ffdi_colormap_html = ffdi_colormap_html.replace('<li
style="text-align: center; list-style: none; display: block; line-
height: 18px; height: 18px; width: 100.0%; margin-left: 0%; margin-
bottom: -2px;"><span style="text-align: right; display: block; width:
40px; float: left; height: 100.0%; background:
rgba(0,0,0,0);">{:.2f}</span><span style="display: block; width: 5.0%;
float: left; height: 100.0%; background: rgba(0,0,0,0);"><svg
class="colorbar" height="18" width="10.0" style="float: right;"><line
stroke-width="1" x1="0" x2="0" y1="1" y2="16"
stroke="#000000"></line></svg></span>', '').replace('<span
style="display: block; width: 5.0%; float: left; height: 100.0%;
background: rgba(0,0,0,0);"><svg class="colorbar" height="18"
width="10.0" style="float: right;"><line stroke-width="1" x1="0"
x2="0" y1="1" y2="16" stroke="#000000"></line></svg></span><span
style="display: block; width: 40px; float: left; height: 100.0%;
background: rgba(0,0,0,0);">{:.2f}</span></li>', '')

        fwi_colormap_html = fwi_colormap_html.replace('<li
style="text-align: center; list-style: none; display: block; line-
height: 18px; height: 18px; width: 100.0%; margin-left: 0%; margin-
bottom: -2px;"><span style="text-align: right; display: block; width:
40px; float: left; height: 100.0%; background:
rgba(0,0,0,0);">{:.2f}</span><span style="display: block; width: 5.0%;
float: left; height: 100.0%; background: rgba(0,0,0,0);"><svg
class="colorbar" height="18" width="10.0" style="float: right;"><line
stroke-width="1" x1="0" x2="0" y1="1" y2="16"
stroke="#000000"></line></svg></span>', '').replace('<span
style="display: block; width: 5.0%; float: left; height: 100.0%;
background: rgba(0,0,0,0);"><svg class="colorbar" height="18"
width="10.0" style="float: right;"><line stroke-width="1" x1="0"
x2="0" y1="1" y2="16" stroke="#000000"></line></svg></span><span
style="display: block; width: 40px; float: left; height: 100.0%;
background: rgba(0,0,0,0);">{:.2f}</span></li>', '')

        mffwi_colormap_html = mffwi_colormap_html.replace('<li
style="text-align: center; list-style: none; display: block; line-
height: 18px; height: 18px; width: 100.0%; margin-left: 0%; margin-
bottom: -2px;"><span style="text-align: right; display: block; width:
40px; float: left; height: 100.0%; background:
rgba(0,0,0,0);">{:.2f}</span><span style="display: block; width: 5.0%;
float: left; height: 100.0%; background: rgba(0,0,0,0);"><svg
class="colorbar" height="18" width="10.0" style="float: right;"><line
stroke-width="1" x1="0" x2="0" y1="1" y2="16"
stroke="#000000"></line></svg></span>', '').replace('<span
style="display: block; width: 5.0%; float: left; height: 100.0%;
```

```
background: rgba(0,0,0,0);"><svg class="colorbar" height="18"
width="10.0" style="float: right;"><line stroke-width="1" x1="0"
x2="0" y1="1" y2="16" stroke="#000000"></line></svg></span><span
style="display: block; width: 40px; float: left; height: 100.0%;
background: rgba(0,0,0,0);">{:.2f}</span></li>', '')


         # Create a Div to hold the colormaps and add it to the map
        colormap_div = folium.Element(
        f"<div id='colormap' style='position: absolute; bottom: 30px;
left: 5px; display: inline-flex; gap: 16px; z-index: 1000; background-
color: rgba(255, 255, 255, 0.7); width: 99%;'>"
         + ffdi_colormap_html
         + fwi_colormap_html
         + mffwi_colormap_html
         + "</div>")

        combined_map.get_root().html.add_child(colormap_div)

        # Add LayerControl to the combined map with only base layers
        control = folium.LayerControl(collapsed=False,
exclusive_groups=["Fire Indices"])  #Create a LayerControl object
        combined_map.add_child(control) # Add the LayerControl to the
map

        # Save the map to an HTML file
        combined_map.save("final_map_with_colormaps.html")  # Save the
map
        print("Map generated successfully for Los Angeles County!")

    conn.close()
```

```
Map generated successfully for Los Angeles County!
```