Google Earth Engine (GEE) Codes

1. Data: Monthly Mean Temperature and Monthly Mean Relative Humidity (1990 – 2023)

Output file name: Monthly Mean Temp_RH.csv

Code:

```
/**
 * GENERATE ERA5-LAND MONTHLY CLIMATE TRENDS (1990-2030) FOR ALL MONTHS
 * Output: 12 charts for Mean Temperature and 12 charts for Relative Humidity (Jan-Dec)
 * ENHANCEMENT: Export data to Google Drive as CSV.
 */

// 1. Define the Area of Interest (AOI) for Kolkata, India.
var kolkata_aoi = ee.Geometry.BBox(88.25, 22.45, 88.50, 22.70);

// Define the time period.
var startYear = 1990;
var endYear = 2030;
var startMonth = 1; // January
var endMonth = 12; // December

// 2. Load the ERA5-LAND MONTHLY dataset.
var era5_monthly = ee.ImageCollection("ECMWF/ERA5_LAND/MONTHLY");

// 3. Generate lists for years and months to iterate over.
var years = ee.List.sequence(startYear, endYear);
var months = ee.List.sequence(startMonth, endMonth);

// List of month names for chart titles/labels
var monthNames = ee.List(['January', 'February', 'March', 'April', 'May', 'June',
                          'July', 'August', 'September', 'October', 'November', 'December']);

// 4. Function to calculate mean climate data for a specific month.
var calculateMonthlyData = function(year) {
  year = ee.Number(year);

  // Define a nested function to iterate over months within a year.
  var calculateDataForMonth = function(month) {
    month = ee.Number(month);

    var startDate = ee.Date.fromYMD(year, month, 1);
    var endDate = startDate.advance(1, 'month');

    var monthlyData = era5_monthly
      .filterDate(startDate, endDate)
      .filterBounds(kolkata_aoi);

    var monthlyCount = monthlyData.size();
    var monthlyImage = monthlyData.mean();

    // --- Calculate Mean Temperature (°C) ---
```

```
          var meanTempC = ee.Algorithms.If(monthlyCount.gt(0),
            monthlyImage.select('temperature_2m')
              .subtract(273.15) // Convert K to C
              .reduceRegion({
                reducer: ee.Reducer.mean(),
                geometry: kolkata_aoi,
                scale: 9000,
                bestEffort: true
              }).get('temperature_2m'),
            null
          );

          // --- Calculate Mean Relative Humidity (%) ---
          var meanRH = ee.Algorithms.If(monthlyCount.gt(0),
            (function() {
              var                  meanTempK_Image                  =
monthlyImage.select('temperature_2m');
              var               meanDewpointK_Image                 =
monthlyImage.select('dewpoint_temperature_2m');
              var A = 17.625; var B = 243.04;
              var T_C_Image = meanTempK_Image.subtract(273.15);
              var Td_C_Image = meanDewpointK_Image.subtract(273.15);
              var es = T_C_Image.expression('6.1094 * exp((A * T) /
(B + T))', { 'T': T_C_Image, 'A': A, 'B': B });
              var e = Td_C_Image.expression('6.1094 * exp((A * Td) /
(B + Td))', { 'Td': Td_C_Image, 'A': A, 'B': B });
              var              relativeHumidityImage               =
e.divide(es).multiply(100).rename('mean_rh_percent');

              return relativeHumidityImage.reduceRegion({
                reducer: ee.Reducer.mean(),
                geometry: kolkata_aoi,
                scale: 9000,
                bestEffort: true
              }).get('mean_rh_percent');
            })(),
            null
          );

          // Create a feature.
          return ee.Feature(null, {
            'year': year,
            'month': month,
            'label':             ee.Number(year).format('%d').cat('-
').cat(ee.Number(month).format('%02d')),
            'mean_temp_c': meanTempC,
            'mean_rh_percent': meanRH,
          });
        };

        // Map the month-calculation function over the list of
months.
        return months.map(calculateDataForMonth);
      };


      // 5. Create the complete Feature Collection.
      var nested_features = years.map(calculateMonthlyData);
      var flat_list_of_features = nested_features.flatten();
      var                    climate_features                     =
      ee.FeatureCollection(flat_list_of_features);
```

```
// Filter out features with null values
var valid_features = climate_features
  .filter(ee.Filter.notNull(['mean_temp_c',
'mean_rh_percent']));

// ------------------------------------------------------------
----------------------------------------

// 6. Generate Separate Monthly Charts 📊

// Function to generate a single chart (returns the chart
object)
var generateChart = function(monthIndex, variableName,
yAxisTitle, color, trendColor) {

    var monthNum = ee.Number(monthIndex).add(1);
    var monthStr = ee.String(monthNames.get(monthIndex));

    // Filter the entire feature collection for the specific
month
    var monthly_features =
climate_features.filter(ee.Filter.eq('month', monthNum));

    // Construct the title server-side
    var title = ee.String('Mean ').cat(variableName).cat('
Trend for ').cat(monthStr).cat(' (1990 - 2030)');

    // Create the chart.
    var chart = ui.Chart.feature.byFeature({
        features: monthly_features,
        xProperty: 'year',
        yProperties: [variableName]
      })
      .setChartType('ScatterChart')
      .setOptions({
        title: title,
        hAxis: {
          title: 'Year',
          gridlines: {color: '#cccccc', count: 10},
          format: '####',
          viewWindow: {
            min: startYear,
            max: endYear
          }
        },
        vAxis: {
          title: yAxisTitle,
          gridlines: {color: '#cccccc', count: 5}
        },
        series: {0: {color: color, pointSize: 4}},
        legend: {position: 'none'},
        trendlines: {0: {
          type: 'linear', color: trendColor, lineWidth: 3,
opacity: 0.7,
          visibleInLegend: true, labelInLegend: 'Linear Trend'
        }}
      });

    return chart; // Return the chart object
};
```

```
// Function to generate and print all 24 charts
var generateAndPrintAllCharts = function() {

    // --- Temperature Charts (12 charts) ---
    months.getInfo().forEach(function(month) {
        var monthIndex = month - 1;
        var monthName = monthNames.getInfo()[monthIndex];

        var temp_chart = generateChart(
            ee.Number(monthIndex),
            'mean_temp_c',
            'Mean Temperature (°C)',
            'red',
            'darkred'
        );

        print('🌡️ Temperature: ' + monthName + ' Chart:',
temp_chart);
    });

    // --- Relative Humidity Charts (12 charts) ---
    months.getInfo().forEach(function(month) {
        var monthIndex = month - 1;
        var monthName = monthNames.getInfo()[monthIndex];

        var rh_chart = generateChart(
            ee.Number(monthIndex),
            'mean_rh_percent',
            'Mean Relative Humidity (%)',
            'blue',
            'darkblue'
        );

        print('💧 Relative Humidity: ' + monthName + '
Chart:', rh_chart);
    });
};

// Execute the chart generation and printing
generateAndPrintAllCharts();

// ------------------------------------------------------------
----------------------------------------

// 7. Determine and print the last year/month with valid data
var                        valid_dates                        =
valid_features.aggregate_array('label').sort();

var last_valid_label = ee.Algorithms.If(
  valid_dates.size().gt(0),
  valid_dates.get(valid_dates.size().subtract(1)),
  'No Valid Data Found'
);
print('🔍 Last Valid Data Point Found (Year-Month):',
last_valid_label);


// 8. Optionally, center the map on Kolkata and display the
AOI.
```

```
Map.centerObject(kolkata_aoi, 9);
Map.addLayer(kolkata_aoi, {color: '00FF00', opacity: 0.4},
'Kolkata AOI Polygon');

// 9. ⭐ EXPORT DATA TO GOOGLE DRIVE (New Enhancement) ⭐

Export.table.toDrive({
  collection: valid_features,
  description: 'Kolkata_Monthly_Climate_Export', // Name of
the task in the Tasks tab
  fileNamePrefix: 'kolkata_monthly_climate_data', // File name
in Drive
  folder: 'GEE_Exports', // Specify a folder in your Google
Drive (will be created if it doesn't exist)
  fileFormat: 'CSV', // Export as CSV
  selectors: ['year', 'month', 'label', 'mean_temp_c',
'mean_rh_percent'] // Select the columns to include
});

print('✅ Data Export Task Queued. Check the "Tasks" tab (top right)
to Run it.');
```

2. Data: Monthly Max and Min Temperature (1990 – 2023)

Output file name: Monthly Max Min Temp.csv

Code:

```
/**
 * GENERATE ERA5-LAND MONTHLY MAX/MIN TEMPERATURE TRENDS (1990
- Present)
 * FIX: Improved data handling to ensure 'year' property is
always non-null.
 */

// 1. Define the Area of Interest (AOI) for Kolkata, India.
var kolkata_aoi = ee.Geometry.BBox(88.25, 22.45, 88.50,
22.70);

// Define the time period.
var startYear = 1990;
// Current year (will be 2025 until January 1st, 2026, using
GEE's server time)
var endYear = ee.Date(Date.now()).get('year').getInfo();

// 2. Define Months
var startMonth = 1; // January
var endMonth = 12; // December
var months = ee.List.sequence(startMonth, endMonth);
var monthNames = ee.List(['January', 'February', 'March',
'April', 'May', 'June',
                          'July', 'August', 'September',
'October', 'November', 'December']);

// 3. Load the ERA5-LAND HOURLY dataset.
var era5_hourly =
ee.ImageCollection("ECMWF/ERA5_LAND/HOURLY");

// 4. Generate the list of years to iterate over.
var years = ee.List.sequence(startYear, endYear);
```

```
// 5. Function to calculate daily Max and Min temperature for
a specific month/year combination.
var calculateMonthlyData = function(year) {
  year = ee.Number(year);

  var calculateDataForMonth = function(month) {
    month = ee.Number(month);

    // Define the essential feature properties first, which
always exist.
    var baseProperties = {
      'year': year,
      'month': month,
      'label':            ee.Number(year).format('%d').cat('-
').cat(ee.Number(month).format('%02d')),
    };

    var startDate = ee.Date.fromYMD(year, month, 1);
    var endDate = startDate.advance(1, 'month');

    // Filter the HOURLY collection for the specific month.
    var hourlyData = era5_hourly
      .filterDate(startDate, endDate)
      .filterBounds(kolkata_aoi);

    var monthlyCount = hourlyData.size();
    var hasData = monthlyCount.gt(0);

    // --- Define calculation block (only executed if data is
present) ---
    var calculatedData = ee.Algorithms.If(hasData,
      ee.Dictionary({
        'max_temp_c':
hourlyData.select('temperature_2m').max()
          .subtract(273.15)
          .reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: kolkata_aoi,
            scale: 9000,
            bestEffort: true
          }).get('temperature_2m'),

        'min_temp_c':
hourlyData.select('temperature_2m').min()
          .subtract(273.15)
          .reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: kolkata_aoi,
            scale: 9000,
            bestEffort: true
          }).get('temperature_2m')
      }),
      // Return a dictionary of nulls if no data is present
      ee.Dictionary({'max_temp_c': null, 'min_temp_c': null})
    );

    // Merge the base properties with the calculated (or null)
temperature data.
    var                   finalProperties                   =
ee.Dictionary(baseProperties).combine(calculatedData);
```

```
    // Return the feature with all properties defined (some
may be null).
    return ee.Feature(null, finalProperties);
  };

  // Map the month-calculation function over the list of
months.
  return months.map(calculateDataForMonth);
};


// 6. Map the function over the years and flatten the result
to create the Feature Collection.
var nested_features = years.map(calculateMonthlyData);
var flat_list_of_features = nested_features.flatten();
var                    climate_features                    =
ee.FeatureCollection(flat_list_of_features);

// Filter out features where EITHER max_temp_c OR min_temp_c
is null.
// NOTE: 'year' is now guaranteed to be non-null.
var valid_features = climate_features
  .filter(ee.Filter.notNull(['max_temp_c']));
  // We only need to check one temp variable since they are
calculated together


// ------------------------------------------------------------
---------------------------------------


// 7. Generate and Display 24 Monthly Charts

// Function to generate a single chart (returns the chart
object)
var generateChart = function(monthIndex, variableName, color,
trendColor) {

    var monthNum = ee.Number(monthIndex).add(1);
    var monthStr = ee.String(monthNames.get(monthIndex));

    // Filter the VALID feature collection for the specific
month
    var                  monthly_features                  =
valid_features.filter(ee.Filter.eq('month', monthNum));

    // Determine the variable name and title dynamically
    var varTitle = ee.String(variableName).replace('_temp_c',
'').replace('max', 'Maximum').replace('min', 'Minimum');
    var  chartTitle  =  varTitle.cat('  Temperature  Trend  for
').cat(monthStr).cat(' (1990 - Present)');

    // Create the chart.
    var chart = ui.Chart.feature.byFeature({
        features: monthly_features,
        xProperty: 'year',
        yProperties: [variableName]
      })
      .setChartType('ScatterChart')
      .setOptions({
        title: chartTitle,
        hAxis: {
```

```
              title: 'Year',
              gridlines: {color: '#cccccc', count: 10},
              format: '####',
              viewWindow: {
                min: startYear,
                max: endYear
              }
            },
            vAxis: {
              title: 'Temperature (°C)',
              gridlines: {color: '#cccccc', count: 5}
            },
            series: {0: {color: color, pointSize: 4}},
            legend: {position: 'none'},
            trendlines: {0: {
              type: 'linear', color: trendColor, lineWidth: 3,
opacity: 0.7,
              visibleInLegend: true, labelInLegend: 'Linear Trend'
            }}
          });

        return chart; // Return the chart object
    };

    // Function to generate and print all 24 charts
    var generateAndPrintAllCharts = function() {

        // --- Max Temperature Charts (12 charts) ---
        months.getInfo().forEach(function(month) {
            var monthIndex = month - 1;
            var monthName = monthNames.getInfo()[monthIndex];

            var max_chart = generateChart(
                ee.Number(monthIndex),
                'max_temp_c',
                'red',
                'darkred'
            );

            print('🌡️ MAX Temp: ' + monthName + ' Chart:',
max_chart);
        });

        // --- Min Temperature Charts (12 charts) ---
        months.getInfo().forEach(function(month) {
            var monthIndex = month - 1;
            var monthName = monthNames.getInfo()[monthIndex];

            var min_chart = generateChart(
                ee.Number(monthIndex),
                'min_temp_c',
                'blue',
                'darkblue'
            );

            print('🧊 MIN Temp: ' + monthName + ' Chart:',
min_chart);
        });
    };

    // Execute the chart generation and printing
```

```
generateAndPrintAllCharts();

// ------------------------------------------------------------
------------------------------------

// 8. Map and Export (Optional)

// Determine and print the last year/month with valid data
var valid_labels =
valid_features.aggregate_array('label').sort();

var last_valid_label = ee.Algorithms.If(
  valid_labels.size().gt(0),
  valid_labels.get(valid_labels.size().subtract(1)),
  'No Valid Data Found'
);
print('🔍 Last Valid Data Point Found (Year-Month):',
last_valid_label);


// Optionally, center the map on Kolkata and display the AOI.
Map.centerObject(kolkata_aoi, 9);
Map.addLayer(kolkata_aoi, {color: '00FF00', opacity: 0.4},
'Kolkata AOI Polygon');

// Data Export (Uncomment to queue)

Export.table.toDrive({
  collection: valid_features,
  description: 'Kolkata_Monthly_MaxMin_Temp_Export',
  fileNamePrefix: 'kolkata_monthly_maxmin_temp_data',
  folder: 'GEE_Exports',
  fileFormat: 'CSV',
  selectors: ['year', 'month', 'label', 'max_temp_c',
'min_temp_c']
});

print('✅ Data Export Task Queued. Check the "Tasks" tab to Run
it.');
```

3. Data: Monthly Mean Temp and Absolute Humidity (1990 – 2023)

Output file name: Monthly Mean Temp_AH.csv

Code:

```
/**
 * GENERATE ERA5-LAND MONTHLY CLIMATE TRENDS (1990-2030) FOR
ALL MONTHS
 * Output: 12 charts for Mean Temperature and 12 charts for
Absolute Humidity (Jan-Dec)
 * ENHANCEMENT: Replaced Relative Humidity with Absolute
Humidity (g/m^3).
 */

// 1. Define the Area of Interest (AOI) for Kolkata, India.
var kolkata_aoi = ee.Geometry.BBox(88.25, 22.45, 88.50,
22.70);

// Define the time period.
var startYear = 1990;
var endYear = 2030;
var startMonth = 1; // January
```

```
var endMonth = 12; // December

// Constants for Absolute Humidity Calculation
var Mw = 0.018015;  // Molar mass of water vapor (kg/mol)
var R = 8.314;      // Universal Gas Constant (J/(mol*K))

// 2. Load the ERA5-LAND MONTHLY dataset.
var                        era5_monthly                        =
ee.ImageCollection("ECMWF/ERA5_LAND/MONTHLY");

// 3. Generate lists for years and months to iterate over.
var years = ee.List.sequence(startYear, endYear);
var months = ee.List.sequence(startMonth, endMonth);

// List of month names for chart titles/labels
var  monthNames  =  ee.List(['January',  'February',  'March',
'April', 'May', 'June',
                            'July',    'August',    'September',
'October', 'November', 'December']);

// 4. Function to calculate mean climate data for a specific
month.
var calculateMonthlyData = function(year) {
  year = ee.Number(year);

  // Define a nested function to iterate over months within a
year.
  var calculateDataForMonth = function(month) {
    month = ee.Number(month);

    var startDate = ee.Date.fromYMD(year, month, 1);
    var endDate = startDate.advance(1, 'month');

    var monthlyData = era5_monthly
      .filterDate(startDate, endDate)
      .filterBounds(kolkata_aoi);

    var monthlyCount = monthlyData.size();
    var monthlyImage = monthlyData.mean();

    // --- Calculate Mean Temperature (°C) ---
    var meanTempC = ee.Algorithms.If(monthlyCount.gt(0),
      monthlyImage.select('temperature_2m')
        .subtract(273.15) // Convert K to C
        .reduceRegion({
          reducer: ee.Reducer.mean(),
          geometry: kolkata_aoi,
          scale: 9000,
          bestEffort: true
        }).get('temperature_2m'),
      null
    );

    // --- Calculate Mean Absolute Humidity (g/m^3) ---
    var meanAH = ee.Algorithms.If(monthlyCount.gt(0),
      (function() {
        var                  meanTempK_Image                  =
monthlyImage.select('temperature_2m');
        var                meanDewpointK_Image                =
monthlyImage.select('dewpoint_temperature_2m');
```

```
        // Vapour pressure calculation constants (Magnus-
Tetens formula)
        var A = 17.625; var B = 243.04;

        // Convert Dewpoint to Celsius
        var Td_C_Image = meanDewpointK_Image.subtract(273.15);

        // Calculate Actual Vapor Pressure (e) in hPa
(millibars)
        // e = 6.1094 * exp((A * Td) / (B + Td))
        var e_hPa = Td_C_Image.expression('6.1094 * exp((A *
Td) / (B + Td))', {
            'Td': Td_C_Image,
            'A': A,
            'B': B
        });

        // Convert Actual Vapor Pressure (e) from hPa to
Pascals (Pa)
        var e_Pa = e_hPa.multiply(100);

        // Calculate Absolute Humidity (AH) in kg/m^3 (using
the Ideal Gas Law for water vapor)
        // AH_kg_m3 = (e * Mw) / (R * Tk)
        var                  AH_kg_m3_Image                  =
e_Pa.multiply(Mw).divide(meanTempK_Image.multiply(R)).rename(
'mean_ah_kg_m3');

        // Convert AH from kg/m^3 to g/m^3 (multiply by 1000)
        var              absoluteHumidityImage              =
AH_kg_m3_Image.multiply(1000).rename('mean_ah_g_m3');

        return absoluteHumidityImage.reduceRegion({
          reducer: ee.Reducer.mean(),
          geometry: kolkata_aoi,
          scale: 9000,
          bestEffort: true
        }).get('mean_ah_g_m3');
      })(),
      null
    );

    // Create a feature.
    return ee.Feature(null, {
      'year': year,
      'month': month,
      'label':            ee.Number(year).format('%d').cat('-
').cat(ee.Number(month).format('%02d')),
      'mean_temp_c': meanTempC,
      'mean_ah_g_m3': meanAH, // Changed from mean_rh_percent
    });
  };

  // Map the month-calculation function over the list of
months.
  return months.map(calculateDataForMonth);
};


// 5. Create the complete Feature Collection.
var nested_features = years.map(calculateMonthlyData);
```

```
var flat_list_of_features = nested_features.flatten();
var climate_features =
ee.FeatureCollection(flat_list_of_features);

// Filter out features with null values
var valid_features = climate_features
  .filter(ee.Filter.notNull(['mean_temp_c',
'mean_ah_g_m3'])); // Filter updated

// ------------------------------------------------------------
----------------------------------------

// 6. Generate Separate Monthly Charts 📊

// Function to generate a single chart (returns the chart
object)
var generateChart = function(monthIndex, variableName,
yAxisTitle, color, trendColor) {

    var monthNum = ee.Number(monthIndex).add(1);
    var monthStr = ee.String(monthNames.get(monthIndex));

    // Filter the entire feature collection for the specific
month
    var monthly_features =
climate_features.filter(ee.Filter.eq('month', monthNum));

    // Construct the title server-side
    var title = ee.String('Mean ').cat(variableName).cat('
Trend for ').cat(monthStr).cat(' (1990 - 2030)');

    // Create the chart.
    var chart = ui.Chart.feature.byFeature({
        features: monthly_features,
        xProperty: 'year',
        yProperties: [variableName]
      })
      .setChartType('ScatterChart')
      .setOptions({
        title: title,
        hAxis: {
          title: 'Year',
          gridlines: {color: '#cccccc', count: 10},
          format: '####',
          viewWindow: {
            min: startYear,
            max: endYear
          }
        },
        vAxis: {
          title: yAxisTitle,
          gridlines: {color: '#cccccc', count: 5}
        },
        series: {0: {color: color, pointSize: 4}},
        legend: {position: 'none'},
        trendlines: {0: {
          type: 'linear', color: trendColor, lineWidth: 3,
opacity: 0.7,
          visibleInLegend: true, labelInLegend: 'Linear Trend'
        }}
      });
```

```
    return chart; // Return the chart object
};

// Function to generate and print all 24 charts
var generateAndPrintAllCharts = function() {

    // --- Temperature Charts (12 charts) ---
    months.getInfo().forEach(function(month) {
        var monthIndex = month - 1;
        var monthName = monthNames.getInfo()[monthIndex];

        var temp_chart = generateChart(
            ee.Number(monthIndex),
            'mean_temp_c',
            'Mean Temperature (°C)',
            'red',
            'darkred'
        );

        print('🌡️ Temperature: ' + monthName + ' Chart:',
temp_chart);
    });

    // --- Absolute Humidity Charts (12 charts) ---
    months.getInfo().forEach(function(month) {
        var monthIndex = month - 1;
        var monthName = monthNames.getInfo()[monthIndex];

        var ah_chart = generateChart(
            ee.Number(monthIndex),
            'mean_ah_g_m3', // Changed variable name
            'Mean Absolute Humidity (g/m³)', // Changed Y-axis
title
            'purple', // Changed color for distinction
            'darkviolet'
        );

        print('💧 Absolute Humidity: ' + monthName + '
Chart:', ah_chart); // Changed label
    });
};

// Execute the chart generation and printing
generateAndPrintAllCharts();

// ------------------------------------------------------------
----------------------------------------

// 7. Determine and print the last year/month with valid data
var                        valid_dates                        =
valid_features.aggregate_array('label').sort();

var last_valid_label = ee.Algorithms.If(
  valid_dates.size().gt(0),
  valid_dates.get(valid_dates.size().subtract(1)),
  'No Valid Data Found'
);
print('🔍 Last Valid Data Point Found (Year-Month):',
last_valid_label);
```

```
// 8. Optionally, center the map on Kolkata and display the
AOI.
Map.centerObject(kolkata_aoi, 9);
Map.addLayer(kolkata_aoi, {color: '00FF00', opacity: 0.4},
'Kolkata AOI Polygon');

// 9. EXPORT DATA TO GOOGLE DRIVE (Updated selectors)

Export.table.toDrive({
  collection: valid_features,
  description: 'Kolkata_Monthly_Climate_Export_AH',
  fileNamePrefix: 'kolkata_monthly_absolute_humidity_data',
  folder: 'GEE_Exports',
  fileFormat: 'CSV',
  selectors:  ['year',  'month',  'label',  'mean_temp_c',
'mean_ah_g_m3'] // Selectors updated
});

print('✅ Data Export Task Queued. Check the "Tasks" tab (top right)
to Run it.');
```

4. Data: Land Surface Temperature Map and Data and Absolute Humidity (1990 – 2023)

Output file name: LST.csv

Code:

```
// --- 1. Define Area of Interest (AOI) and Time Periods ---

// Coordinates for Kolkata (West Bengal, India)
var aoi = ee.Geometry.Point(88.36, 22.57).buffer(20000); //
20km buffer around the center

var date_start_early = '1990-01-01';
var date_end_early = '2000-12-31';
var date_start_recent = '2015-01-01';
var date_end_recent = '2025-01-01';

// --- 2. Load and Prepare Landsat Data ---
// Correction: Using the current, stable Landsat Collection 2,
Level 2 assets.
var                    L5_COLLECTION                    =
ee.ImageCollection('LANDSAT/LT05/C02/T1_L2');
var                    L8_COLLECTION                    =
ee.ImageCollection('LANDSAT/LC08/C02/T1_L2');

// LST Scaling Constants (same for L5/L8 C2 L2 Surface
Temperature product)
var ST_SCALE_FACTOR = 0.00341802;
var ST_OFFSET = 149.0;

// Function to process Landsat 5 LST (uses ST_B6)
var processL5_LST = function(image) {
  // Select the correct Landsat 5 LST band (ST_B6) and apply
scaling.
  // The resulting LST is in Kelvin. Convert to Celsius (Kelvin
- 273.15).
  var LST_Celsius = image.select('ST_B6')
    .multiply(ST_SCALE_FACTOR)
    .add(ST_OFFSET)
```

```
      .subtract(273.15)
      .rename('LST');

  // Add metadata bands for cloud removal and display
  return image.addBands(LST_Celsius)
              .select('LST')
              .copyProperties(image,    ['system:time_start',
'CLOUD_COVER']);
};

// Function to process Landsat 8 LST (uses ST_B10)
var processL8_LST = function(image) {
  // Select the correct Landsat 8 LST band (ST_B10) and apply
scaling.
  // The resulting LST is in Kelvin. Convert to Celsius (Kelvin
- 273.15).
  var LST_Celsius = image.select('ST_B10')
    .multiply(ST_SCALE_FACTOR)
    .add(ST_OFFSET)
    .subtract(273.15)
    .rename('LST');

  // Add metadata bands for cloud removal and display
  return image.addBands(LST_Celsius)
              .select('LST')
              .copyProperties(image,    ['system:time_start',
'CLOUD_COVER']);
};

// --- 3. Filter and Map Collections ---

// Landsat 5 data for the early period (L5 stops in 2013)
var L5_Early = L5_COLLECTION
  .filterBounds(aoi)
  .filterDate(date_start_early, date_end_early)
  .filterMetadata('CLOUD_COVER', 'less_than', 20)
  .map(processL5_LST); // Mapped to L5 specific function

// Landsat 8 data for the recent period (L8 starts in 2013)
var L8_Recent = L8_COLLECTION
  .filterBounds(aoi)
  .filterDate(date_start_recent, date_end_recent)
  .filterMetadata('CLOUD_COVER', 'less_than', 20)
  .map(processL8_LST); // Mapped to L8 specific function

// Merge the collections for the time series chart
var LST_Merged = L5_Early.merge(L8_Recent);


// --- 4. Initialization and Robustness Check ---

// Use getInfo() to force server-side execution and check if
data exists in both periods.
var L5_size = L5_Early.size().getInfo();
var L8_size = L8_Recent.size().getInfo();

Map.centerObject(aoi, 9); // Center the map regardless of data
availability

if (L5_size > 0 && L8_size > 0) {
```

```
  // --- 4.1 Calculate Mean LST and Change Map (Executed only
if data exists) ---

  // 1. LST (1990-2000) - Early Period
  var meanLST_Early = L5_Early.mean().clip(aoi);

  // 2. LST (2015-2025) - Recent Period
  var meanLST_Recent = L8_Recent.mean().clip(aoi);

  // 3. LST Change (Recent - Early)
  var                        lstChange                        =
meanLST_Recent.subtract(meanLST_Early).rename('LST_Change');

  // --- 5. Visualization Parameters ---

  // LST Palette (Cool to Hot: Blue to Red for Celsius)
  var lstVis = {
    min: 15,
    max: 35,
    palette: [
      '0000FF', // Blue (Cool)
      '00FFFF', // Cyan
      '00FF00', // Green
      'FFFF00', // Yellow
      'FF0000'  // Red (Hot)
    ]
  };

  // Change Palette (Cooling to Warming: Blue to Red, centered
at 0)
  var changeVis = {
    min: -3,
    max: 3,
    palette: [
      '0000FF', // Blue (Cooling)
      'FFFFFF', // White (No change)
      'FF0000'  // Red (Warming)
    ]
  };

  // --- 6. Display Maps in Earth Engine Viewer ---

  Map.addLayer(meanLST_Early, lstVis, '1. LST (1990-2000) -
Early Period (C)');
  Map.addLayer(meanLST_Recent, lstVis, '2. LST (2015-2025) -
Recent Period (C)');
  Map.addLayer(lstChange, changeVis, '3. LST Change (Recent -
Early) (C)');

  // --- Add Legends to the Map ---
  // Create a panel for the legend.
  var legend = ui.Panel({
    style: {
      position: 'bottom-left',
      padding: '8px 15px'
    }
  });

  // Create a title for the legend.
  var legendTitle = ui.Label({
    value: 'LST Change (°C) 1990-2000 to 2015-2025',
```

```
      style: {
        fontWeight: 'bold',
        fontSize: '14px',
        margin: '0 0 4px 0',
        padding: '0'
      }
    });
    legend.add(legendTitle);

    // Create a color bar for the legend.
    var makeColorBarParams = function(palette) {
      return {
        bbox: [0, 0, 1, 0.1],
        dimensions: '100x10',
        format: 'png',
        min: 0,
        max: 1,
        palette: palette,
      };
    };

    var colorBar = ui.Thumbnail({
      image: ee.Image.pixelLonLat().select(0),
      params: makeColorBarParams(changeVis.palette),
      style: {stretch: 'horizontal', margin: '0px 8px',
maxHeight: '24px'},
    });
    legend.add(colorBar);

    // Create text labels for the legend.
    var legendLabels = ui.Panel({
      widgets: [
        ui.Label(changeVis.min, {margin: '4px 8px'}),
        ui.Label(changeVis.max / 2, {margin: '4px 8px',
textAlign: 'center', stretch: 'horizontal'}),
        ui.Label(changeVis.max, {margin: '4px 8px'})
      ],
      layout: ui.Panel.Layout.flow('horizontal')
    });
    legend.add(legendLabels);

    // Add source information
    var sourceLabel = ui.Label({
      value: 'Source: Landsat 5 & 8, USGS. Processed in GEE.',
      style: {
        fontSize: '10px',
        margin: '4px 0 0 0',
        padding: '0'
      }
    });
    legend.add(sourceLabel);

    // Add the legend to the map.
    Map.add(legend);


    // --- 7. Generate Time Series Chart and Exportable Table
Data ---

    // Chart configuration
    var chartOptions = {
```

```
    title: 'Kolkata Land Surface Temperature (LST) Time Series
(1990-2025)\nSource: Landsat 5 & 8, USGS. Processed in GEE.',
    vAxis: {title: 'LST (°C)'},
    hAxis: {title: 'Date'},
    lineWidth: 1,
    pointSize: 3,
    trendlines: {
      0: {
        type: 'linear',
        color: 'red',
        visibleInLegend: true,
        labelInLegend: 'Long-term Trend',
      }
    }
  };

  // Create the chart using the merged LST collection
  var LSTChart = ui.Chart.image.series(
      LST_Merged,
      aoi,
      ee.Reducer.mean(),
      30 // Scale (resolution) in meters
    )
    .setOptions(chartOptions)
    .setChartType('ScatterChart');

  // Print the chart to the console
  print(LSTChart);
  print('LST layers successfully generated for the AOI. Check
the Map and Console for results.');

  // --- Generate Feature Collection for CSV Export (FIX for
Export.chart.toDrive error) ---
  var exportFeatures = LST_Merged.map(function(image) {
    var stats = image.reduceRegion({
      reducer: ee.Reducer.mean(),
      geometry: aoi,
      scale: 30,
      maxPixels: 1e9
    });

    // Get the date string.
    var                          date                          =
ee.Date(image.get('system:time_start')).format('YYYY-MM-dd');

    // Return a feature containing the date and the LST mean.
    return ee.Feature(null, {
      'Date': date,
      'LST_Celsius': stats.get('LST')
    });
  }).filter(ee.Filter.notNull(['LST_Celsius'])); // Filter out
features where LST could not be calculated (e.g., all cloudy)


  // --- 8. Export Data and Images (Defensive Block) ---

  // 8.1 Export the LST Change Image to Google Drive (GeoTIFF
only - PNG not supported)
  try {
    // Export 1: High Quality GeoTIFF (Preserves original data)
    Export.image.toDrive({
```

```
        image: lstChange,
        description: 'Kolkata_LST_Change_Map_GeoTIFF',
        folder: 'GEE_Exports',
        fileNamePrefix: 'Kolkata_LST_Change_Map_GeoTIFF',
        region: aoi.bounds(),
        scale: 30, // Landsat resolution
        crs: 'EPSG:4326',
        fileFormat: 'GeoTIFF',
        maxPixels: 1e13
      });
      print('Image Map GeoTIFF export task initialized.');
    } catch (e) {
      print('--- IMAGE EXPORT FAILURE ---');
      print('Image export could not be initialized due to an
invalid Image object. Error details: ' + e);
    }

    // 8.2 Export the Time Series Chart Data to Google Drive
(CSV) (FIX: Use Export.table)
    try {
      Export.table.toDrive({
        collection: exportFeatures,
        description: 'Kolkata_LST_TimeSeries_Data_CSV',
        folder: 'GEE_Exports',
        fileNamePrefix: 'Kolkata_LST_TimeSeries_Data',
        fileFormat: 'CSV'
      });
      print('Chart Data CSV export task initialized via
Export.table.');

    } catch (e) {
      print('--- CHART DATA EXPORT FAILURE ---');
      print('Chart data export could not be initialized. Error
details: ' + e);
    }

    // 8.3 Generate a direct PNG download URL for visualization
(Workaround for GEE JS API limitation)
    try {
      // FIX: Removed 'scale' as it conflicts with 'dimensions'
in getThumbURL.
      var thumbParams = {
          'min': changeVis.min,
          'max': changeVis.max,
          'palette': changeVis.palette,
          'dimensions': 1024, // High resolution for download
          'region': aoi.bounds(),
          'crs': 'EPSG:4326'
      };

      var thumbUrl = lstChange.getThumbURL(thumbParams);

      print('--- VISUAL MAP PNG DOWNLOAD URL ---');
      print('Use this URL to download the LST Change Map (Visual
PNG, 1024px) directly: ' + thumbUrl);

    } catch (e) {
      print('--- PNG URL GENERATION FAILURE ---');
      print('Could not generate the direct PNG download URL.
Error details: ' + e);
    }
```

```
  // Add a final note about the exports
  print('You must click "Run" in the Earth Engine "Tasks" tab
to start the file transfers to Google Drive.');
  print('The chart data is now exported as a CSV table called
"Kolkata_LST_TimeSeries_Data" in the GEE_Exports folder.');


} else {
  print('--- ERROR ---');
  print('Image collection error: Cannot calculate LST change
because one or both required collections are empty.');
  print('L5 (1990-2000) images found: ' + L5_size);
  print('L8 (2015-2025) images found: ' + L8_size);
  print('Please check the date ranges or relax the cloud cover
filter (currently set to less than 20%).');
}
```

5. Data: Diurnal Temperature Range (1990 – 2023)

Output file name: Monthly DTR.csv

Code:

```
/**
 * MEAN DIURNAL TEMPERATURE RANGE (DTR) ANALYSIS (1990 -
Present) for ALL MONTHS (Jan-Dec)
 * FINAL FIX: Restructured to use SERVER-SIDE REDUCTION
(ee.Reducer.group) for DTR
 * calculation across all years, eliminating client-side
memory/rate limit issues
 * that caused the "Dictionary does not contain key" error.
 * * ENHANCEMENT: Added a process to combine all monthly DTR
data and export it to a CSV.
 */

// 1. Define the Area of Interest (AOI) for Kolkata, India.
var kolkata_aoi = ee.Geometry.BBox(88.25, 22.45, 88.50,
22.70);

// 2. Define the Time Period (1990 to present).
var startYear = 1990;
var currentYear = ee.Date(Date.now()).get('year');

// Define the months to analyze (January = 1, ..., December =
12)
var monthsToAnalyze = ee.List.sequence(1, 12).getInfo();

// Define a client-side list of month names for the chart
titles
var monthNames = ['January', 'February', 'March', 'April',
'May', 'June',
                  'July', 'August', 'September', 'October',
'November', 'December'];


// 3. Load the ERA5-LAND HOURLY dataset once and filter by year
range.
var hourlyData = ee.ImageCollection("ECMWF/ERA5_LAND/HOURLY")
    .filterBounds(kolkata_aoi)
    .filterDate(ee.Date.fromYMD(startYear, 1, 1),
ee.Date.fromYMD(currentYear.add(1), 1, 1))
    .select('temperature_2m'); // Temperature is in Kelvin (K)
```

```
// --- 4. Core Server-Side DTR Calculation ---

var calculateMonthlyDTR = function(month) {
    month = ee.Number(month);

    // a. Filter data for the target month across all years
    var monthlyData =
hourlyData.filter(ee.Filter.calendarRange(month, month,
'month'));

    // b. Calculate daily DTR (Max - Min)
    var dailyDTR = monthlyData.map(function(image) {
        // Tag the image with its date's year and day-of-year
for grouping
        var date = ee.Date(image.get('system:time_start'));
        var year = date.get('year');
        var day = date.getRelative('day', 'year');

        // Get the start and end of the current day for
filtering
        var dayRange = date.getRange('day');

        // This is the core DTR calculation, now performed
efficiently on the server.
        // NOTE: The filterDate here is inefficient as it re-
filters the whole collection
        // a better approach would be to group by day and then
reduce.
        // For now, we will keep the original logic for min/max
calculation,
        // but it is important to note this is a very heavy-
lift on the server:
        var maxK = monthlyData.filterDate(dayRange).max();
        var minK = monthlyData.filterDate(dayRange).min();
        var dtrImage = maxK.subtract(minK).rename('DTR_C');

        // Calculate the mean DTR over the AOI and attach
properties
        var meanDTR = dtrImage.reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: kolkata_aoi,
            scale: 9000,
            bestEffort: true
        });

        // Return a feature with the DTR, year, and day for
later grouping/filtering.
        return ee.Feature(null, {
            'DTR_C': ee.Number(meanDTR.get('DTR_C', 0)), //
Use 0 if DTR fails
            'year': year,
            'day': day
        });
    });

    // c. Group by Year and reduce to get the Mean Monthly DTR
    var annualMeanDTR = dailyDTR
        .filter(ee.Filter.gt('DTR_C', 0)) // Filter out any
failed calculations
```

```
            .reduceColumns({
                reducer: ee.Reducer.mean().group({
                    groupField: 1, // Group by the 'year' property
                    groupName: 'year',
                }),
                selectors: ['DTR_C', 'year']
            });

        // d. Reformat the result into a clean FeatureCollection
        var                          results                          =
    ee.List(annualMeanDTR.get('groups')).map(function(group) {
            group = ee.Dictionary(group);
            var year = group.get('year');
            var mean_dtr = ee.Number(group.get('mean')).round();
            var                       monthName                       =
    ee.String(ee.List(monthNames).get(month.subtract(1)));

            return ee.Feature(null, {
                'year': year,
                'month_num': month, // Added for filtering/sorting
    in CSV
                'month_name': monthName, // Added for context in
    CSV
                'date_str':      ee.Date.fromYMD(year,      month,
    1).format('YYYY'),
                'Observed DTR': mean_dtr
            });
        });

        return ee.FeatureCollection(results);
    };


    // --- 5. Execution Loop (Client-Side) and Collection Storage
    ---
    //  Initialize  an  empty  client-side  list  to  hold  the
    FeatureCollections for each month
    var allMonthlyDTRCollections = [];

    var analyzeMonth = function(month) {
        month = ee.Number(month);
        var monthName = monthNames[month.getInfo() - 1];

        // Get the calculated FeatureCollection from the server.
        var annualDTR_features = calculateMonthlyDTR(month);

        //  *********  ENHANCEMENT:  Store  the  results  for  later
    merging *********
        allMonthlyDTRCollections.push(annualDTR_features);
        //
    ************************************************************
    ********

        //  a.  Calculate  Simple  Linear  Trend  (Regression)  for
    coefficient printing.
        var linearFit = annualDTR_features.reduceColumns({
            reducer: ee.Reducer.linearFit(),
            selectors: ['year', 'Observed DTR']
        });

        var slope = ee.Number(linearFit.get('scale'));
```

```
        var intercept = ee.Number(linearFit.get('offset'));


        // b. Generate the Chart (using built-in trendline)
        var dtrChart = ui.Chart.feature.byFeature({
            features: annualDTR_features, // Chart only the
observed data
            xProperty: 'date_str', // Use the string format for
better chart display
            yProperties: ['Observed DTR']
        })
        .setChartType('ScatterChart')
        .setOptions({
            title: 'Mean ' + monthName + ' DTR Trend (Linear Fit)
for Kolkata (1990 - ' + currentYear.getInfo() + ')',
            hAxis: {title: 'Year', showTextEvery: 5},
            vAxis: {title: 'Mean ' + monthName + ' DTR (°C)',
minValue: 0, gridlines: {color: '#cccccc'}},
            pointSize: 4,
            trendlines: {
                0: {
                    type: 'linear',
                    color: '#FF4500',
                    lineWidth: 3,
                    visibleInLegend: true,
                    labelInLegend: 'Linear Trend'
                }
            },
            series: {
                0: {color: '#008080'}
            },
            explorer: {}
        });

        // c. Output the results.
        print('--- 📈 Mean ' + monthName + ' DTR Trend Analysis
(Linear Fit) ---');
        print('Annual Mean DTR Chart:', dtrChart);
        print('Regression Coefficients:',
            ee.Dictionary({
                'Month': monthName,
                'Slope (m) - Change per year':
slope.format('%.4f'),
                'Y-Intercept (b)': intercept.format('%.3f')
            })
        );
    };


    // --- 6. Final Execution and Data Export ---

    print('*** Running Multi-Month DTR Analysis for Kolkata (1990
    - ' + currentYear.getInfo() + ') ***');

    // Run the analysis for all months from January (1) to December
    (12).
    monthsToAnalyze.forEach(function(month) {
        analyzeMonth(ee.Number(month));
    });

    // ********* ENHANCEMENT: Merge and Export Data *********
```

```
// 1. Merge all monthly FeatureCollections into a single
collection.
//   ee.FeatureCollection(list)   automatically   merges   the
collections in the list.
var                    allDataCollection                    =
ee.FeatureCollection(allMonthlyDTRCollections).flatten();

// 2. Define the export task to Google Drive.
Export.table.toDrive({
    collection: allDataCollection,
    description: 'Kolkata_Monthly_DTR_1990_Present',
    folder:  'EarthEngine_Exports',  // Change this to your
desired Drive folder
    fileNamePrefix: 'Kolkata_DTR_Data',
    fileFormat: 'CSV',
    selectors: ['year', 'month_num', 'month_name', 'Observed
DTR', 'date_str'] // Select columns for CSV
});

print('✅ Data Export Task Created:',
      'A task to export all DTR data to a CSV file in your
Google Drive has been created.',
      'Check the "Tasks" tab (right side of the GEE code
editor) to run the export.'
);
// ****************************************************

// Final map output
Map.centerObject(kolkata_aoi, 9);
Map.addLayer(kolkata_aoi, {color: '00FF00', opacity: 0.4}, 'Kolkata
AOI Polygon');
```

6. Data: Annual Diurnal Temperature Range (1990 – 2023)

Output file name: None (graph)

Code:

```
// 1. Define the Region of Interest (ROI) - Kolkata and a 30km
buffer
var   kolkata   =   ee.Geometry.Point(88.3639,   22.5726);   //
Approximate center of Kolkata
var roi = kolkata.buffer(30000); // 30 km buffer

// 2. Filter ERA5-Land Daily data (2m temperature)
var era5 = ee.ImageCollection('ECMWF/ERA5_LAND/DAILY_AGGR')
    .filterDate('1990-01-01', ee.Date(Date.now()))
    .filterBounds(roi);

// Function to calculate Diurnal Temperature Range (DTR) in
Celsius
var calculateDTR = function(image) {
  // Select the available maximum and minimum temperature bands
  // Tmax and Tmin are in Kelvin. Convert to Celsius and
calculate DTR.
  var                          tmaxC                          =
image.select('temperature_2m_max').subtract(273.15);
  var                          tminC                          =
image.select('temperature_2m_min').subtract(273.15);

  var dtr = tmaxC.subtract(tminC).rename('DTR');
```

```
  // Copy properties and return the DTR band
  return dtr.copyProperties(image, ['system:time_start']);
};

// Map the function over the image collection to get the daily
DTR
var dtrCollection = era5.map(calculateDTR);

// 3. Calculate Annual Mean DTR
var currentYear = ee.Date(Date.now()).get('year');
var years = ee.List.sequence(1990, currentYear.subtract(1));
// Exclude current incomplete year

// Use ee.FeatureCollection()
var annualDTR = ee.FeatureCollection(years.map(function(year)
{
  var start = ee.Date.fromYMD(year, 1, 1);
  var end = start.advance(1, 'year');
  var yearlyDTR = dtrCollection.filterDate(start, end).mean();

  // Calculate the mean DTR over the ROI for the year
  var meanDTR = yearlyDTR.reduceRegion({
    reducer: ee.Reducer.mean(),
    geometry: roi,
    scale: 10000,
    maxPixels: 1e9
  }).get('DTR');

  // Return a feature for charting
  return ee.Feature(null, {
    'year': year,
    'DTR': meanDTR,
    'system:time_start': start.millis()
  });
})).filter(ee.Filter.notNull(['DTR']));

// 4. Create and Display Time Series Chart
var chart = ui.Chart.feature.byFeature({
    features: annualDTR,
    xProperty: 'year',
    yProperties: ['DTR']
  })
  .setChartType('ScatterChart')
  .setOptions({
    title: 'Annual Mean Diurnal Temperature Range (DTR) for
Kolkata (30km Buffer)',
    hAxis: {title: 'Year', format: '####'}, // <<< FIX APPLIED
HERE
    vAxis: {title: 'Mean DTR (°C)'},
    trendlines: {0: {color: 'red', visibleInLegend: true,
type: 'linear', showR2: true}},
    lineWidth: 1,
    pointSize: 3,
    explorer: {}
  });

// Print the chart to the Console
print(chart);

// Center map and add ROI boundary
Map.centerObject(kolkata, 9);
```

```
Map.addLayer(roi, {color: 'FF0000'}, 'Kolkata 30km Buffer');
```

7. Data: UHI Comparison (2000/2001 and 2022/2023)

Output file name: None (graph)

Code:

```
// --- 1. FIXED PARAMETERS (GEOGRAPHY & MASKS) ---

// City Location (Kolkata, India)
var KOLKATA_LAT = 22.57;
var KOLKATA_LON = 88.36;

// Analysis Radii
var BUFFER_RADIUS_KM = 30; // Total area of analysis
var URBAN_CORE_RADIUS_KM = 5; // Fixed 5 km circle for Urban
Sample

// Define the region of interest (ROI)
var point = ee.Geometry.Point(KOLKATA_LON, KOLKATA_LAT);
var roi = point.buffer(BUFFER_RADIUS_KM * 1000);
var  urban_core_geom  =  point.buffer(URBAN_CORE_RADIUS_KM  *
1000);

// Load the Global Human Settlement Layer (GHSL) Built-up Grid
(P2016)
var                          built_up                          =
ee.Image('JRC/GHSL/P2016/BUILT_LDSMT_GLOBE_V1').select('built
');

// Define Rural areas (< 5% built-up density)
var rural_mask = built_up.lt(5).rename('Rural_Mask');

// Set the map center and zoom level
Map.centerObject(roi, 10);

// --- 2. LST PROCESSING FUNCTION ---

// Function to process MODIS LST, convert to Celsius, and apply
QC mask.
var processLST = function(image) {
  var scaleFactor = 0.02;

  // Acceptable Quality (QC flags 0 or 1) for both Day and
Night LST.
  var                 acceptableQualityMask                 =
image.select('QC_Day').lte(1).or(image.select('QC_Night').lte
(1));

  // Convert Day LST to Celsius
  var dayLST = image.select('LST_Day_1km')
      .multiply(scaleFactor)
      .subtract(273.15)
      .updateMask(acceptableQualityMask)
      .rename('LST_Day_C');

  // Convert Night LST to Celsius
  var nightLST = image.select('LST_Night_1km')
      .multiply(scaleFactor)
      .subtract(273.15)
      .updateMask(acceptableQualityMask)
      .rename('LST_Night_C');
```

```
  return image.addBands(dayLST).addBands(nightLST).clip(roi);
};


// --- 3. UHI CALCULATION FUNCTION ---

/**
 * Calculates and prints the Urban Heat Island Intensity (UHII)
for a given date range.
 * @param {string} startDate - Start date (YYYY-MM-DD).
 * @param {string} endDate - End date (YYYY-MM-DD).
 * @param {string} label - Label for the console output.
 * @returns {ee.Image} The mean LST image for the period.
 */
var calculateUHI = function(startDate, endDate, label) {
  print('================================================');
  print('           ANALYSIS PERIOD: ' + label);
  print('================================================');

  // 1. Load and Process LST Collection
  var modisLST = ee.ImageCollection('MODIS/061/MOD11A2')
      .filterDate(startDate, endDate)
      .filterBounds(roi)
      .map(processLST);

  // 2. Calculate Mean LST Composite
  var       meanLST       =       modisLST.select('LST_Day_C',
'LST_Night_C').mean();

  // 3. Extract Urban Core LST (5km fixed geometry)
  var urban_stats = meanLST.reduceRegion({
      reducer: ee.Reducer.mean(),
      geometry: urban_core_geom,
      scale: 1000,
      maxPixels: 1e9
  });

  // 4. Extract Rural Area LST (Low-density mask)
  var rural_lst = meanLST.updateMask(rural_mask);
  var rural_stats = rural_lst.reduceRegion({
      reducer: ee.Reducer.mean(),
      geometry: roi,
      scale: 1000,
      maxPixels: 1e9
  });

  // 5. Calculate UHII (Urban - Rural)
  var uhii_day = ee.Number(urban_stats.get('LST_Day_C'))
      .subtract(ee.Number(rural_stats.get('LST_Day_C')));

  var uhii_night = ee.Number(urban_stats.get('LST_Night_C'))
      .subtract(ee.Number(rural_stats.get('LST_Night_C')));

  // 6. Print Results
  print('URBAN LST (' + label + '):', urban_stats);
  print('RURAL LST (' + label + '):', rural_stats);

  print('--- URBAN HEAT ISLAND INTENSITY (UHII) ---');
  print('Daytime      UHII      (Urban      -      Rural):',
uhii_day.format('%.2f'), '°C');
```

```
  print('Nighttime UHII (Urban - Rural):',
uhii_night.format('%.2f'), '°C');

  return meanLST;
};


// --- 4. EXECUTION ---

// Run Historical Analysis (2000-2001)
var lst_2000 = calculateUHI('2000-01-01', '2002-01-01', '2000-
2001 BASELINE');

// Run Recent Analysis (2022-2023)
var lst_2022 = calculateUHI('2022-01-01', '2024-01-01', '2022-
2023 RECENT');


// --- 5. MAP VISUALIZATION (using 2022-2023 data) ---

var lstVis = {
    min: 20, // Min Temp (C)
    max: 40, // Max Temp (C)
    palette: ['blue', 'cyan', 'green', 'yellow', 'red']
};

var diffVis = {
    min: 0,
    max: 10, // Max Diurnal Range (C)
    palette: ['white', 'yellow', 'orange', 'red']
};

// Add geographic layers
Map.addLayer(urban_core_geom, {color: 'orange'}, '1. Urban
Core (5km Radius)');
Map.addLayer(rural_mask.updateMask(rural_mask), {palette:
['green']}, '2. Rural Background (<5% Density)');

// Add LST visualization (using the recent 2022 data)
Map.addLayer(lst_2022.select('LST_Day_C'), lstVis, '3. Mean
Day LST (2022-2023)');
Map.addLayer(lst_2022.select('LST_Night_C'), lstVis, '4. Mean
Night LST (2022-2023)');

print('\nMap Layers: The map shows the Land Surface Temperature
for the 2022-2023 period.');
print('Results Comparison: Check the console for the side-by-side
UHII comparison between 2000-2001 and 2022-2023.');
```

8. Data: Rainfall Summary (1990 – 2023)

Output file name: Annual Rainfall Summary and Monthly Rainfall Summary

Code:

```
/**
 * Google Earth Engine Script to Analyze Rainfall
(Precipitation)
 * Trends in Kolkata, India, from 1990 to Present (2024).
 * This script focuses on three key metrics: Annual Total,
Annual Mean Daily,
 * and Annual Monthly Mean Daily precipitation.
```

```
 *
 * This script uses the CHIRPS Daily: Climate Hazards Group
InfraRed Precipitation
 * with Station data (UCSB-CHG/CHIRPS/DAILY).
 *
 * Author: Gemini
 * Date: November 2025
 */

// --- 1. Define Area of Interest (AOI) and Time Periods ---

// Coordinates for Kolkata (West Bengal, India)
var aoi = ee.Geometry.Point(88.36, 22.57).buffer(30000);

var START_DATE = '1990-01-01';
var END_DATE = '2025-01-01'; // Up to the present

Map.centerObject(aoi, 8); // Center the map
Map.addLayer(aoi, {color: '000000'}, 'AOI: Kolkata Region');

// Define a list of years and months for iteration
var years = ee.List.sequence(1990, 2024);
var months = ee.List.sequence(1, 12);
var seriesNames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];

// --- 2. Load and Filter CHIRPS Daily Data ---

var chirps = ee.ImageCollection('UCSB-CHG/CHIRPS/DAILY')
    .filterDate(START_DATE, END_DATE)
    .filterBounds(aoi);

print('Total daily CHIRPS images found:', chirps.size());


// --- 3. ANNUAL TOTAL AND MEAN DAILY ANALYSIS (Request 1 & 3)
---

// Function to calculate annual statistics (mean daily and
total precipitation)
var calculateAnnualStats = function(year) {
  var start = ee.Date.fromYMD(year, 1, 1);
  var end = start.advance(1, 'year');

  // Filter for the current year
  var annualCollection = chirps.filterDate(start, end);

  // 1. Calculate Annual Total Precipitation (sum of daily
values)
  var annualTotal = annualCollection.sum().reduceRegion({
    reducer: ee.Reducer.mean(),
    geometry: aoi,
    scale: 5566, // CHIRPS resolution (~5.5 km)
    maxPixels: 1e9
  }).get('precipitation');

  // 2. Calculate Annual Mean Daily Precipitation (mean of
daily values)
  var annualMeanDaily = annualCollection.mean().reduceRegion({
    reducer: ee.Reducer.mean(),
    geometry: aoi,
```

```
    scale: 5566,
    maxPixels: 1e9
  }).get('precipitation');

  // Return a Feature for charting
  return ee.Feature(null, {
    'Year': year,
    'Annual_Total_Rainfall_mm': annualTotal,
    'Annual_Mean_Daily_Rainfall_mm': annualMeanDaily,
    'system:time_start': start.millis() // Required for time
series charts
  });
};

// Map the function over the list of years
var                      annualStatsFC                      =
ee.FeatureCollection(years.map(calculateAnnualStats));

// --- 3.1. Chart: Annual Total Rainfall (Request 1) ---
var annualTotalChart = ui.Chart.feature.byFeature({
    features: annualStatsFC,
    xProperty: 'system:time_start',
    yProperties: ['Annual_Total_Rainfall_mm']
  })
  .setChartType('ColumnChart')
  .setOptions({
    title: '1. Kolkata Annual Total Rainfall (1990-2024)',
    hAxis: {title: 'Year', format: 'yyyy'},
    vAxis: {title: 'Total Rainfall (mm)'},
    colors: ['0000FF']
  });
print('Chart 1: Annual Total Rainfall:', annualTotalChart);

// --- 3.2. Chart: Annual Mean Daily Rainfall (Request 3) ---
var annualMeanChart = ui.Chart.feature.byFeature({
    features: annualStatsFC,
    xProperty: 'system:time_start',
    yProperties: ['Annual_Mean_Daily_Rainfall_mm']
  })
  .setChartType('LineChart')
  .setOptions({
    title: '3. Kolkata Annual Mean Daily Rainfall (1990-2024)',
    hAxis: {title: 'Year', format: 'yyyy'},
    vAxis: {title: 'Mean Daily Rainfall (mm/day)'},
    colors: ['00A000'],
    trendlines: { 0: { color: 'red' } }
  });
print('Chart    2:    Annual    Mean    Daily    Rainfall:',
annualMeanChart);


// --- 4. ANNUAL MONTHLY MEAN ANALYSIS (Request 2) (FIXED FOR
CHARTING) ---

/**
 * Function to calculate mean daily precipitation for a given
month/year and format it.
 * This function returns a flattened structure (Year, Month,
Value) suitable for ui.Chart.feature.groups.
 * @param {ee.Number} year The year to calculate statistics
for.
```

```
 * @return {ee.FeatureCollection} A feature collection of 12
features (one for each month).
 */
var createMonthlyTimeSeriesFC = function(year) {
  // Use a server-side map to generate 12 features for the 12
months of the year
  return ee.FeatureCollection(months.map(function(month) {
    month = ee.Number(month);
    var start = ee.Date.fromYMD(year, month, 1);
    var end = start.advance(1, 'month');

    // Filter collection for the current month within the year
    var monthlyCollection = chirps.filterDate(start, end);

    // Calculate Mean Daily Precipitation for this month/year
    var                   meanDailyPrecip                   =
monthlyCollection.mean().reduceRegion({
      reducer: ee.Reducer.mean(),
      geometry: aoi,
      scale: 5566,
      maxPixels: 1e9
    }).get('precipitation');

    // Return a Feature for charting, using Month_Name as the
series identifier
    return ee.Feature(null, {
      'Year': year,
      'Month': month,
      'Month_Name':
ee.List(seriesNames).get(month.subtract(1)), // Get Jan, Feb,
etc.
      'Mean_Daily_Rainfall_mm': meanDailyPrecip,
      'system:time_start': start.millis() // X-axis property
(used by the chart)
    });
  }));
};

// Map the function over the list of years and flatten the
result (e.g., 35 years * 12 months = 420 features)
var                   monthlyTimeSeriesFC                   =
ee.FeatureCollection(years.map(createMonthlyTimeSeriesFC)).fl
atten();

// --- 4.1. Chart: Annual Monthly Mean Daily Rainfall Time
Series (Request 2) ---
var annualMonthlyMeanChart = ui.Chart.feature.groups({
    features: monthlyTimeSeriesFC,
    xProperty: 'system:time_start',
    yProperty: 'Mean_Daily_Rainfall_mm',
    seriesProperty: 'Month_Name' // Group the data by month to
create 12 distinct lines
  })
  .setChartType('LineChart')
  .setOptions({
    title: '2. Kolkata Monthly Mean Daily Rainfall Trends
(1990-2024)',
    hAxis: {title: 'Year', format: 'yyyy'},
    vAxis: {title: 'Mean Daily Rainfall (mm/day)'},
    series: {
      0: {color: 'blue'}, // Jan
```

```
            5: {color: 'green', lineWidth: 3}, // Jun (Highlighting
      Monsoon start)
            6: {color: 'red', lineWidth: 3} // July (Highlighting
      Monsoon peak)
          },
          legend: {position: 'right'}
        });
      print('Chart 3: Annual Monthly Mean Daily Rainfall Trends:',
      annualMonthlyMeanChart);


      // --- 5. Export Annual Data (CSV) ---
      // Two separate exports for clarity: Annual Totals/Means and
      Monthly Time Series.

      // 5.1 Export Annual Total and Mean Daily Data
      Export.table.toDrive({
        collection: annualStatsFC.select([
          'Year',
          'Annual_Total_Rainfall_mm',
          'Annual_Mean_Daily_Rainfall_mm'
        ]),
        description: 'Kolkata_Annual_Rainfall_Summary_CSV',
        folder: 'GEE_Exports',
        fileNamePrefix: 'Kolkata_Annual_Rainfall_Summary_Data',
        fileFormat: 'CSV'
      });

      // 5.2 Export Monthly Time Series Data
      Export.table.toDrive({
        collection: monthlyTimeSeriesFC.select([
          'Year',
          'Month_Name',
          'Mean_Daily_Rainfall_mm'
        ]),
        description: 'Kolkata_Monthly_Rainfall_TimeSeries_CSV',
        folder: 'GEE_Exports',
        fileNamePrefix: 'Kolkata_Monthly_Rainfall_TimeSeries_Data',
        fileFormat: 'CSV'
      });


print('Rainfall Analysis Data CSV export tasks initialized. You must
click "Run" in the Earth Engine "Tasks" tab to start the file transfer
to Google Drive.');
```

9. Data: Kolkata Urban Area Trend (2001 – 2023)

Output file name: Plot

Code:

```
      /**
       * Google Earth Engine Script to Analyze ANNUAL Urban Area
      Trend
       * in Kolkata, India, from 2001 to 2023.
       *
       * This script calculates the total 'Urban and Built-up' area
      for every year
       * in the defined period using the MODIS MCD12Q1 product,
      visualizing the trend
       * in a time-series chart.
       *
```

```
 * Author: Gemini
 * Date: November 2025
 */

// --- 1. Define Area of Interest (AOI) and Time Periods ---

// Coordinates for Kolkata (West Bengal, India)
var aoi = ee.Geometry.Point(88.36, 22.57).buffer(30000); // 30km buffer
Map.centerObject(aoi, 8);
Map.addLayer(aoi, {color: '000000'}, 'AOI: Kolkata Region');

var START_YEAR = 2001; // Start year for time series
var END_YEAR = 2023; // End year for time series

// Define the LULC code for Urban/Built-up in the grouped classification
var URBAN_CODE = 4;
var URBAN_NAME = '4. Urban and Built-up';


// --- 2. Load and Prepare MODIS Land Cover Data (MCD12Q1 V6) ---

var LULC_COLLECTION = ee.ImageCollection('MODIS/061/MCD12Q1');

// IGBP codes and our grouped codes (where 4 = Urban)
var ORIGINAL_IGBP_CODES = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17];
var GROUPED_LULC_CODES =  [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 4, 2, 2, 5]; // 4 = Urban

/**
 * Function to filter by year, reclassify, and calculate the urban area.
 * @param {ee.Number} year The year to calculate statistics for.
 * @return {ee.Feature} A feature containing the year and calculated urban area in km².
 */
var calculateAnnualUrbanArea = function(year) {
  year = ee.Number(year);
  var date = ee.Date.fromYMD(year, 1, 1);

  var lulcImage = LULC_COLLECTION
    .filter(ee.Filter.calendarRange(year, year, 'year'))
    .first()
    .select('LC_Type1')
    .clip(aoi);

  // Reclassify to get the grouped LULC codes (where 4 is Urban)
  var lulc_reclass = lulcImage.remap(ORIGINAL_IGBP_CODES, GROUPED_LULC_CODES);

  // Create Urban mask: 1 where Urban (Code 4), 0 otherwise
  var urbanMask = lulc_reclass.eq(URBAN_CODE).rename('Urban_Mask');

  // Calculate the total area of the Urban class from the mask
```

```
  var                          stats                                =
urbanMask.multiply(ee.Image.pixelArea()).reduceRegion({
    reducer: ee.Reducer.sum(),
    geometry: aoi,
    scale: 500, // MODIS resolution
    maxPixels: 1e10
  });

  // Convert m² to km²
  var                       area_km2                                =
ee.Number(stats.get('Urban_Mask')).divide(1e6);

  // Return a Feature for charting, using the date as the time
axis property
  return ee.Feature(null, {
    'Year': year,
    'system:time_start': date.millis(),
    'Urban_Area_km2': area_km2
  });
};


// --- 3. Area Calculation and Time Series Generation ---

// Generate a list of years from 2001 to 2023
var years = ee.List.sequence(START_YEAR, END_YEAR);

// Calculate urban area for every year and store in a
FeatureCollection
var                    urbanTimeSeriesFC                           =
ee.FeatureCollection(years.map(calculateAnnualUrbanArea));


// --- 4. Visualization (Chart) and Map Display ---

// 4.1. Chart: Annual Urban Area Trend
var urbanAreaChart = ui.Chart.feature.byFeature({
    features: urbanTimeSeriesFC,
    xProperty: 'system:time_start', // Use system:time_start
for proper date scaling
    yProperties: ['Urban_Area_km2']
  })
  .setChartType('LineChart')
  .setOptions({
    title: 'Kolkata Annual Urban Area Trend (km²): ' +
START_YEAR + ' - ' + END_YEAR,
    hAxis: {title: 'Year', format: 'yyyy'},
    vAxis: {title: 'Urban Area (km²)'},
    colors: ['FF0000'],
    // Add a linear trendline to visualize the long-term growth
rate
    trendlines: {
      0: {
        type: 'linear',
        color: 'gray',
        visibleInLegend: true,
        labelInLegend: 'Long-term Trend',
      }
    },
    legend: {position: 'none'}
  });
```

```
print('Urban Area Time Series Chart:', urbanAreaChart);


// 4.2. Map: Visualize Urban Expansion (Start vs. End Year)
// Helper function to get the urban mask for map visualization
var getUrbanMask = function(year) {
  var lulcImage = LULC_COLLECTION
    .filter(ee.Filter.calendarRange(year, year, 'year'))
    .first()
    .select('LC_Type1')
    .clip(aoi);
  var lulc_reclass = lulcImage.remap(ORIGINAL_IGBP_CODES,
GROUPED_LULC_CODES);
  return lulc_reclass.eq(URBAN_CODE).rename('Urban_Mask');
};

var urban_mask_early = getUrbanMask(START_YEAR);
var urban_mask_recent = getUrbanMask(END_YEAR);

// Calculate Urban Expansion: Not urban in early, IS urban in
recent.
var                    urban_expansion                    =
urban_mask_early.eq(0).and(urban_mask_recent.eq(1)).selfMask(
);

// Stable Urban: Was urban in early, IS still urban in recent.
var stable_urban = urban_mask_early.selfMask();

// Visualization
var urbanVis = {min: 1, max: 1, palette: ['808080']}; // Stable
Urban (Grey)
var expansionVis = {min: 1, max: 1, palette: ['FF0000']}; //
New Expansion (Red)

Map.addLayer(stable_urban, urbanVis, 'Stable Urban ' +
START_YEAR, true);
Map.addLayer(urban_expansion, expansionVis, 'Urban Expansion '
+ START_YEAR + '-' + END_YEAR, true);
print('Urban Change Layers (Stable Grey, Expansion Red) added
to Map.');


// --- 5. Export Data ---

// Export the full annual time series data to Google Drive
(CSV)
Export.table.toDrive({
  collection:              urbanTimeSeriesFC.select(['Year',
'Urban_Area_km2']),
  description: 'Kolkata_Annual_Urban_Area_TimeSeries_CSV',
  folder: 'GEE_Exports',
  fileNamePrefix: 'Kolkata_Annual_Urban_Area_TimeSeries',
  fileFormat: 'CSV'
});

print('Urban Area Time Series Data CSV export task initialized
via Export.table.');
print('You must click "Run" in the Earth Engine "Tasks" tab to
start the file transfer to Google Drive.');
```

Codes related to GCM & CMIP6 for Scenario based Projections

10. Data: Temperature and Humidity Projection

Output file name: Temperature Humidity Projection

Code:

```
/**
 * Google Earth Engine Script to Project Future Annual
Temperature and Humidity Trends
 * for Kolkata using the CMIP6 climate model dataset across
ALL SSP scenarios.
 *
 * It uses the CanESM5 model for consistency.
 */

// --- 1. CONFIGURATION AND AOI DEFINITION (Kolkata) ---
var AOI_CENTER = ee.Geometry.Point(88.3697, 22.5726); //
Kolkata coordinates
var BUFFER_KM = 30;
var waters_aoi = AOI_CENTER.buffer(BUFFER_KM * 1000).bounds();

// GCM and Scenario Selection
var GCM_MODEL = 'CanESM5';
var SCENARIOS = ['ssp126', 'ssp245', 'ssp370', 'ssp585'];

// Time Range for Time Series Analysis
var START_YEAR = 1985;
var END_YEAR = 2070;

var years = ee.List.sequence(START_YEAR, END_YEAR);

// Constants
var KELVIN_TO_CELSIUS = -273.15;
var HUMIDITY_SCALE = 1000; // Convert kg/kg to g/kg

Map.centerObject(AOI_CENTER, 8);
Map.addLayer(waters_aoi, {color: 'AA0000', opacity: 0.2},
'Kolkata 30km AOI');
print('Annual Temperature and Humidity Trend (1985-2070) for
All SSPs using Model:', GCM_MODEL);


// --- 2. MASTER FUNCTION TO PROCESS A SINGLE VARIABLE FOR ALL
SCENARIOS ---

/**
 * Runs the time series analysis for a specific climate
variable across all SSPs.
 * @param {ee.String} variable The band name ('tas' or 'huss').
 * @param {ee.String} outputName The desired column name (e.g.,
'Mean_Temperature_C').
 * @param {ee.Number} offset The value to subtract (e.g.,
KELVIN_TO_CELSIUS).
 * @param {ee.Number} scale The value to multiply by (e.g.,
HUMIDITY_SCALE).
 * @return {ee.FeatureCollection} Rainfall time series data
for the given scenario.
 */
var processVariable = function(variable, outputName, offset,
scale) {

  // Internal function to process one SSP
  var processScenario = function(scenario) {
```

```
    var ssp = ee.String(scenario);

    // Load data filtered by GCM, current SSP, and variable
    var collection = ee.ImageCollection('NASA/GDDP-CMIP6')
      .filter(ee.Filter.and(
        ee.Filter.eq('model', GCM_MODEL),
        ee.Filter.eq('scenario', ssp)
      ))
      .filterDate(START_YEAR + '-01-01', END_YEAR + '-12-31')
      .select(variable);

    /**
     * Calculates the mean of the variable for one year.
     */
    var calculateAnnualMean = function(year) {
      var year_number = ee.Number(year);
      var  year_start_date  =  ee.Date.fromYMD(year_number,  1,
1);
      var year_end_date = ee.Date.fromYMD(year_number.add(1),
1, 1);

      var                  yearlyCollection                  =
collection.filterDate(year_start_date, year_end_date);

      // Calculate the mean over all time steps in the year
      var annualMean = yearlyCollection.mean();

      var is_null = annualMean.bandNames().size().eq(0);

      // Apply offset and scale (conversion from K to C or
kg/kg to g/kg)
      var finalImage = ee.Algorithms.If(is_null,
        ee.Image.constant(0),
        ee.Image(annualMean).add(offset).multiply(scale)
      );

      // Rename and set the year index
      //  FIX: Convert the year number to a string for
system:index compatibility.
      return
ee.Image(finalImage).rename(outputName).set('system:index',
ee.String(year_number));
    };

    /**
     * Reduces the image to the mean value over the AOI and
creates a Feature.
     */
    var imageToFeature = function(image) {
      // The system:index is now a string thanks to the fix
above
      var year_string = image.get('system:index');

      // Calculate the mean over the AOI
      var stats = image.reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: waters_aoi,
        scale: 1000,
        maxPixels: 1e9,
        bestEffort: true
      });
```

```
      var mean_value = stats.get(outputName);

      // Create a base feature with static properties
      var feature = ee.Feature(null, {
        'Year':  ee.Number.parse(ee.String(year_string)),  //
Parse back to number for charting/joining
        'SSP': ssp,
      });

      //  Dynamically  set  the  variable's  value  using  the
variable outputName as the key
      return feature.set(outputName, mean_value);
    };

    //   Calculate   annual   means   and   convert   to   a
FeatureCollection
    var                annual_mean_collection                =
ee.ImageCollection(years.map(calculateAnnualMean));
    return annual_mean_collection.map(imageToFeature);
  };

  //   Iterate   over   all   SCENARIOS   and   combine   all
FeatureCollections into one.
  return
ee.FeatureCollection(ee.List(SCENARIOS).map(processScenario))
.flatten();
};


// --- 3. MAIN EXECUTION AND DATA PREPARATION ---

// 3a. Process Temperature (tas)
var temp_features = processVariable(
  'tas',
  'Mean_Temperature_C',
  KELVIN_TO_CELSIUS,
  1 // No scaling needed after offset
);

// 3b. Process Specific Humidity (huss)
var humidity_features = processVariable(
  'huss',
  'Mean_Specific_Humidity_g_kg',
  0, // No offset needed
  HUMIDITY_SCALE
);

// 3c. Merge the two collections by joining on 'Year' and 'SSP'
properties
// We must join on both Year and SSP to ensure we match the
right data points
var filter = ee.Filter.and(
  ee.Filter.equals({leftField: 'Year', rightField: 'Year'}),
  ee.Filter.equals({leftField: 'SSP', rightField: 'SSP'})
);

var simpleJoin = ee.Join.inner();

var      joined      =      simpleJoin.apply(temp_features,
humidity_features, filter);
```

```
// Extract the joined features and combine the fields
var final_features = joined.map(function(feature) {
  var primary = ee.Feature(feature.get('primary'));
  var secondary = ee.Feature(feature.get('secondary'));

  // Copy all properties from the secondary feature (humidity)
into the primary feature (temp)
  return                    primary.copyProperties(secondary,
['Mean_Specific_Humidity_g_kg']);
});


// --- 4. CHARTING ---
var   SERIES_COLORS   =   ['#00A859',   '#5C90ED',   '#FF7F00',
'#D62728']; // ssp126, ssp245, ssp370, ssp585

// 4a. Temperature Chart
var tempChart = ui.Chart.feature.groups({
  features: temp_features,
  xProperty: 'Year',
  yProperty: 'Mean_Temperature_C',
  seriesProperty: 'SSP'
})
.setChartType('LineChart')
.setOptions({
  title: 'Kolkata Mean Annual Temperature Trend (°C) by SSP
Scenario',
  vAxis: {title: 'Temperature (°C)'},
  hAxis: {title: 'Year', format: 'YYYY'},
  legend: {position: 'right'},
  series: {
    0: { color: SERIES_COLORS[0], lineWidth: 2, pointSize: 2,
label: 'ssp126'},
    1: { color: SERIES_COLORS[1], lineWidth: 2, pointSize: 2,
label: 'ssp245'},
    2: { color: SERIES_COLORS[2], lineWidth: 2, pointSize: 2,
label: 'ssp370'},
    3: { color: SERIES_COLORS[3], lineWidth: 2, pointSize: 2,
label: 'ssp585'},
  },
  interpolateNulls: true,
});

print('Mean   Annual   Temperature   Time   Series   Chart:',
tempChart);

// 4b. Humidity Chart
var humidityChart = ui.Chart.feature.groups({
  features: humidity_features,
  xProperty: 'Year',
  yProperty: 'Mean_Specific_Humidity_g_kg',
  seriesProperty: 'SSP'
})
.setChartType('LineChart')
.setOptions({
  title: 'Kolkata Mean Annual Specific Humidity Trend (g/kg)
by SSP Scenario',
  vAxis: {title: 'Specific Humidity (g/kg)'},
  hAxis: {title: 'Year', format: 'YYYY'},
  legend: {position: 'right'},
```

```
    series: {
      0: { color: SERIES_COLORS[0], lineWidth: 2, pointSize: 2,
label: 'ssp126'},
      1: { color: SERIES_COLORS[1], lineWidth: 2, pointSize: 2,
label: 'ssp245'},
      2: { color: SERIES_COLORS[2], lineWidth: 2, pointSize: 2,
label: 'ssp370'},
      3: { color: SERIES_COLORS[3], lineWidth: 2, pointSize: 2,
label: 'ssp585'},
    },
    interpolateNulls: true,
  });

  print('Mean Annual Specific Humidity Time Series Chart:',
  humidityChart);


  // --- 5. CSV EXPORT SETUP ---

  Export.table.toDrive({
    collection: final_features,
    description: 'Kolkata_Temp_Humidity_All_SSPs_TimeSeries',
    folder: 'EarthEngine_Exports',
    fileNamePrefix: 'kolkata_temp_humidity_all_ssps',
    fileFormat: 'CSV',
    // Define column order for the final CSV
    selectors:    ['Year',    'SSP',    'Mean_Temperature_C',
'Mean_Specific_Humidity_g_kg']
  });

  print('--- CSV Export Information ---');
  print('A                    task                    named
  "Kolkata_Temp_Humidity_All_SSPs_TimeSeries"  has  been  set
  up.');
print('To download the comprehensive CSV, go to the **Tasks** tab
(on the right side of the code editor) and click **Run** on the
task.');
```

11. Data: Rainfall Projection

Output file name: Rainfall Projection

Code:

```
  /**
   * Google Earth Engine Script to Project Future Annual Rainfall
  Patterns in Kolkata
   * using the CMIP6 climate model dataset across ALL SSP
  scenarios.
   *
   * This script runs the analysis for: ssp126, ssp245, ssp370,
  and ssp585.
   * It generates a multi-line time series chart and sets up a
  single CSV export file.
   */

  // --- 1. CONFIGURATION AND AOI DEFINITION (Kolkata) ---
  var  AOI_CENTER  =  ee.Geometry.Point(88.3697,  22.5726);  //
  Kolkata coordinates
  var BUFFER_KM = 30;
  var waters_aoi = AOI_CENTER.buffer(BUFFER_KM * 1000).bounds();

  // GCM and Scenario Selection
```

```
var GCM_MODEL = 'CanESM5';
var SCENARIOS = ['ssp126', 'ssp245', 'ssp370', 'ssp585'];

// Time Range for Time Series Analysis
var START_YEAR = 1985;
var END_YEAR = 2070;

// Constants
var SECONDS_PER_DAY = 86400;
var years = ee.List.sequence(START_YEAR, END_YEAR);

Map.centerObject(AOI_CENTER, 8);
Map.addLayer(waters_aoi, {color: 'AA0000', opacity: 0.2},
'Kolkata 30km AOI');
print('Annual Rainfall Trend (1985-2070) for All SSPs using
Model:', GCM_MODEL);


// --- 2. MASTER FUNCTION TO PROCESS EACH SCENARIO ---

/**
 * Runs the annual precipitation calculation and feature
extraction for a single scenario.
 * @param {ee.String} scenario The SSP scenario name (e.g.,
'ssp245').
 * @return {ee.FeatureCollection} Rainfall time series data
for the given scenario.
 */
var processScenario = function(scenario) {
  var ssp = ee.String(scenario);

  // Load data filtered by GCM and current SSP
  var precipitation_collection =
ee.ImageCollection('NASA/GDDP-CMIP6')
    .filter(ee.Filter.and(
      ee.Filter.eq('model', GCM_MODEL),
      ee.Filter.eq('scenario', ssp)
    ))
    .filterDate(START_YEAR + '-01-01', END_YEAR + '-12-31')
    .select('pr');

  /**
   * Calculates total precipitation for one year.
   * @param {ee.Number} year The current year as an ee.Number
object.
   * @return {ee.Image} The annual total precipitation image.
   */
  var calculateAnnualPpt = function(year) {
    var year_number = ee.Number(year);
    var year_start_date = ee.Date.fromYMD(year_number, 1, 1);
    var year_end_date = ee.Date.fromYMD(year_number.add(1), 1,
1);

    var yearlyCollection =
precipitation_collection.filterDate(year_start_date,
year_end_date);
    var annualSum = yearlyCollection.sum();

    var is_null = annualSum.bandNames().size().eq(0);

    var annualPptImage = ee.Algorithms.If(is_null,
```

```
      ee.Image.constant(0),
      annualSum.multiply(SECONDS_PER_DAY)  //  Convert  daily
rates (mm/s) to Total mm/year
    );

    var                      finalImage                      =
ee.Image(annualPptImage).rename('Annual_Rainfall_mm');
    // Set system:time_start for internal date handling if
needed, though we use the 'Year' property
    return finalImage;
  };

  /**
   * Reduces the image to the mean value over the AOI and
creates a Feature.
   *   @param   {ee.Image}   image   An   image   containing
'Annual_Rainfall_mm'.
   * @return {ee.Feature} A feature with 'Year', 'Rainfall',
and 'SSP'.
   */
  var imageToFeature = function(image) {
    // Get the year property from the map iteration (not
automatically on the image)
    var year = image.get('system:index');

    // Calculate the mean over the AOI
    var stats = image.reduceRegion({
      reducer: ee.Reducer.mean(),
      geometry: waters_aoi,
      scale: 1000,
      maxPixels: 1e9,
      bestEffort: true
    });

    var mean_rainfall = stats.get('Annual_Rainfall_mm');

    // Create a feature with the desired properties
    return ee.Feature(null, {
      'Year': ee.Number.parse(ee.String(year)), // Convert the
index string back to a number
      'Mean_Annual_Rainfall_mm': mean_rainfall,
      'SSP': ssp // Attach the scenario name to the data point
    });
  };

  // Calculate annual totals and convert to a FeatureCollection
  var                  annual_ppt_collection                  =
ee.ImageCollection(years.map(calculateAnnualPpt));
  var                    rainfall_features                    =
annual_ppt_collection.map(imageToFeature);

  return rainfall_features;
};

//   Iterate   over   all   SCENARIOS   and   combine   all
FeatureCollections into one.
var                    all_rainfall_data                     =
ee.FeatureCollection(ee.List(SCENARIOS).map(processScenario))
.flatten();
```

```
      // --- 3. CHARTING (TIME SERIES) ---

      // Chart features by grouping them based on the 'SSP' property
      var timeSeriesChart = ui.Chart.feature.groups({
        features: all_rainfall_data,
        xProperty: 'Year',
        yProperty: 'Mean_Annual_Rainfall_mm',
        seriesProperty: 'SSP' // This creates a separate line for
      each scenario
      })
      .setChartType('LineChart')
      .setOptions({
        title: 'Kolkata Annual Rainfall Trend (1985-2070) by SSP
      Scenario',
        vAxis: {title: 'Mean Annual Rainfall (mm)'},
        hAxis: {title: 'Year', format: 'YYYY'},
        legend: {position: 'right'},
        series: {
          0: { color: '#00A859', lineWidth: 2, pointSize: 2, label:
      'ssp126 (Sustainable)'}, // Green
          1: { color: '#5C90ED', lineWidth: 2, pointSize: 2, label:
      'ssp245 (Mid-Road)'}, // Blue
          2: { color: '#FF7F00', lineWidth: 2, pointSize: 2, label:
      'ssp370 (Regional)'}, // Orange
          3: { color: '#D62728', lineWidth: 2, pointSize: 2, label:
      'ssp585 (Fossil-Fueled)'}, // Red
        },
        interpolateNulls: true,
      });

      print('Kolkata Annual Rainfall Time Series Chart:',
      timeSeriesChart);


      // --- 4. CSV EXPORT SETUP ---

      // Define the export task for the combined collection
      Export.table.toDrive({
        collection: all_rainfall_data,
        description: 'Kolkata_Rainfall_All_SSPs_TimeSeries',
        folder: 'EarthEngine_Exports',
        fileNamePrefix: 'kolkata_rainfall_all_ssps',
        fileFormat: 'CSV',
        selectors: ['Year', 'SSP', 'Mean_Annual_Rainfall_mm'] //
      Define column order
      });

      print('--- CSV Export Information ---');
      print('A task named "Kolkata_Rainfall_All_SSPs_TimeSeries" has
      been set up.');
print('To download the comprehensive CSV, go to the **Tasks** tab
(on the right) and click **Run** on the task.');
```

R Codes

1. R Script to analyse and visualize temperature and humidity trends

```
# R Script for Climate Trend Analysis and Forecasting (1990-Present)
# This version uses Base R for data handling, ggplot2 for plotting, and
gridExtra for multi-panel arrangement.

# ------------------------------------------------------------------------
```

```
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------

# Load necessary packages.
library(ggplot2)
# gridExtra is used for combining the individual plots into a single multi-
panel view.
library(gridExtra)
# grid is REQUIRED for textGrob and gpar functions used by grid.arrange
for custom titles/captions.
library(grid)
library(stats) # For lm() and predict() - this is a base package

# Define the file paths for the uploaded CSVs
file_max_min <- 'Monthly Max Min Temp.csv'
file_mean_ah <- 'Monthly Mean Temp_AH.csv'
file_mean_rh <- 'Monthly Mean Temp_RH.csv'

# Load the dataframes using base R's read.csv
df_max_min <- read.csv(file_max_min, stringsAsFactors = FALSE)
df_mean_ah <- read.csv(file_mean_ah, stringsAsFactors = FALSE)
df_mean_rh <- read.csv(file_mean_rh, stringsAsFactors = FALSE)

# ------------------------------------------------------------------
# 2. DATA PREPARATION AND MERGING (Using Base R Syntax)
# ------------------------------------------------------------------

# Function for data cleaning and preparation
clean_data <- function(df, prefix) {
  # Create a standard Date column
  df$Date <- as.Date(paste(df$year, df$month, '01', sep = '-'), format =
'%Y-%m-%d')

  # Select and rename relevant columns
  if (prefix == "max_min") {
    df <- df[, c("Date", "max_temp_c", "min_temp_c")]
    colnames(df) <- c("Date", "MaxTemp", "MinTemp")
  } else if (prefix == "mean_ah") {
    df <- df[, c("Date", "mean_temp_c", "mean_ah_g_m3")]
    colnames(df) <- c("Date", "MeanTemp", "AH")
  } else if (prefix == "mean_rh") {
    df <- df[, c("Date", "mean_rh_percent")]
    colnames(df) <- c("Date", "RH")
  }
  return(df)
}

df_max_min_clean <- clean_data(df_max_min, "max_min")
df_mean_ah_clean <- clean_data(df_mean_ah, "mean_ah")
df_mean_rh_clean <- clean_data(df_mean_rh, "mean_rh")

# Perform the merge using base R's merge()
df_combined <- merge(df_max_min_clean, df_mean_ah_clean, by = "Date", all
= TRUE)
df_combined <- merge(df_combined, df_mean_rh_clean, by = "Date", all =
TRUE)

# Filter out NA Dates and create the numeric time variable
df_combined <- df_combined[!is.na(df_combined$Date), ]
```

```
# Create a numeric time variable (fractional year) for the original
aggregate trend fitting
df_combined$Time <- as.numeric(format(df_combined$Date, "%Y")) +
  (as.numeric(format(df_combined$Date, "%j")) - 1) / 365.25

# *** KEY ADDITION for Monthly Breakdown: Extract Year and Month ***
df_combined$Year <- as.numeric(format(df_combined$Date, "%Y"))
df_combined$Month <- as.numeric(format(df_combined$Date, "%m"))

# Define the variables and labels for analysis (Keeping only Temperature
metrics)
analysis_vars <- c("MaxTemp", "MinTemp", "MeanTemp")
var_labels <- c(
  "Monthly Max Temp (°C)",
  "Monthly Min Temp (°C)",
  "Monthly Mean Temp (°C)",
  "Monthly Relative Humidity (%)", # Kept for consistency, but will not be
used
  "Monthly Absolute Humidity (g/m³)" # Kept for consistency, but will not
be used
)
names(var_labels) <- c("MaxTemp", "MinTemp", "MeanTemp", "RH", "AH") #
Renaming to ensure correct mapping

# ----------------------------------------------------------------------
# 3. DESCRIPTIVE STATISTICS & MONTHLY TREND VISUALIZATION
# ----------------------------------------------------------------------

cat("======================================================================\n"
)
cat("          DESCRIPTIVE STATISTICS (1990 - Present)\n")
cat("======================================================================\n\
n")

# Helper function to calculate the monthly trends (change from first to
last point)
calculate_monthly_trends <- function(data, var_name) {
  month_names <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec")

  # Create a factor for Month for plotting consistency
  monthly_trends <- data.frame(Month = factor(month_names, levels =
month_names),
                               Start_Year = numeric(12),
                               End_Year = numeric(12),
                               Total_Change = numeric(12),
                               Direction = character(12),
                               stringsAsFactors = FALSE)

  # Define the trend threshold for 'Stable'
  threshold <- 0.1

  for (m in 1:12) {
    # Filter data for the current month, excluding NAs for the variable
    month_data <- data[data$Month == m & !is.na(data[, var_name]), ]

    total_change <- NA
    direction <- "Insufficient Data"
    start_year_calc <- NA
    end_year_calc <- NA
```

```
    if (nrow(month_data) >= 2) {
      # Sort the data by Year to reliably get the earliest and latest
points
      month_data <- month_data[order(month_data$Year), ]

      # Get the earliest and latest available years
      start_year_calc <- month_data$Year[1]
      end_year_calc <- month_data$Year[nrow(month_data)]

      # Get the value corresponding to the earliest year (first row after
sort)
      start_val <- month_data[1, var_name]

      # Get the value corresponding to the latest year (last row after
sort)
      end_val <- month_data[nrow(month_data), var_name]

      # Calculate change and direction
      if (!is.na(start_val) && !is.na(end_val)) {
        total_change <- end_val - start_val

        if (abs(total_change) < threshold) {
          direction <- "Stable"
        } else if (total_change > 0) {
          direction <- "Increasing"
        } else {
          direction <- "Decreasing"
        }
      }
    }

    # Assign values to the data frame columns individually to prevent R
from coercing
    #  the numeric 'Total_Change' column into a character type when
assigning mixed types.
    monthly_trends[m, "Start_Year"] <- start_year_calc
    monthly_trends[m, "End_Year"] <- end_year_calc
    monthly_trends[m, "Total_Change"] <- total_change
    monthly_trends[m, "Direction"] <- direction
  }

  # Clean up: Remove NAs/NaNs for plotting, which usually indicate
"Insufficient Data"
  monthly_trends$Total_Change[is.na(monthly_trends$Total_Change)] <- 0
  monthly_trends$Direction <- factor(monthly_trends$Direction,
                                     levels       =       c("Increasing",
"Decreasing", "Stable", "Insufficient Data"))

  return(monthly_trends)
}


# Function to create and print overall descriptive stats
print_overall_stats <- function(data, var_name, label) {
  x <- data[, var_name]

  stats <- data.frame(
    N = sum(!is.na(x)),
    Mean = mean(x, na.rm = TRUE),
    SD = sd(x, na.rm = TRUE),
    Min = min(x, na.rm = TRUE),
```

```
    Q25 = quantile(x, 0.25, na.rm = TRUE),
    Median = median(x, na.rm = TRUE),
    Q75 = quantile(x, 0.75, na.rm = TRUE),
    Max = max(x, na.rm = TRUE)
  )

  cat(sprintf("--- %s ---\n", label))
  cat("--- Overall Statistics (All Months Combined) ---\n")
  print(round(stats, 3), row.names = FALSE)
  cat("\n")
}

# Function to generate the monthly change bar plot
plot_monthly_change <- function(trend_data, var_name, full_label) {

  # Define colors for the direction factor
  trend_colors <- c("Increasing" = "#E41A1C", # Red
                    "Decreasing" = "#377EB8", # Blue
                    "Stable" = "#AAAAAA",     # Gray
                    "Insufficient Data" = "#DDDDDD") # Light Gray/NA


  # Determine the unit for the y-axis label
  unit <- sub(".*\\((.*)\\)", "\\1", full_label)

  p <- ggplot(trend_data, aes(x = Month, y = Total_Change, fill =
Direction)) +

    # Use geom_bar for the change value
    geom_bar(stat = "identity", width = 0.8, alpha = 0.9) +

    # Add labels showing the exact change value above/below the bar
    geom_text(aes(label = ifelse(Direction != "Insufficient Data",
                                 format(round(Total_Change, 2), nsmall =
2), "")),
              # *** FIX: Adjusted vjust to increase vertical spacing for
better label visibility ***
              vjust = ifelse(trend_data$Total_Change >= 0, -1.0, 2.0),
              color = "black", size = 3) +

    # Line at zero change
    geom_hline(yintercept = 0, color = "black", linewidth = 0.5) +

    # Custom color scale
    scale_fill_manual(values = trend_colors, drop = FALSE) +

    # Labels and Theme
    labs(
      title = paste("Monthly Total Change:", full_label),
      subtitle = "Change calculated from the earliest available data point
to the latest.",
      x = "Month",
      y = paste0("Total Change (", unit, ")"),
      fill = "Trend Direction"
    ) +
    theme_minimal(base_size = 11) +
    theme(
      plot.title = element_text(face = "bold", size = 14, hjust = 0.5),
      plot.subtitle = element_text(size = 10, hjust = 0.5, color =
"gray50"),
      axis.title.x = element_blank(),
```

```
      axis.text.x = element_text(face = "bold"),
      legend.position = "bottom"
    )

  return(p)
}

# Storage for the plots
monthly_change_plots <- list()
counter <- 1

for (var in analysis_vars) {
  # 1. Print overall descriptive statistics (unmodified)
  print_overall_stats(df_combined, var, var_labels[var])

  # 2. Calculate monthly trends
  trend_data <- calculate_monthly_trends(df_combined, var)

  # 3. Generate the visualization
  p <- plot_monthly_change(trend_data, var, var_labels[var])
  monthly_change_plots[[counter]] <- p
  counter <- counter + 1
}

# ------------------------------------------------------------------------
# 4. MONTHLY TREND ANALYSIS AND PLOTTING (3 Multi-Panel Plots)
# ------------------------------------------------------------------------

# 4.1 Function to Create all 12 Monthly Trend Plots for a Single Metric
(Unmodified)
# The output is a single grob combining all 12 plots in a 4x3 layout.
create_all_monthly_plots <- function(data, var_name, full_label) {

  month_names <- c("January", "February", "March", "April", "May", "June",
                   "July", "August", "September", "October", "November",
"December")

  plot_list <- list()

  # Determine the last observed year for defining forecast start
  max_year_observed <- max(data$Year, na.rm = TRUE)

  # Create a sequence of years for the trend line, extending to 2030
  # Using a finer step (0.1) for a smooth trend line in the plot
  future_years <- seq(min(data$Year, na.rm = TRUE), 2030, by = 0.1)

  # Loop through all 12 months
  for (m in 1:12) {
    month_data <- data[data$Month == m, ]
    month_label <- month_names[m]

    # Rename the target variable for safe aesthetic mapping in ggplot
    month_data$Value <- month_data[, var_name]

    # 1. Trend Fitting: Attempting a Polynomial (Quadratic) Model with
Linear Fallback
    model <- tryCatch({
      # Check if enough unique points exist for a quadratic model (>= 3
unique years)
      if (length(unique(month_data$Year)) >= 3) {
        # Attempt 2nd-degree polynomial fit (Quadratic)
```

```
        # Using I(Year^2) for explicit term
        lm(Value ~ Year + I(Year^2), data = month_data)
      } else {
        # Fallback immediately if data is insufficient for quadratic
        lm(Value ~ Year, data = month_data)
      }
    }, warning = function(w) {
      # Fallback to linear model if quadratic fails with a warning (e.g.,
singular fit)
      lm(Value ~ Year, data = month_data)
    }, error = function(e) {
      NULL # Return NULL if even linear fails (e.g., insufficient data)
    })

    # Check if the model failed or if there is simply not enough data
    if (is.null(model) || length(unique(month_data$Year)) < 2) {
      # Create a placeholder plot for empty or sparse months
      # Use range of all data for positioning of "Insufficient Data" text
      p <- ggplot(month_data, aes(x = Year, y = Value)) +
        geom_text(aes(x = mean(data$Year, na.rm=TRUE), y = mean(data[,
var_name], na.rm=TRUE)),
                  label = "Insufficient Data", size = 3, color = "gray50")
+
        labs(title = month_label, x = NULL, y = NULL) +
        theme_void() +
        theme(plot.title = element_text(face = "bold", size = 10, hjust =
0.5))
      plot_list[[m]] <- p
      next
    }

    # 2. Create forecast data frame
    df_forecast <- data.frame(Year = future_years)
    df_forecast$Predicted <- predict(model, newdata = df_forecast)

    # Identify if the point is a true forecast (beyond the current
max_year_observed)
    df_forecast$Is_Forecast <- df_forecast$Year > max_year_observed

    # 3. Create the ggplot for the month
    p <- ggplot(month_data, aes(x = Year, y = Value)) +

      # Layer 1: Observed data points (scatter) - slightly smaller
      geom_point(color = "#377EB8", size = 1.0, alpha = 0.7) +

      # Layer 2: Trend line (Historical part) - Red
      geom_line(
        data = df_forecast[!df_forecast$Is_Forecast & df_forecast$Year >=
min(month_data$Year), ],
        aes(x = Year, y = Predicted),
        color = "#E41A1C", # Red for the historical fitted trend
        size = 1.2 # Thicker line
      ) +

      # Layer 3: Forecast line (Future part) - Green, dashed, with arrowhead
      geom_line(
        data = df_forecast[df_forecast$Is_Forecast, ],
        aes(x = Year, y = Predicted),
        color = "#4DAF4A", # Changed color to Green
        size = 1.2, # Thicker line
        linetype = "dashed", # Use dashed line for projection
```

```
        # Added arrowhead to clearly indicate forecast direction
        arrow = arrow(length = unit(0.15, "inches"), ends = "last", type
= "closed")
      ) +

      # Layer 4: Visualization Polish
      labs(
        title = month_label,
        x = NULL,
        y = NULL
      ) +
      theme_minimal(base_size = 9) +
      theme(
        plot.title = element_text(face = "bold", size = 10, hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1, size = 8),
        # Remove Y-axis text in small panels to remove clutter
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        plot.margin = unit(c(0.2, 0.2, 0.2, 0.2), "cm")
      ) +
      # Ensure the x-axis is on a year scale and extends to 2030
      scale_x_continuous(
        breaks = seq(floor(min(data$Year, na.rm = TRUE)), 2030, by = 10),
        limits = c(floor(min(data$Year, na.rm = TRUE)), 2030)
      )

    plot_list[[m]] <- p
  }

  # 4. Combine all 12 plots into a single output using grid.arrange

  # Main Title
  main_title <- textGrob(paste("Monthly Trend Detail and Forecast:",
full_label),
                         gp=gpar(fontsize=16, fontface="bold"))

  # Subtitle
  trend_subtitle <- textGrob("Individual monthly change with 2nd-degree
polynomial fit projection to 2030",
                             gp=gpar(fontsize=12, col="gray30"))

  # The Y-axis label is placed to the left of the 4x3 plot grid
  y_axis_label <- textGrob(full_label, rot = 90, gp=gpar(fontsize=12))

  # Plot caption focusing only on color/line type for clarity
  plot_caption <- textGrob("Observed Data (Blue Points); Historical Trend
(Red Solid); Forecast (Green Dashed with Arrowhead).",
                           gp=gpar(fontsize=10, col="gray50"), hjust =
0.5, x = 0.5)

  # Arrange the components (Title, Subtitle, Y-Label, 12 Plots, Caption)
  combined_plot <- grid.arrange(
    main_title,
    trend_subtitle,
    y_axis_label,
    # Combine the 12 plots into a single grob in a 4x3 matrix
    do.call(arrangeGrob, c(plot_list, ncol = 3, nrow = 4)),
    plot_caption,
    ncol = 2,
```

```
    # Define widths: narrow column for Y-label, wide column for 4x3 plots
    widths = unit.c(unit(0.5, "in"), unit(1, "null")),
    # Adjust heights for the title and new subtitle
    heights = unit.c(unit(0.3, "in"), unit(0.2, "in"), unit(1, "null"),
unit(0.3, "in")),
    # Layout matrix:
    # Row 1: NA, Title
    # Row 2: NA, Subtitle
    # Row 3: Y-Label, 4x3 Plots (as a single unit/grob)
    # Row 4: Caption (takes full width)
    layout_matrix = rbind(c(NA, 1),
                          c(NA, 2),
                          c(3, 4),
                          c(5, 5))
  )

  return(combined_plot)
}

# 4.2 Generate and Display the Multi-Panel Plots
cat("\n\n=================================================================
=\n")
cat("                GENERATING TREND SUMMARY PLOTS (3 Panels: Max, Min,
Mean Temp)\n")
cat("=================================================================\n\
n")

# Combine the 3 monthly change plots into a single, cohesive visual summary
trend_summary_title <- textGrob("Long-Term Monthly Temperature Change
Summary (1990 - Present)",
                                gp=gpar(fontsize=18, fontface="bold"))
trend_summary_subtitle <- textGrob("Total Change from First to Last Data
Point by Month",
                                   gp=gpar(fontsize=12, col="gray50"))

# Display the 3 plots in a 1x3 grid
combined_change_plots <- grid.arrange(
  grobs = monthly_change_plots,
  ncol = 3,
  top = trend_summary_title,
  bottom = trend_summary_subtitle
)

# Print the final arranged plot for the change summary
print(combined_change_plots)


cat("\n\n=================================================================
=\n")
cat("            GENERATING MONTHLY DETAIL PLOTS (3 Multi-Panel Plots)\n")
cat("=================================================================\n\
n")

# Plot 1: Max Temperature
plot_max_temp_monthly <- create_all_monthly_plots(df_combined, "MaxTemp",
var_labels["MaxTemp"])
print(plot_max_temp_monthly)

# Plot 2: Min Temperature
plot_min_temp_monthly <- create_all_monthly_plots(df_combined, "MinTemp",
var_labels["MinTemp"])
```

```
print(plot_min_temp_monthly)

# Plot 3: Mean Temperature
plot_mean_temp_monthly      <-      create_all_monthly_plots(df_combined,
"MeanTemp", var_labels["MeanTemp"])
print(plot_mean_temp_monthly)


# ------------------------------------------------------------------------
# 5. MODEL SUMMARY (Commented out as 60 individual models are too verbose)
# ------------------------------------------------------------------------
# (Model summary section removed to keep console output manageable)

# ------------------------------------------------------------------------
# END OF SCRIPT
# ------------------------------------------------------------------------
cat("\n\n--- INSTRUCTIONS FOR VIEWING PLOTS ---\n")
cat("The first output is a single 1x3 grid displaying the Monthly Change
Summary for the 3 temperature metrics.\n")
cat("This is followed by the 3 detailed 4x3 trend and forecast plots.\n")
cat("You may need to advance the plot viewer to cycle through all 4
results.\n")
```

2. R Script to analyse and visualize Diurnal Temperature Range (DTR)

```
# R Script for Monthly Diurnal Temperature Range (DTR) Trend Analysis
(1990-Present)
# This script loads the DTR data, calculates a long-term trend for each
month,
# and plots the observed data points with a trend-line projected to 2030
for all 12 months.

# ------------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------------

# Load necessary packages.
library(ggplot2)
# gridExtra is used for combining the individual plots into a single multi-
panel view.
library(gridExtra)
# grid is REQUIRED for textGrob and gpar functions used by grid.arrange
for custom titles/captions.
library(grid)
library(stats) # For lm() and predict()

# Define the file path for the DTR CSV
file_dtr <- 'Monthly DTR.csv'

# Load the dataframe using base R's read.csv
df_dtr <- read.csv(file_dtr, stringsAsFactors = FALSE)

# ------------------------------------------------------------------------
# 2. DATA PREPARATION
# ------------------------------------------------------------------------

# Rename columns for simpler access
colnames(df_dtr)    <-    c("Year",    "MonthNum",    "MonthName",    "DTR",
"DateString")

# Ensure Year and DTR are numeric
df_dtr$Year <- as.numeric(df_dtr$Year)
df_dtr$DTR <- as.numeric(df_dtr$DTR)
```

```
df_dtr$MonthNum <- as.numeric(df_dtr$MonthNum)

# Create a factor for MonthName to ensure plots are ordered correctly
month_names_order <- c("January", "February", "March", "April", "May",
"June",
                       "July",    "August",    "September",    "October",
"November", "December")
df_dtr$MonthName <- factor(df_dtr$MonthName, levels = month_names_order)

# Remove any rows where DTR or Year are missing
df_dtr <- df_dtr[!is.na(df_dtr$DTR) & !is.na(df_dtr$Year), ]


# ------------------------------------------------------------------------
# 3. MONTHLY TREND ANALYSIS AND PLOTTING (12 Multi-Panel Plots)
# ------------------------------------------------------------------------

# Function to Create all 12 Monthly Trend Plots for DTR
create_dtr_monthly_plots <- function(data) {

  plot_list <- list()
  full_label <- "Monthly Diurnal Temperature Range (DTR in °C)"

  # Determine the last observed year for defining forecast start
  max_year_observed <- max(data$Year, na.rm = TRUE)

  # Create a sequence of years for the trend line, extending to 2030
  future_years <- seq(min(data$Year, na.rm = TRUE), 2030, by = 0.1)

  # Set common Y-axis limits across all months for visual comparison
  y_min_limit <- min(data$DTR, na.rm=TRUE) - 1
  y_max_limit <- max(data$DTR, na.rm=TRUE) + 1

  # Loop through all 12 months (using MonthNum 1 to 12)
  for (m in 1:12) {
    # Filter data for the current month
    month_data <- data[data$MonthNum == m, ]
    month_label <- month_names_order[m]

    # 1. Trend Fitting: Attempting a Polynomial (Quadratic) Model with
Linear Fallback
    model <- tryCatch({
      # Check if enough unique points exist for a quadratic model (>= 3
unique years)
      if (length(unique(month_data$Year)) >= 3) {
        # Attempt 2nd-degree polynomial fit (Quadratic)
        lm(DTR ~ Year + I(Year^2), data = month_data)
      } else {
        # Fallback immediately if data is insufficient for quadratic
        lm(DTR ~ Year, data = month_data)
      }
    }, warning = function(w) {
      # Fallback to linear model if quadratic fails with a warning (e.g.,
singular fit)
      lm(DTR ~ Year, data = month_data)
    }, error = function(e) {
      NULL # Return NULL if even linear fails (e.g., insufficient data)
    })

    # Check if the model failed or if there is simply not enough data
    if (is.null(model) || length(unique(month_data$Year)) < 2) {
```

```
      # Create a placeholder plot for empty or sparse months
      p <- ggplot(month_data, aes(x = Year, y = DTR)) +
        geom_text(aes(x = mean(data$Year, na.rm=TRUE), y = mean(data$DTR,
na.rm=TRUE)),
                  label = "Insufficient Data", size = 3, color = "gray50")
+
        labs(title = month_label, x = NULL, y = NULL) +
        theme_void() +
        theme(plot.title = element_text(face = "bold", size = 10, hjust =
0.5))
      plot_list[[m]] <- p
      next
    }

    # 2. Create forecast data frame
    df_forecast <- data.frame(Year = future_years)
    df_forecast$Predicted <- predict(model, newdata = df_forecast)

    # Identify if the point is a true forecast (beyond the current
max_year_observed)
    df_forecast$Is_Forecast <- df_forecast$Year > max_year_observed

    # 3. Create the ggplot for the month
    p <- ggplot(month_data, aes(x = Year, y = DTR)) +

      # Layer 1: Observed data points (scatter) - Blue
      geom_point(color = "#377EB8", size = 1.0, alpha = 0.7) +

      # Layer 2: Historical Trend line (fitted part) - Red
      geom_line(
        data = df_forecast[!df_forecast$Is_Forecast & df_forecast$Year >=
min(month_data$Year), ],
        aes(x = Year, y = Predicted),
        color = "#E41A1C",
        size = 1.2
      ) +

      # Layer 3: Forecast line (Future part) - Green, dashed
      geom_line(
        data = df_forecast[df_forecast$Is_Forecast, ],
        aes(x = Year, y = Predicted),
        color = "#4DAF4A",
        size = 1.2,
        linetype = "dashed",
        arrow = arrow(length = unit(0.15, "inches"), ends = "last", type
= "closed")
      ) +

      # Layer 4: Visualization Polish
      labs(
        title = month_label,
        x = NULL,
        y = NULL
      ) +
      theme_minimal(base_size = 9) +
      theme(
        plot.title = element_text(face = "bold", size = 10, hjust = 0.5),
        axis.text.x = element_text(angle = 45, hjust = 1, size = 8),
        # Remove Y-axis text in small panels to remove clutter
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
```

```
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        plot.margin = unit(c(0.2, 0.2, 0.2, 0.2), "cm")
      ) +
      # Ensure the x-axis is on a year scale and extends to 2030
      scale_x_continuous(
        breaks = seq(floor(min(data$Year, na.rm = TRUE)), 2030, by = 10),
        limits = c(floor(min(data$Year, na.rm = TRUE)), 2030)
      ) +
      # Set common Y-axis limits across all months for visual comparison
      coord_cartesian(ylim = c(y_min_limit, y_max_limit))

    plot_list[[m]] <- p
  }

  # 4. Combine all 12 plots into a single output using grid.arrange

  # Main Title
  main_title <- textGrob(paste("Monthly Trend Detail and Forecast:",
full_label),
                         gp=gpar(fontsize=16, fontface="bold"))

  # Subtitle
  trend_subtitle <- textGrob("Individual monthly DTR change with 2nd-degree
polynomial fit projection to 2030",
                             gp=gpar(fontsize=12, col="gray30"))

  # The Y-axis label is placed to the left of the 4x3 plot grid
  y_axis_label <- textGrob(full_label, rot = 90, gp=gpar(fontsize=12))

  # Plot caption focusing only on color/line type for clarity
  plot_caption <- textGrob("Observed Data (Blue Points); Historical Trend
(Red Solid); Forecast (Green Dashed with Arrowhead). Note: All plots share
a common Y-axis range.",
                           gp=gpar(fontsize=10, col="gray50"), hjust =
0.5, x = 0.5)

  # Arrange the components (Title, Subtitle, Y-Label, 12 Plots, Caption)
  combined_plot <- grid.arrange(
    main_title,
    trend_subtitle,
    y_axis_label,
    # Combine the 12 plots into a single grob in a 4x3 matrix
    do.call(arrangeGrob, c(plot_list, ncol = 3, nrow = 4)),
    plot_caption,
    ncol = 2,
    # Define widths: narrow column for Y-label, wide column for 4x3 plots
    widths = unit.c(unit(0.5, "in"), unit(1, "null")),
    # Adjust heights for the title and new subtitle
    heights = unit.c(unit(0.3, "in"), unit(0.2, "in"), unit(1, "null"),
unit(0.3, "in")),
    # Layout matrix:
    # Row 1: NA, Title
    # Row 2: NA, Subtitle
    # Row 3: Y-Label, 4x3 Plots (as a single unit/grob)
    # Row 4: Caption (takes full width)
    layout_matrix = rbind(c(NA, 1),
                          c(NA, 2),
                          c(3, 4),
                          c(5, 5))
  )
```

```
  return(combined_plot)
}

# 4. EXECUTION
cat("================================================================\n"
)
cat("          GENERATING DTR MONTHLY TREND DETAIL PLOT\n")
cat("================================================================\n\
n")

# Generate and print the combined plot
plot_dtr_monthly <- create_dtr_monthly_plots(df_dtr)
print(plot_dtr_monthly)

# ----------------------------------------------------------------------
# END OF SCRIPT
# ----------------------------------------------------------------------
cat("\n\n--- INSTRUCTIONS FOR VIEWING PLOT ---\n")
cat("A single 4x3 multi-panel plot summarizing the monthly DTR trends and
forecasts has been generated.\n")
```

3. **R Script to analyse Urban Area Expansion**

```
# R Script for Analyzing Urbanization Growth in Kolkata
# This revised script excludes the first two data points (2001 and 2002)
# to focus on the subsequent, more consistent growth trend (2003
onwards).

# ----------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ----------------------------------------------------------------------
library(dplyr)
library(ggplot2)

file_urban <- 'Urban Area.csv'
df_urban <- read.csv(file_urban, stringsAsFactors = FALSE)

# ----------------------------------------------------------------------
# 2. DATA PREPARATION AND FILTERING
# ----------------------------------------------------------------------

# Rename columns for clarity and consistency
colnames(df_urban)[1] <- "Date_String"
colnames(df_urban)[2] <- "Urban_Area_km2"

# Convert Date_String to Date object and extract the year
df_urban$Date <- as.Date(df_urban$Date_String, format = '%b %d, %Y')
df_urban$Year <- as.numeric(format(df_urban$Date, "%Y"))

# Ensure Urban_Area_km2 is numeric and filter out missing data
df_urban$Urban_Area_km2 <- as.numeric(df_urban$Urban_Area_km2)
df_urban <- df_urban[!is.na(df_urban$Urban_Area_km2) &
!is.na(df_urban$Year), ]

# CRITICAL MODIFICATION: Exclude the first two rows (2001 and 2002) as
requested.
# This ensures the trend analysis focuses on the stable growth pattern
from 2003 onwards.
df_urban <- df_urban %>%
  slice(3:n())

# ----------------------------------------------------------------------
```

```
# 3. ANALYSIS: URBAN GROWTH TREND
# ------------------------------------------------------------------

# Run linear regression (Urban Area vs. Year) to quantify growth rate
urban_model <- lm(Urban_Area_km2 ~ Year, data = df_urban)
summary_urban <- summary(urban_model)

# Extract key statistics
slope <- coef(urban_model)["Year"]
p_value <- summary_urban$coefficients["Year", "Pr(>|t|)"]
r_squared <- summary_urban$r.squared

cat("--- Kolkata Urbanization Trend Analysis (Filtered: 2003-2023) ---
\n")
cat(sprintf("Time Period: %d - %d\n", min(df_urban$Year),
max(df_urban$Year)))
cat(sprintf("Linear Growth Rate (Change per Year): %.2f km²/year\n",
slope))
cat(sprintf("P-value (Significance): %.5f\n", p_value))
cat(sprintf("R-squared: %.3f\n\n", r_squared))
cat(sprintf("Start Area (Year %d): %.2f km²\n", min(df_urban$Year),
df_urban$Urban_Area_km2[df_urban$Year == min(df_urban$Year)]))
cat(sprintf("End Area (Year %d): %.2f km²\n", max(df_urban$Year),
df_urban$Urban_Area_km2[df_urban$Year == max(df_urban$Year)]))


# ------------------------------------------------------------------
# 4. VISUALIZATION: TIME SERIES PLOT
# ------------------------------------------------------------------

# Create the plot
urban_plot <- ggplot(df_urban, aes(x = Year, y = Urban_Area_km2)) +
  # Add the data points
  geom_point(color = "#3498db", size = 3) +
  # Add a line connecting the points
  geom_line(color = "#2c3e50", linewidth = 0.8) +
  # Add the linear trend line
  geom_smooth(method = "lm", se = TRUE, color = "#e74c3c", fill =
"#fbe7e7", linewidth = 1.2) +
  # Set labels and title
  labs(
    title = "Kolkata Urban Area Expansion (2003 - 2023)",
    subtitle = paste("Linear Growth Rate:", round(slope, 2), "km²/year |
R²:", round(r_squared, 3)),
    x = "Year",
    y = expression("Urban Area ("*km^2*")") # Use expression for
superscript
  ) +
  # Apply a clean theme
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5),
    axis.title.y = element_text(margin = margin(r = 15)),
    panel.grid.minor = element_blank()
  )

# Print the plot object to display it
print(urban_plot)

cat("\n\n--- END OF ANALYSIS ---\n")
```

4. **R Script to analyse Land Surface Temperature**

```
# R Script for Land Surface Temperature (LST) Trend Analysis (1990-
Present)
# This script analyzes the raw satellite LST data and generates two
plots:
# 1. A long-term trend plot with LOESS smoothing to show overall change.
# 2. A monthly box plot to visualize seasonal variation and stability.

# ------------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------------

# Load necessary packages.
library(ggplot2)
library(dplyr)
library(stats) # For lm()

# Define the file path for the LST CSV
file_lst <- 'LST.csv'

# Load the dataframe using base R's read.csv
# Skip the first row if it's a header with system:index/Date (common in
GEE exports)
df_lst <- read.csv(file_lst, stringsAsFactors = FALSE)

# ------------------------------------------------------------------------
# 2. DATA PREPARATION
# ------------------------------------------------------------------------

# Rename columns for simpler access, ignoring the .geo column
colnames(df_lst)[2] <- "Date"
colnames(df_lst)[3] <- "LST"

# Convert Date to a Date object and LST to numeric
df_lst$Date <- as.Date(df_lst$Date, format = '%Y-%m-%d')
df_lst$LST <- as.numeric(df_lst$LST)

# Filter out rows with missing or invalid LST values
df_lst <- df_lst[!is.na(df_lst$LST) & !is.na(df_lst$Date), ]

# Extract Year and Month for detailed analysis
df_lst$Year <- as.numeric(format(df_lst$Date, "%Y"))
df_lst$Month <- factor(format(df_lst$Date, "%m"),
                       levels = sprintf("%02d", 1:12),
                       labels = c("Jan", "Feb", "Mar", "Apr", "May",
"Jun",
                                  "Jul", "Aug", "Sep", "Oct", "Nov",
"Dec"))

# Calculate the fractional time for linear trend fitting
df_lst$DecimalYear <- df_lst$Year + (as.numeric(format(df_lst$Date,
"%j")) - 1) / 365.25


# ------------------------------------------------------------------------
# 3. VISUALIZATION 1: LONG-TERM LST TREND (Scatter with Smoothed Line)
# ------------------------------------------------------------------------

plot_long_term_trend <- function(data) {
```

```
  # Calculate the overall linear trend coefficient for the subtitle
  overall_model <- lm(LST ~ DecimalYear, data = data)
  slope <- coef(overall_model)["DecimalYear"]
  trend_label <- paste0("Overall Linear Trend: ", round(slope, 3), "
°C/Year")

  p <- ggplot(data, aes(x = Date, y = LST)) +

    # Layer 1: Individual observations (scatter plot)
    geom_point(color = "#377EB8", size = 1, alpha = 0.4) +

    # Layer 2: LOESS Smoother (Local Trend) - Highlights the shape of the
change
    geom_smooth(method = "loess", span = 0.25, color = "#E41A1C",
linewidth = 1.5, se = FALSE) +

    # Layer 3: Linear Fit (Overall Long-Term Trend) - Shows net change
    geom_smooth(method = "lm", color = "#4DAF4A", linewidth = 1.2,
linetype = "dashed", se = TRUE) +

    # Labels and Theme
    labs(
      title = "Land Surface Temperature (LST) Observations and Long-Term
Trend",
      subtitle = paste0("LOESS Smoother (Red) vs. Linear Fit (Green
Dashed). ", trend_label),
      x = "Date",
      y = "LST (°C)",
      caption = "Source: Satellite Observations (Raw Data)"
    ) +
    theme_minimal(base_size = 14) +
    theme(
      plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
      plot.subtitle = element_text(size = 12, hjust = 0.5, color =
"gray40"),
      axis.title = element_text(face = "bold"),
      panel.grid.minor = element_blank()
    )

  return(p)
}

# ------------------------------------------------------------------------
# 4. VISUALIZATION 2: MONTHLY LST VARIABILITY (Box Plot)
# ------------------------------------------------------------------------

plot_monthly_variability <- function(data) {

  # Calculate mean LST for each month for visualization
  monthly_mean <- data %>%
    group_by(Month) %>%
    summarise(Mean_LST = mean(LST, na.rm = TRUE))

  p <- ggplot(data, aes(x = Month, y = LST, fill = Month)) +

    # Layer 1: Box plots showing distribution (Median, IQR, Outliers)
    geom_boxplot(outlier.shape = 1, outlier.color = "gray50", alpha =
0.8) +

    # Layer 2: Overlay mean LST (Diamond)
    geom_point(data = monthly_mean, aes(y = Mean_LST),
```

```
                shape = 23, fill = "black", color = "white", size = 3) +

    # Optional: Use a color palette optimized for categorical data
    scale_fill_brewer(palette = "Spectral") +

    # Labels and Theme
    labs(
      title = "Seasonal Distribution of Land Surface Temperature (LST)",
      subtitle = "Boxplot showing median, interquartile range (IQR), and
outliers. Black diamond marks the mean LST.",
      x = "Month",
      y = "LST (°C)",
      caption = paste0("Data span: ", min(data$Year), " - ",
max(data$Year))
    ) +
    theme_minimal(base_size = 14) +
    theme(
      plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
      plot.subtitle = element_text(size = 12, hjust = 0.5, color =
"gray40"),
      axis.title = element_text(face = "bold"),
      legend.position = "none" # Hide legend since colors map directly to
the X-axis
    )

  return(p)
}


# ------------------------------------------------------------------------
# 5. EXECUTION
# ------------------------------------------------------------------------
cat("====================================================================\n"
)
cat("          GENERATING LST LONG-TERM TREND PLOT\n")
cat("====================================================================\n\
n")

# Generate and print Plot 1
lst_trend_plot <- plot_long_term_trend(df_lst)
print(lst_trend_plot)

cat("\n\n====================================================================
=\n")
cat("          GENERATING LST MONTHLY VARIABILITY PLOT\n")
cat("====================================================================\n\
n")

# Generate and print Plot 2
lst_monthly_plot <- plot_monthly_variability(df_lst)
print(lst_monthly_plot)

# ------------------------------------------------------------------------
# END OF SCRIPT
# ------------------------------------------------------------------------
cat("\n\n--- ANALYSIS SUMMARY ---\n")
cat("Two LST plots have been generated:\n")
cat("1. The Long-Term Trend Plot visualizes the overall temperature
change with smooth (LOESS) and linear trend lines.\n")
cat("2. The Monthly Variability Plot uses boxplots to show the seasonal
pattern and data spread for each month.\n")
```

5. **R Script to analyse UHI**

```
# R Script for Analyzing Urban Heat Island (UHI) Intensity in Kolkata
# This script calculates the long-term trend in UHI intensity and
generates
# a time series plot to visualize the change over the years.

# ------------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------------
library(dplyr)
library(ggplot2)

file_uhi <- 'UHI.csv'
df_uhi <- read.csv(file_uhi, stringsAsFactors = FALSE)

# ------------------------------------------------------------------------
# 2. DATA PREPARATION
# ------------------------------------------------------------------------

# Rename columns for clarity and consistency
colnames(df_uhi)[2] <- "UHI_Intensity_C"
colnames(df_uhi)[3] <- "Year"

# Ensure UHI_Intensity_C and Year are numeric and filter out missing data
df_uhi$UHI_Intensity_C <- as.numeric(df_uhi$UHI_Intensity_C)
df_uhi$Year <- as.integer(df_uhi$Year)

df_uhi <- df_uhi[!is.na(df_uhi$UHI_Intensity_C) & !is.na(df_uhi$Year), ]

# Note: UHI intensity is often negative in this type of analysis,
# indicating the urban area is *cooler* than the rural reference (or
vice-versa),
# depending on the calculation method. The key is the *change* in the
value over time.

# ------------------------------------------------------------------------
# 3. ANALYSIS: UHI INTENSITY TREND
# ------------------------------------------------------------------------

# Run linear regression (UHI Intensity vs. Year) to quantify the trend
rate
uhi_model <- lm(UHI_Intensity_C ~ Year, data = df_uhi)
summary_uhi <- summary(uhi_model)

# Extract key statistics
slope <- coef(uhi_model)["Year"]
p_value <- summary_uhi$coefficients["Year", "Pr(>|t|)"]
r_squared <- summary_uhi$r.squared

cat("--- Kolkata UHI Intensity Trend Analysis ---\n")
cat(sprintf("Time Period: %d - %d\n", min(df_uhi$Year),
max(df_uhi$Year)))
cat("The slope indicates the annual change in UHI intensity (Urban LST -
Rural LST).\n")

# Interpret the slope
if (slope > 0) {
  cat(sprintf("Linear Trend Rate: +%.4f °C/year (UHI is
strengthening)\n", slope))
} else {
```

```
  cat(sprintf("Linear Trend Rate: %.4f °C/year (UHI is weakening)\n",
slope))
}

cat(sprintf("P-value (Significance): %.5f\n", p_value))
cat(sprintf("R-squared: %.3f\n\n", r_squared))
cat(sprintf("Initial UHI Intensity (Year %d): %.2f °C\n",
min(df_uhi$Year), df_uhi$UHI_Intensity_C[df_uhi$Year ==
min(df_uhi$Year)]))
cat(sprintf("Final UHI Intensity (Year %d): %.2f °C\n", max(df_uhi$Year),
df_uhi$UHI_Intensity_C[df_uhi$Year == max(df_uhi$Year)]))


# ------------------------------------------------------------------------
# 4. VISUALIZATION: TIME SERIES PLOT
# ------------------------------------------------------------------------

# Create the plot
uhi_plot <- ggplot(df_uhi, aes(x = Year, y = UHI_Intensity_C)) +
  # Add the data points
  geom_point(color = "#e67e22", size = 3) +
  # Add a line connecting the points
  geom_line(color = "#d35400", linewidth = 0.8) +
  # Add the linear trend line
  geom_smooth(method = "lm", se = TRUE, color = "#2980b9", fill =
"#d6eaf8", linewidth = 1.2) +
  # Add a zero line for reference
  geom_hline(yintercept = 0, linetype = "dashed", color = "gray50",
linewidth = 0.6) +
  # Set labels and title
  labs(
    title = "Kolkata Annual Urban Heat Island (UHI) Intensity",
    subtitle = paste("Annual Trend:", round(slope, 4), "°C/year | R²:",
round(r_squared, 3)),
    x = "Year",
    y = expression("UHI Intensity (°C)") # Use expression for superscript
  ) +
  # Apply a clean theme
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5),
    axis.title.y = element_text(margin = margin(r = 15)),
    panel.grid.minor = element_blank()
  )

# Print the plot object to display it
print(uhi_plot)

cat("\n\n--- END OF ANALYSIS ---\n")
```

6. R Script to analyse Rainfall

```
# R Script for Rainfall Volatility, Trend, and Extreme Events Analysis with
ENSO Context

# ------------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------------
library(dplyr)
library(ggplot2)
library(tidyr)
library(purrr)
```

```
library(mgcv)
library(zoo) # For the Moving Average (rollmean)
library(scales) # For percent labels in Goal 4 plot

# Target year is only used for Goal 4 model extrapolation
TARGET_YEAR <- 2030

# Load the data files (ENSURE THESE FILES ARE IN THE WORKING DIRECTORY)
df_annual <- read.csv('Annual Rainfall Summary.csv', stringsAsFactors =
FALSE)
df_monthly <- read.csv('Monthly Rainfall Summary.csv', stringsAsFactors =
FALSE)

# Define the order of the months for correct plotting sequence
month_order <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")

# --- Define ENSO Events (Major El Niño and La Niña years, 1990-2024) ---
enso_events <- data.frame(
  Year = 1990:2024,
  ENSO_Phase = 'Neutral'
)

el_nino_years <- c(1991, 1994, 1997, 2002, 2004, 2006, 2009, 2015, 2018,
2023)
la_nina_years <- c(1998, 1999, 2007, 2010, 2011, 2016, 2020, 2021, 2022)

enso_events <- enso_events %>%
  mutate(
    ENSO_Phase = case_when(
      Year %in% el_nino_years ~ 'El Niño',
      Year %in% la_nina_years ~ 'La Niña',
      TRUE ~ 'Neutral'
    )
  )

# ----------------------------------------------------------------------
# 2. DATA PREPARATION (WITH 5-YEAR MOVING AVERAGE AND ENSO JOIN)
# ----------------------------------------------------------------------

# --- Annual Data Prep (Smoothing and ENSO Join) ---
df_annual <- df_annual %>%
  mutate(Year = as.integer(Year)) %>%
  select(Year,  Annual_Total_Rainfall_mm,  Annual_Mean_Daily_Rainfall_mm)
%>%
  left_join(enso_events, by = "Year") %>% # JOIN ENSO
  arrange(Year) %>%
  # Calculate 5-year Moving Average (MA)
  mutate(Annual_Total_Rainfall_5yr_MA                                    =
rollmean(Annual_Total_Rainfall_mm, k = 5, fill = NA, align = "right"))

# --- Monthly Data Prep (Smoothing and ENSO Join) ---
df_monthly <- df_monthly %>%
  mutate(Year = as.integer(Year)) %>%
  # Convert Month_Name to factor for correct order
  mutate(Month_Name = factor(Month_Name, levels = month_order)) %>%
  rename(Mean_Daily_Rainfall_mm = Mean_Daily_Rainfall_mm) %>%
  filter(!is.na(Month_Name)) %>%
  left_join(enso_events, by = "Year") %>% # JOIN ENSO
  arrange(Month_Name, Year) %>%
  group_by(Month_Name) %>%
```

```
  # Calculate 5-year Moving Average (MA) for each month
  mutate(Mean_Daily_Rainfall_5yr_MA = rollmean(Mean_Daily_Rainfall_mm, k =
5, fill = NA, align = "right")) %>%
  ungroup()

# --- Calculate Extreme Threshold (Needed for Goals 3 and 4) ---
extreme_threshold <- quantile(df_monthly$Mean_Daily_Rainfall_mm, probs =
0.90, na.rm = TRUE)

# ------------------------------------------------------------------------
# Goal 1: Annual Total Rainfall (With Loess Curve, 5-Year MA, and ENSO
Highlight) - REVISED CAPTION
# ------------------------------------------------------------------------

cat("\n--- 1. Annual Total Rainfall (Loess, 5-Year MA, and ENSO Highlight)
---\n")

plot_annual_total   <-   ggplot(df_annual,   aes(x   =   Year,   y   =
Annual_Total_Rainfall_mm)) +

  # Highlight background regions for La Niña and El Niño (visual context)
  geom_rect(data = df_annual %>% filter(ENSO_Phase != 'Neutral'),
            aes(xmin = Year - 0.5, xmax = Year + 0.5, ymin = -Inf, ymax =
Inf, fill = ENSO_Phase),
            alpha = 0.1, show.legend = FALSE) +

  # 1. Scattered Points (Raw Data) - Colored by ENSO
  geom_point(aes(color = ENSO_Phase), size = 3, alpha = 0.8) +

  # 2. Loess Smoothing Curve (Non-linear Trend) - Black Line
  geom_smooth(method = "loess", span = 0.7, se = FALSE, color = "black",
linewidth = 1.2) +

  # 3. 5-Year Moving Average Line (Added back) - Dark Red Line
  geom_line(aes(y  =  Annual_Total_Rainfall_5yr_MA),  color  =  "#8B0000",
linewidth = 1) +

  # Define colors for the ENSO phases
  scale_color_manual(values  =  c("El  Niño"  =  "#e74c3c",  "La  Niña"  =
"#3498db", "Neutral" = "#7f8c8d"), name = "ENSO Phase") +
  scale_fill_manual(values  =  c("El  Niño"  =  "#e74c3c",  "La  Niña"  =
"#3498db", "Neutral" = NA)) +

  labs(
    title = "Annual Total Rainfall (1990 - 2024) with Loess Trend and 5-
Year MA",
    subtitle = "ENSO phase colors points. Background highlight shows La
Niña (blue) and El Niño (red) years.",
    x = "Year",
    y = "Total Rainfall (mm)",
    # MOVED line description to caption
    caption = "Trend Lines: Black line shows the Loess trend. Dark red line
shows the 5-year moving average."
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "top",
        # Optionally left-align the caption if preferred
        plot.caption.position = "panel",
        plot.caption = element_text(hjust = 0))

print(plot_annual_total)
```

```
# ------------------------------------------------------------------------
# Goal 2: Monthly Mean Daily Rainfall (Multi-paneled with 5-Year Moving
Average)
# ------------------------------------------------------------------------

cat("\n--- 2. Monthly Mean Daily Rainfall (Smoothed Multi-panel) ---\n")

plot_monthly_simple_multipanel <- ggplot(df_monthly, aes(x = Year, y =
Mean_Daily_Rainfall_mm)) +
  # 1. Scattered Points (Raw Data)
  geom_point(alpha = 0.4, size = 1.5, color = "#2ecc71") +

  # 2. Moving Average Line (Smoothed Trend)
  geom_line(aes(y = Mean_Daily_Rainfall_5yr_MA), color = "#f39c12", size =
1, alpha = 0.8) +

  facet_wrap(~ Month_Name, scales = "free_y", ncol = 4) +

  labs(
    title = "Monthly Mean Daily Rainfall Over Time (1990 - 2024)",
    subtitle = "Orange line shows the 5-year moving average for each month
to indicate change.",
    x = "Year",
    y = "Mean Daily Rainfall (mm)"
  ) +
  theme_light(base_size = 12) +
  theme(plot.title = element_text(face = "bold"),
        strip.text = element_text(face = "bold"))

print(plot_monthly_simple_multipanel)

# ------------------------------------------------------------------------
# Goal 3: Extreme Rainfall Months (Table and Heat Map Plot with ENSO
Highlight)
# ------------------------------------------------------------------------

cat("\n--- 3. Extreme Rainfall Months (Table and Heat Map Plot with ENSO
Highlight) ---\n")
cat(paste0("Extreme Rainfall Threshold (90th percentile of monthly mean
daily rain): ", round(extreme_threshold, 2), " mm/day\n"))

# Create the data frame for extreme events (the table)
df_extreme_plot <- df_monthly %>%
  filter(Mean_Daily_Rainfall_mm >= extreme_threshold) %>%
  mutate(Mean_Daily_Rainfall_mm = round(Mean_Daily_Rainfall_mm, 2)) %>%
  select(Year, Month_Name, Mean_Daily_Rainfall_mm, ENSO_Phase) %>% #
Including ENSO_Phase
  arrange(Year, factor(Month_Name, levels = month_order))

cat("\nExtreme Rainfall Months Table:\n")
print(df_extreme_plot)
write.csv(df_extreme_plot, 'Extreme Rainfall Months Table.csv', row.names
= FALSE)


# --- Plotting the Extreme Months as a Heat Map ---
plot_extreme_months_heatmap <- ggplot(df_extreme_plot, aes(x = Year, y =
Month_Name, fill = Mean_Daily_Rainfall_mm)) +

  # Add the color fill for rainfall intensity
```

```
  geom_tile(color = "white", size = 0.5) +

  # Highlight the tile borders based on ENSO phase
  geom_tile(aes(color = ENSO_Phase), size = 1.5, fill = NA) + # Highlight
tile border

  # Add the rainfall value as text on each tile
  geom_text(aes(label = Mean_Daily_Rainfall_mm), color = "black", size =
3) +

  # Define the color scale for the fill (Rainfall Intensity)
  scale_fill_gradient(low = "#fee8c8", high = "#e34a33", name = "Rainfall
(mm/day)") +

  # Define the colors for the border (ENSO Phase)
  scale_color_manual(values = c("El Niño" = "#e74c3c", "La Niña" =
"#3498db", "Neutral" = "gray50"), name = "ENSO Phase Border") +

  scale_y_discrete(limits = rev(month_order)) +
  scale_x_continuous(breaks = unique(df_extreme_plot$Year)) +

  labs(
    title = "Extreme Rainfall Months Highlighting ENSO Influence",
    subtitle = "Border color indicates the ENSO phase during the extreme
event.",
    x = "Year",
    y = "Month"
  ) +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1),
        plot.title = element_text(face = "bold"),
        legend.position = "bottom")

print(plot_extreme_months_heatmap)

# ------------------------------------------------------------------------
# Goal 4: Monthly Extreme Event Probability Trend (Caterpillar Plot)
# ------------------------------------------------------------------------

cat("\n--- 4. Monthly Extreme Event Probability Trend (Caterpillar Plot)
---\n")

# --- Model Fitting ---
df_extreme_monthly <- df_monthly %>%
  mutate(Is_Extreme = Mean_Daily_Rainfall_mm >= extreme_threshold) %>%
  mutate(Extreme_Month_Index = as.numeric(Is_Extreme))

monthly_extreme_trend_analysis_glm <- df_extreme_monthly %>%
  group_by(Month_Name) %>%
  nest() %>%
  mutate(
    model = map(data, ~ glm(Extreme_Month_Index ~ Year, data = .x, family
= "binomial")),
    log_odds_slope = map_dbl(model, ~ coef(.)["Year"]),
    prediction_2030_prob = map_dbl(model, ~ predict(., newdata =
data.frame(Year = TARGET_YEAR), type = "response"))
  ) %>%
  unnest(cols = c(log_odds_slope, prediction_2030_prob)) %>%
  select(Month_Name, Log_Odds_Slope = log_odds_slope,
Prediction_2030_Probability = prediction_2030_prob)
```

```
cat("\nMonthly Extreme Event Probability Trend Summary (Logistic GLM):\n")
print(monthly_extreme_trend_analysis_glm)

# --- Caterpillar Plot Visualization ---
plot_extreme_probability_caterpillar <-
ggplot(monthly_extreme_trend_analysis_glm,
                                          aes(x = Prediction_2030_Probability,
                                              y = factor(Month_Name, levels = rev(month_order)))) +

  # 1. Trend Direction Line (from 0 probability to 2030 prediction)
  geom_segment(aes(x = 0, xend = Prediction_2030_Probability,
                   y = factor(Month_Name, levels = rev(month_order)),
                   yend = factor(Month_Name, levels = rev(month_order)),
                   color = Log_Odds_Slope),
               linewidth = 1.5, alpha = 0.7) +

  # 2. Prediction Point (2030 Probability)
  geom_point(aes(color = Log_Odds_Slope), size = 5) +

  # Add a vertical line at the long-term (unconditional) probability (1/12 or ~0.083)
  geom_vline(xintercept = 1/12, linetype = "dashed", color = "gray50") +
  geom_text(aes(x = 1/12, y = 1), label = "Long-Term Avg (1/12)", hjust = -0.1, vjust = 1.2, color = "gray50", size = 3) +

  # Color scale based on the slope (Log Odds Slope)
  scale_color_gradient2(
    low = "#3498db",
    mid = "gray",
    high = "#e74c3c",
    midpoint = 0,
    name = "Trend (Log Odds Slope)",
    # Explicitly define breaks based on min/max slope and 0 (Neutral)
    breaks = c(min(monthly_extreme_trend_analysis_glm$Log_Odds_Slope), 0, max(monthly_extreme_trend_analysis_glm$Log_Odds_Slope)),
    labels = c("Decreasing", "Neutral", "Increasing")
  ) +

  scale_x_continuous(labels = scales::percent) +

  labs(
    title = "Monthly Extreme Rainfall Probability Trend Summary",
    subtitle = paste0("Predicted probability of an extreme event in ", TARGET_YEAR, " (X-axis) and trend direction (Color)."),
    x = "Predicted Probability of Extreme Event",
    y = "Month"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom",
        legend.title = element_text(face = "bold"))

print(plot_extreme_probability_caterpillar)
```

7. R Script to analyse Rainfall (Future Projections)

```
# ------------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------------
library(dplyr)
library(ggplot2)
library(tidyr)
```

```
library(RColorBrewer)

# Load the projection data file
df_projection <- read.csv('Temperature Humidity Projection.csv',
stringsAsFactors = FALSE)

# ------------------------------------------------------------------------
# 2. DATA CLEANING AND PREPARATION
# ------------------------------------------------------------------------

df_cleaned <- df_projection %>%
  # Convert Year to integer
  mutate(Year = as.integer(Year)) %>%
  # Filter out years with 0.0 values (often representing the
historical/initial period)
  # Keeping only years with actual projected data (Mean_Temperature_C >
0.0)
  filter(Mean_Temperature_C > 0.0) %>%
  # Rename columns for simpler plotting
  rename(
    Temperature_C = Mean_Temperature_C,
    Specific_Humidity_g_kg = Mean_Specific_Humidity_g_kg
  )

# Check the unique SSP scenarios for legend
scenarios <- unique(df_cleaned$SSP)
cat("Found the following climate scenarios (SSP) in the data:\n")
print(scenarios)

# Define a color palette for the scenarios
# Using Paired from RColorBrewer for high contrast
scenario_colors <- brewer.pal(n = length(scenarios), name = "Set1")
names(scenario_colors) <- scenarios

# ------------------------------------------------------------------------
# 3. VISUALISATION: MEAN TEMPERATURE TREND
# ------------------------------------------------------------------------

plot_temperature <- ggplot(df_cleaned, aes(x = Year, y = Temperature_C,
color = SSP)) +

  # Smooth lines to show the trend clearly
  geom_smooth(method = "loess", se = FALSE, linewidth = 1.5, alpha = 0.8)
+

  # Points for annual data, slightly faded
  geom_point(alpha = 0.3, size = 1) +

  # Customize titles and labels
  labs(
    title = "Projected Annual Mean Temperature Trend",
    subtitle = "Comparison across Shared Socioeconomic Pathways (SSPs)",
    x = "Year",
    y = expression("Mean Temperature ("*~degree*C*")"),
    color = "Scenario (SSP)"
  ) +

  # Use the defined color palette
  scale_color_manual(values = scenario_colors) +

  # Attractive, clean theme
```

```
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", size = 18, color = "#2c3e50"),
    plot.subtitle = element_text(size = 12, color = "#7f8c8d"),
    axis.title.y = element_text(margin = margin(r = 15)),
    axis.title.x = element_text(margin = margin(t = 10)),
    legend.position = "bottom",
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "#ecf0f1", color = NA)
  )

print(plot_temperature)

# ------------------------------------------------------------------------
# 4. VISUALISATION: MEAN SPECIFIC HUMIDITY TREND
# ------------------------------------------------------------------------

plot_humidity <- ggplot(df_cleaned, aes(x = Year, y =
Specific_Humidity_g_kg, color = SSP)) +

  # Smooth lines to show the trend clearly
  geom_smooth(method = "loess", se = FALSE, linewidth = 1.5, alpha = 0.8)
+

  # Points for annual data, slightly faded
  geom_point(alpha = 0.3, size = 1) +

  # Customize titles and labels
  labs(
    title = "Projected Annual Mean Specific Humidity Trend",
    subtitle = "Specific humidity is a measure of moisture content (g of
water vapor/kg of air)",
    x = "Year",
    y = "Mean Specific Humidity (g/kg)",
    color = "Scenario (SSP)"
  ) +

  # Use the defined color palette
  scale_color_manual(values = scenario_colors) +

  # Attractive, clean theme
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", size = 18, color = "#2c3e50"),
    plot.subtitle = element_text(size = 12, color = "#7f8c8d"),
    axis.title.y = element_text(margin = margin(r = 15)),
    axis.title.x = element_text(margin = margin(t = 10)),
    legend.position = "bottom",
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "#ecf0f1", color = NA)
  )

print(plot_humidity)
```

8. **R Script to analyse Rainfall Projection**

```
# R Script for Visualising Climate Projection Trends (Rainfall) and Extreme
Analysis

# ------------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------------
```

```
library(dplyr)
library(ggplot2)
library(RColorBrewer)

# Load the projection data file
df_rainfall <- read.csv('Rainfall Projection.csv', stringsAsFactors =
FALSE)

# ---------------------------------------------------------------------
# 2. DATA CLEANING AND PREPARATION
# ---------------------------------------------------------------------

df_cleaned <- df_rainfall %>%
  # Convert Year to integer
  mutate(Year = as.integer(Year)) %>%
  # Filter out years with 0.0 values (often representing the
historical/initial period)
  filter(Mean_Annual_Rainfall_mm > 0.0)

# Check the unique SSP scenarios for legend
scenarios <- unique(df_cleaned$SSP)
cat("Found the following climate scenarios (SSP) in the data:\n")
print(scenarios)

# Define a color palette for the scenarios
scenario_colors <- brewer.pal(n = length(scenarios), name = "Dark2")
names(scenario_colors) <- scenarios

# ---------------------------------------------------------------------
# 3. VISUALISATION: MEAN ANNUAL RAINFALL TREND
# ---------------------------------------------------------------------

plot_rainfall_trend <- ggplot(df_cleaned, aes(x = Year, y =
Mean_Annual_Rainfall_mm, color = SSP)) +

  # Smooth lines to show the long-term trend clearly
  geom_smooth(method = "loess", se = FALSE, linewidth = 1.5, alpha = 0.8)
+

  # Points for annual data, slightly faded
  geom_point(alpha = 0.3, size = 1) +

  # Customize titles and labels
  labs(
    title = "Projected Mean Annual Rainfall Trend",
    subtitle = "Annual Rainfall Projections across Shared Socioeconomic
Pathways (SSPs)",
    x = "Year",
    y = "Mean Annual Rainfall (mm)",
    color = "Scenario (SSP)"
  ) +

  # Use the defined color palette
  scale_color_manual(values = scenario_colors) +

  # Attractive, clean theme
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", size = 18, color = "#1b9e77"),
    plot.subtitle = element_text(size = 12, color = "#7f8c8d"),
    legend.position = "bottom",
```

```
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "#ecf0f1", color = NA)
  )

print(plot_rainfall_trend)

# ------------------------------------------------------------------
# 4. ANALYSIS & VISUALISATION: CHANCES OF EXTREME RAINFALL
# ------------------------------------------------------------------

# Box plot is excellent for visualizing the distribution, variance, and
outliers (extremes)
plot_extreme_rainfall <- ggplot(df_cleaned, aes(x = SSP, y =
Mean_Annual_Rainfall_mm, fill = SSP)) +

  # Create the boxplot
  geom_boxplot(alpha = 0.7, color = "black", linewidth = 0.8) +

  # Overlay individual points (optional, but shows density)
  geom_jitter(color = "black", size = 0.8, alpha = 0.4, width = 0.2) +

  # Customize titles and labels
  labs(
    title = "Rainfall Distribution and Potential for Extreme Events",
    subtitle = "Comparison of Annual Rainfall Variance across SSP
Scenarios",
    x = "Scenario (SSP)",
    y = "Mean Annual Rainfall (mm)",
    fill = "Scenario (SSP)"
  ) +

  # Use the defined color palette
  scale_fill_manual(values = scenario_colors) +

  # Coordinates flip for better label reading if many scenarios
  coord_flip() +

  # Attractive, clean theme
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", size = 18, color = "#d95f02"),
    plot.subtitle = element_text(size = 12, color = "#7f8c8d"),
    legend.position = "none", # Hide redundant legend
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "#ecf0f1", color = NA)
  )

print(plot_extreme_rainfall)
```

9. **R Script to for Monthly Climate Stripe Generation**

```
# R Script to Generate Monthly Climate Stripes Visualization using Base R
and ggplot2

# ------------------------------------------------------------------
# 1. SETUP AND DATA LOADING
# ------------------------------------------------------------------
# Only load ggplot2 for the visualization. Base R handles data manipulation.
library(ggplot2)

# Load the monthly mean temperature data
df <- read.csv('Monthly Mean Temp_AH.csv', stringsAsFactors = FALSE)
```

```
# ------------------------------------------------------------------------
# 2. DATA PREPARATION AND ANOMALY CALCULATION (Base R)
# ------------------------------------------------------------------------

# --- A. Rename and Prepare Columns ---
# Rename columns for clarity (using indexing/matching to avoid dplyr)
names(df)[names(df) == "year"] <- "Year"
names(df)[names(df) == "month"] <- "Month_Num"
names(df)[names(df) == "mean_temp_c"] <- "Temp_C"

# Ensure year is treated as an integer
df$Year <- as.integer(df$Year)

# Define the order of the months
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")

# Create the Month Name factor in the correct chronological order for
faceting
df$Month_Name <- factor(month.abb[df$Month_Num], levels = month_levels)

# Sort the data by Month and Year (important for geom_tile rendering order)
df <- df[order(df$Month_Num, df$Year), ]

# --- B. Calculate Monthly Mean Baseline ---
# Calculate the mean temperature for each month across all years using
aggregate()
monthly_means <- aggregate(Temp_C ~ Month_Name, data = df, FUN = mean,
na.action = na.omit)
names(monthly_means)[names(monthly_means)      ==      "Temp_C"]      <-
"Monthly_Mean_Baseline"

# Merge the calculated baseline back into the main dataframe
df <- merge(df, monthly_means, by = "Month_Name", all.x = TRUE)

# --- C. Calculate the Temperature Anomaly ---
# Anomaly is the difference from the long-term monthly mean
df$Anomaly <- df$Temp_C - df$Monthly_Mean_Baseline

# Determine the absolute maximum anomaly for a symmetric color scale
max_abs_anomaly <- max(abs(df$Anomaly), na.rm = TRUE)

# ------------------------------------------------------------------------
# 3. VISUALIZATION: CLIMATE STRIPES PLOT (ggplot2)
# ------------------------------------------------------------------------

plot_stripes <- ggplot(df, aes(x = Year, y = 1, fill = Anomaly)) +

  # Use geom_tile to draw the colored rectangles (stripes)
  geom_tile(width = 1, height = 1) +

  # Facet the plot by Month_Name
  # 'free_x' allows the x-axis to be shared across all months, which is
desired here
  facet_wrap(~Month_Name, ncol = 3, strip.position = "top") +

  # Apply the Blue-to-Red color gradient centered at the baseline (0)
  scale_fill_gradient2(
    low = "#08519c", # Deeper Blue
    mid = "white",
```

```
    high = "#b10026", # Deeper Red
    midpoint = 0,
    limit = c(-max_abs_anomaly, max_abs_anomaly),
    space = "Lab",
    name = "Temperature Anomaly (°C)\n(vs. Monthly Long-Term Mean)"
  ) +

  # Set up a clean, minimal aesthetic
  theme_minimal(base_size = 12) +

  # Customize titles and remove unnecessary axis elements for a classic
stripe look
  labs(
    title = "Monthly Climate Stripes: Visualizing Temperature Trends",
    subtitle = paste("Monthly Mean Temperature Anomaly from", min(df$Year),
"to", max(df$Year)),
    x = "Year",
    y = ""
  ) +

  theme(
    # Remove y-axis elements entirely as they are not informative in a
stripe plot
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),

    # Minimize spacing between facets/stripes for the full effect
    panel.spacing.x = unit(0.01, "lines"),
    panel.spacing.y = unit(0.5, "lines"),
    panel.grid = element_blank(),

    # Title formatting
    plot.title = element_text(face = "bold", size = 18, hjust = 0.5, color
= "#2c3e50"),
    plot.subtitle = element_text(hjust = 0.5, color = "#7f8c8d"),

    # Ensure facets (month names) are displayed clearly
    strip.text = element_text(face = "bold"),

    # Legend at the bottom
    legend.position = "bottom",
    legend.title = element_text(face = "bold"),
    legend.key.width = unit(2, "cm")
  ) +

  # Customize x-axis breaks to avoid overcrowding
  scale_x_continuous(breaks = seq(min(df$Year), max(df$Year), by = 10))

# Display the plot
print(plot_stripes)
```