1 Building an Early Warning System for Dengue Outbreaks: Spatiotemporal Predictive Modeling of Dengue Cases in Philippine Cities - Exploratory Data Analysis

1.1 General Resources

1.1.1 Libraries

In [1]:
```
# Data importation and manipulation
import os
import pandas as pd
import numpy as np
```

In [2]:
```
import geopandas as gpd
```

In [3]:
```
# Data visualization, exploratory and result analysis
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns
```

In [4]:
```
# Auxilliaries
from time import time
from datetime import datetime, timedelta
from collections import defaultdict
```

In [5]:
```
import warnings

# Suppress 'deprecated' warnings, added in final run to clean notebook when printing into pdf
warnings.filterwarnings("ignore")
```

1.1.2 References

```
https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html
```

1.1.3 Dataset

In [6]:
```
def import_csv_dataset(file_path, header_ind):
    df = pd.read_csv(file_path, header=header_ind, encoding='latin-1')

    # Check if 'date' column exists and parse it
    if 'date' in df.columns:
        date_formats = ['%Y-%m-%d', '%d/%m/%Y', '%m/%d/%Y']
        parsed = False
        for date_format in date_formats:
            try:
                df['date'] = pd.to_datetime(df['date'], format=date_format)
                parsed = True
                break
            except ValueError:
                continue

        if not parsed:
            raise ValueError("Date format not recognized or incompatible with any provided format.")

    return df
```

In [7]:
```
datas = {
    'dengue' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_dengue_C3.csv",
    'climate' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_climate_weekly_C
    'climate_parameters' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_daily
    'static' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_static_C3.csv",

    'city_coordinates' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_cities_

    'trend_dengue' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_google tren
    'trend_denguesympt' : r"C:\Users\Josh\000 Files\003 Mengg AI\01b 221 (AI)\03 Project\Data C3\ph_google
}
```

In [65]:
```
for name, path in datas.items():
    globals()[f"{name}_df"] = import_csv_dataset(path, 0)
```

In [9]:
```
# print(dengue_df)
# print(climate_df)
# prtin(climate_parameters_df)
# print(hospitals_df)
# print(static_df)
# print(city_coordinates_df)
# print(city_coordinates2_df)
# print(trend_dengue_df)
# print(trend_denguesympt_df)
```

1.2 Philippine Cities - Static Variables

1.2.1 Content

In [10]:
```
city_coordinates_df
```

Out[10]:

	full loc	coordinates	latitude	longitude	loc
0	Bulacan, Philippines	(15.0, 121.083333)	15.000000	121.083333	BULACAN
1	Rizal, Philippines	(14.65, 121.25)	14.650000	121.250000	RIZAL
2	Quezon City, Philippines	(14.6510546, 121.0486254)	14.651055	121.048625	QUEZON

In [11]:
```
static_df
```

Out[11]:

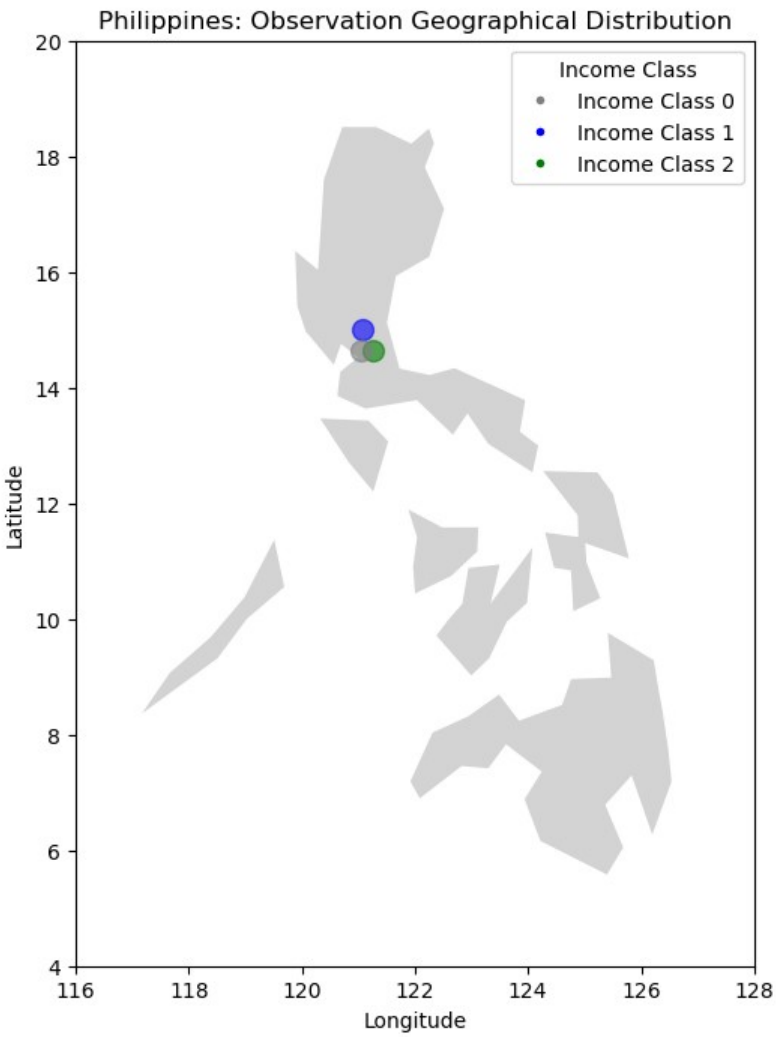
	city	income class	land area	01-Aug-15	01-May-20	closest_gov_hospital	gov_distance	closest_priv_hospital	priv_distance
0	BULACAN	1	2,796.10	3292071	3708890	Gregorio Del Pilar District Hospital	0.260	St Paul Hospital Bulacan Inc	8.760
1	QUEZON	0	9,069.60	2936116	2960048	National Kidney and Transplant Institute	3.417	F Y Manalo Medical Foundation Inc	1.168
2	RIZAL	2	1,191.94	17994	19077	Antipolo City Hospital System Cabading	15.185	Assumption Specialty Hospital And Medical Cent...	24.685

In [12]:
```
static_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   city                   3 non-null      object
 1   income class           3 non-null      int64
 2   land area              3 non-null      object
 3   01-Aug-15              3 non-null      int64
 4   01-May-20              3 non-null      int64
 5   closest_gov_hospital   3 non-null      object
 6   gov_distance           3 non-null      float64
 7   closest_priv_hospital  3 non-null      object
 8   priv_distance          3 non-null      float64
dtypes: float64(2), int64(3), object(4)
memory usage: 348.0+ bytes
```

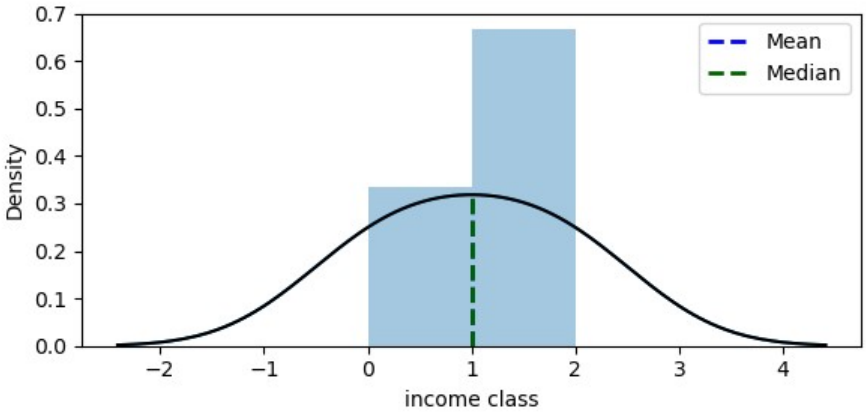
Cities under study

In [13]:
```
# Load the world map
worldmap = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))

# Filter the world map to only show the Philippines
ph_map = worldmap[worldmap['name'] == 'Philippines']

# Define colors for income classes
# colors = {0: 'gray', 1: 'blue', 2: 'green', 3: 'yellow', 4: 'orange', 5: 'red', 6: 'purple'}
colors = {0: 'gray', 1: 'blue', 2: 'green'}

# Create a scatter plot of city coordinates on the map, colored by income class
fig, ax = plt.subplots(figsize=(15, 8))
ph_map.plot(color="lightgrey", ax=ax)

# Plot cities with income class color
for index, row in city_coordinates_df.iterrows():
    city = row['loc']
    if city in static_df['city'].values:
        income_class = static_df[static_df['city'] == city]['income class'].values[0]
        plt.scatter(row['longitude'], row['latitude'],
                    color=colors[income_class], s=100, alpha=0.6, label=f"Income Class {income_class} - {c

# Set axis limits and labels
plt.xlim([116, 128])  # Adjusted for the Philippines
plt.ylim([4, 20])      # Adjusted for the Philippines
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Philippines: Observation Geographical Distribution")

# Create legend
legend_elements = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=colors[i], markersize=5, la
plt.legend(handles=legend_elements, title='Income Class', loc='best')

plt.show()
```



In [14]:
```
def dist_plot(size_x=20, size_y= 15, row=2, col=2, df=None):
    fig = plt.figure(figsize=(size_x, size_y))
    for index, column in enumerate(df.columns):
        plt.subplot(row, col, index + 1)
        feature = df[column]
        sns.distplot(df[column])

        mean_val = feature.mean()
        median_val = feature.median()
        max_density = sns.kdeplot(feature, color='black').get_lines()[0].get_data()[1].max()

        plt.plot([mean_val, mean_val], [0, max_density], color='blue', linestyle='dashed', linewidth=2, la
        plt.plot([median_val, median_val], [0, max_density], color='darkgreen', linestyle='dashed', linewi
    #     plt.title(f'{column.title()}')
        plt.legend()
        plt.tight_layout()

    plt.show()
```

In [15]:
```
dist_plot(size_x=6, size_y= 3, row=1, col=1, df=static_df[['income class']])
```



1.3 Dengue Data

1.3.1 Content

In [16]:
```
dengue_df
```

Out[16]:

	loc	cases	deaths	date	Region	year	month
0	BULACAN	97	0	2016-01-10	REGION III-CENTRAL LUZON	2016	1
1	BULACAN	58	0	2016-01-17	REGION III-CENTRAL LUZON	2016	1
2	BULACAN	76	1	2016-01-24	REGION III-CENTRAL LUZON	2016	1
3	BULACAN	101	0	2016-01-31	REGION III-CENTRAL LUZON	2016	1
4	BULACAN	85	0	2016-02-07	REGION III-CENTRAL LUZON	2016	2
...	...	...	...	...	...	...	...
776	QUEZON CITY	0	26	2020-09-06	NATIONAL CAPITAL REGION	2020	9
777	QUEZON CITY	0	12	2020-09-20	NATIONAL CAPITAL REGION	2020	9
778	QUEZON CITY	0	13	2020-10-04	NATIONAL CAPITAL REGION	2020	10
779	QUEZON CITY	0	22	2020-10-18	NATIONAL CAPITAL REGION	2020	10
780	QUEZON CITY	0	29	2020-11-01	NATIONAL CAPITAL REGION	2020	11

781 rows × 7 columns

In [17]:
```
dengue_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 781 entries, 0 to 780
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   loc     781 non-null    object
 1   cases   781 non-null    int64
 2   deaths  781 non-null    int64
 3   date    781 non-null    datetime64[ns]
 4   Region  781 non-null    object
 5   year    781 non-null    int64
 6   month   781 non-null    int64
dtypes: datetime64[ns](1), int64(4), object(2)
memory usage: 42.8+ KB
```

In [18]:
```
dengue_df.isna().sum()
```

Out[18]:
```
loc       0
cases     0
deaths    0
date      0
Region    0
year      0
month     0
dtype: int64
```

In [19]:
```
dengue_df.describe().T
```

Out[19]:

	count	mean	min	25%	50%	75%	max	std
cases	781.0	120.184379	0.0	38.0	82.0	157.0	812.0	123.259376
deaths	781.0	1.93854	0.0	0.0	0.0	1.0	203.0	13.617723
date	781	2018-07-10 11:48:00.921895168	2016-01-10 00:00:00	2017-04-09 00:00:00	2018-07-08 00:00:00	2019-10-13 00:00:00	2021-01-10 00:00:00	NaN
year	781.0	2018.023047	2016.0	2017.0	2018.0	2019.0	2021.0	1.429356
month	781.0	6.524968	1.0	4.0	7.0	10.0	12.0	3.452751

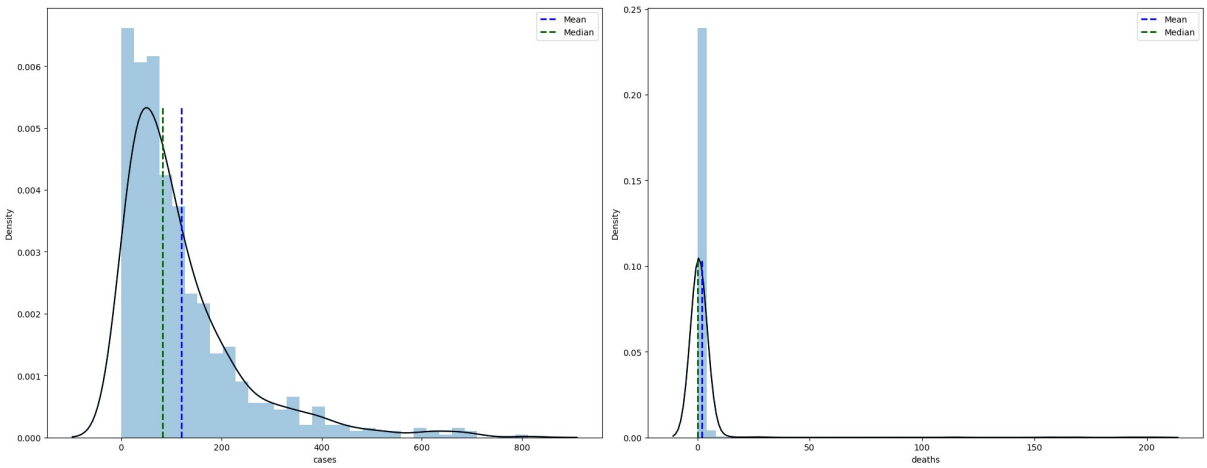
In [20]:
```
print(f"Number of cities: {len(np.unique(dengue_df['loc']))}")
```

```
Number of cities: 3
```

1.3.2 Visualization

In [21]:
```
dengue_cd_df = dengue_df[['cases', 'deaths']]
```

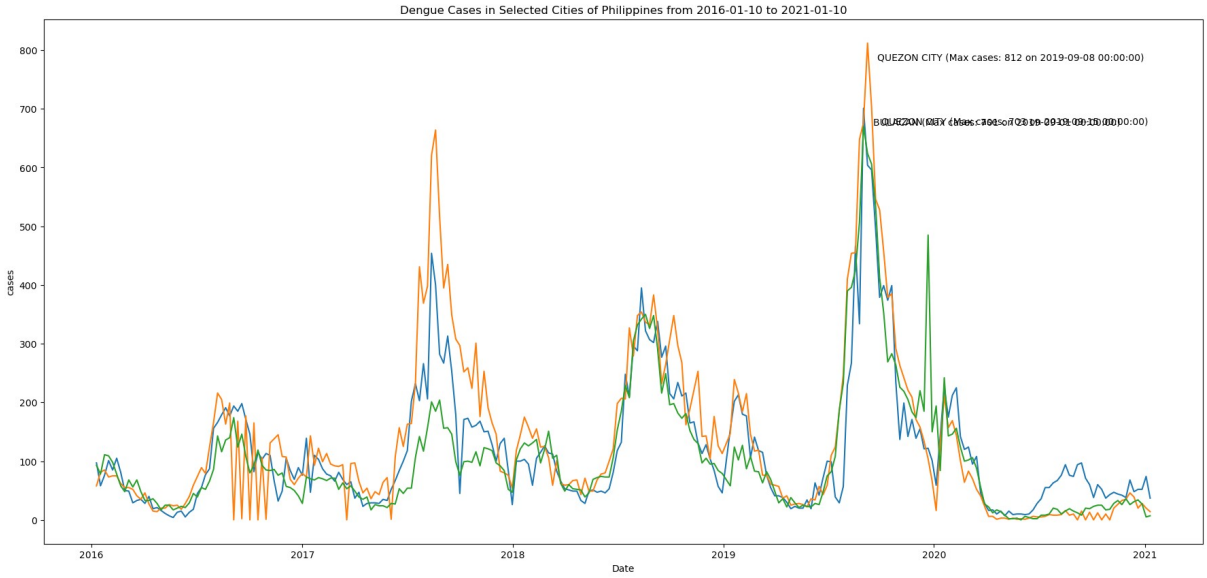
In [22]:
```
dist_plot(size_x=20, size_y= 15, row=2, col=2, df=dengue_cd_df)
```



Cases

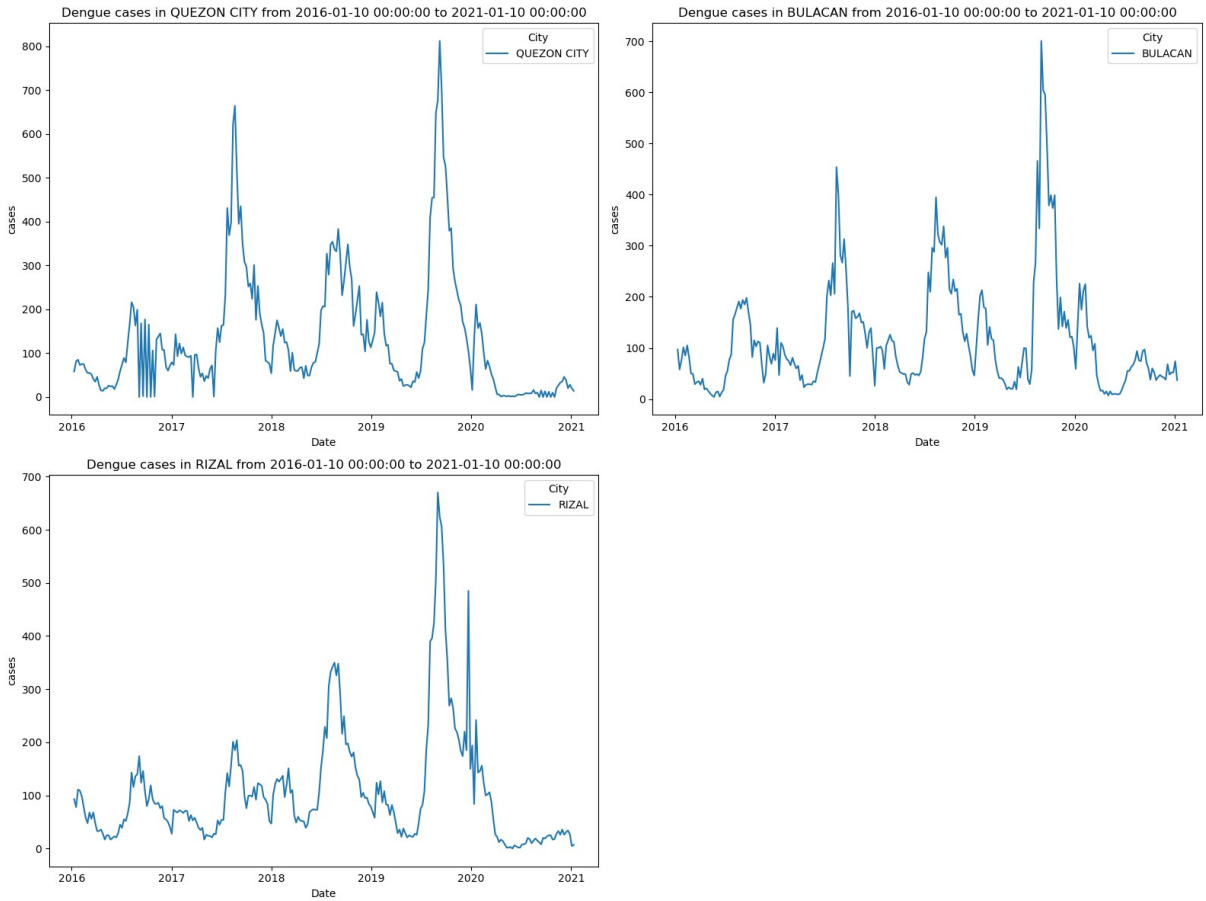
In [23]:
```
def plot_top_(n, variable, df):
    # Format Date Column
    df['date'] = pd.to_datetime(df['date'])
    df_sorted = df.sort_values('date')


    # Top cities
    top_cities = df_sorted.groupby(['loc', 'date'])[variable].sum().nlargest(n).index.tolist()

    # Group by location and plot
    fig, ax = plt.subplots(figsize=(22, 10))
    for name, group in df_sorted.groupby('loc'):
        ax.plot(group['date'], group[variable], label=name)

    ax.set_title(f"Dengue Cases in Selected Cities of Philippines from {df_sorted['date'].min().date()} to
    ax.set_xlabel("Date")
    ax.set_ylabel(f"{variable}")
    # ax.legend(title='City')

    # Annotate
    for loc, date in top_cities:
        max_cases = df_sorted[(df_sorted['loc'] == loc) & (df_sorted['date'] == date)][variable].iloc[0]
        ax.annotate(f"{loc} (Max {variable}: {max_cases} on {date})",
                    xy=(date, max_cases),
                    xytext=(10, -10),
                    textcoords='offset points',
                    ha='left',
                    va='top',
                    fontsize=10,
                    color='black')

    plt.show()
```

In [24]:
```
plot_top_(3, 'cases', dengue_df)
```



In [25]:
```
def separate_plot_top(n, variable, df):
    # Format Date Column
    df['date'] = pd.to_datetime(df['date'])
    df_sorted = df.sort_values('date')

    # Calculate the largest value for each city and sort by it
    top_cities = df_sorted.groupby('loc')[variable].max().nlargest(n).index.tolist()

    num_plots = len(top_cities)
    num_rows = (num_plots + 1) // 2  # Calculate the number of rows required for 2 columns

    fig, axs = plt.subplots(num_rows, 2, figsize=(16, 6 * num_rows))

    for i, loc in enumerate(top_cities):
        city_data = df_sorted[df_sorted['loc'] == loc]
        row = i // 2
        col = i % 2
        ax = axs[row, col] if num_rows > 1 else axs[col]

        ax.plot(city_data['date'], city_data[variable], label=loc)
        ax.set_title(f"Dengue {variable} in {loc} from {city_data['date'].min()} to {city_data['date'].ma
        ax.set_xlabel("Date")
        ax.set_ylabel(f"{variable}")
        ax.legend(title='City')

    # Hide empty subplots
    if num_plots % 2 != 0:
        axs[num_rows - 1, 1].axis('off')

    plt.tight_layout()
    plt.show()

# Example usage:
# df = pd.read_csv('your_data.csv')
# separate_plot_top(10, 'cases', df)
```

In [26]:
```
separate_plot_top(3, 'cases', dengue_df)
```



1.3.3 Stationarity

There are three basic criterion for a time series to understand whether it is stationary series or not.

Statistical properties of time series such as mean, variance should remain constant over time to call time series is stationary

- constant mean
- constant variance
- autocovariance that does not depend on time. autocovariance is covariance between time series and lagged time series.

Terminologies

```
Trend: varying mean over time. We need constant mean for stationary of time series.
Seasonality: variations at specific time. We need constant variations for stationary of time seri
es.
```

Methods:

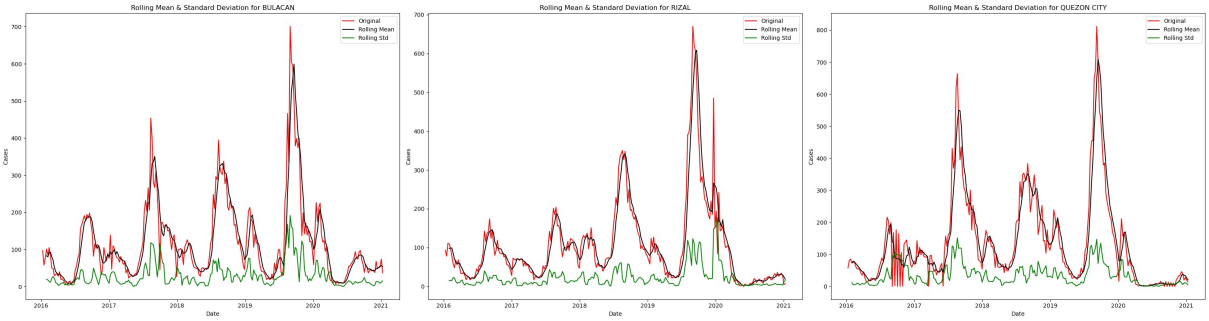
- Plotting Rolling Statistics: We have a window and then we find rolling mean and variance to check stationary.
- Dickey-Fuller Test: The test results comprise of a Test Statistic and some Critical Values for difference confidence levels. If the test statistic is less than the critical value, we can say that time series is stationary.

1.3.3.1 Rolling Statistics

In [27]:
```
def create_timeseries(df, variable, city):
    # Filter dataframe
    city_df = df[df['loc'] == city]

    # Sort date
    df['date'] = pd.to_datetime(df['date'], format='%d/%m/%Y')
    city_df_sorted = city_df.sort_values('date')

    # Create individual temporal timeseries data
    time_series = city_df_sorted.loc[:, ['date', variable]]
    time_series.index = time_series['date']
    ts = time_series.drop('date', axis=1)

    return ts
```

In [28]:
```
def check_mean_std(ts, city, window_size, ax=None):
    # Rolling statistics
    rolmean = ts.rolling(window=window_size).mean()
    rolstd = ts.rolling(window=window_size).std()

    if ax is None:
        ax = plt.gca()  # Get the current axes if None is provided

    ax.plot(ts.index, ts.values, color='red', label='Original')
    ax.plot(rolmean.index, rolmean.values, color='black', label='Rolling Mean')
    ax.plot(rolstd.index, rolstd.values, color='green', label='Rolling Std')

    ax.set_xlabel("Date")
    ax.set_ylabel("Cases")
    ax.set_title(f'Rolling Mean & Standard Deviation for {city}')
    ax.legend()

    return ax
```

In [29]:
```
variable = 'cases'
window_size = 4

fig, axs = plt.subplots(1, 3, figsize=(30, 8), sharex=True)
for i, city in enumerate(dengue_df['loc'].unique()):
    cases_ts = create_timeseries(dengue_df, variable, city)
    ax = axs[i // 1]
    check_mean_std(cases_ts, city, window_size, ax=ax)

plt.tight_layout()
plt.show()
```



1.3.3.2 Augemented Dickey-Fuller (ADF) Test

Dickey-Fuller Test:

- The test results comprise of a Test Statistic and some Critical Values for difference confidence levels.
- Hypothesis:

```
Hypothesis: The time series is stationary.
Null Hypothesis: The time series is not stationary.

If the 'Test Statistic' is less than the 'Critical Value', we can reject the null hypothesi
s and say that the series is stationary.
```

```
https://www.researchgate.net/publication/243644934_Distribution_of_the_Estimators_for_Autoregress
ive_Time_Series_With_a_Unit_Root
```

In [30]:
```
# Library
from statsmodels.tsa.stattools import adfuller
```

In [31]:
```
def check_adfuller(ts, city):

    # Dickey-Fuller test
    result = adfuller(ts, autolag='AIC')
    test_statistic = result[0]
    p_value = result[1]
    critical_values = result[4]

    print(f'Results for: {city}')
    print('ADF Test statistic: ' , result[0])
    print('p-value: ' , result[1])
    print('Critical Values:' ,result[4])
    print(' ')

    return test_statistic, critical_values
```

In [32]:
```
stationary_cities = []
non_stationary_cities = []

# Loop through unique cities
for city in dengue_df['loc'].unique():
    # Create time series for the city
    cases_ts = create_timeseries(dengue_df, 'cases',city)

    # Perform Dickey-Fuller test
    test_statistic, critical_values = check_adfuller(cases_ts['cases'], city)

    # Check if the test statistic is less than the critical value
    if test_statistic < critical_values['5%']:
        stationary_cities.append(city)
    else:
        non_stationary_cities.append(city)
```

```
Results for: BULACAN
ADF Test statistic:  -4.687209028908886
p-value:  8.89340327049059e-05
Critical Values: {'1%': -3.4562572510874396, '5%': -2.8729420379793598, '10%': -2.5728461399461744}

Results for: RIZAL
ADF Test statistic:  -3.961942028539276
p-value:  0.0016234425953196826
Critical Values: {'1%': -3.4562572510874396, '5%': -2.8729420379793598, '10%': -2.5728461399461744}

Results for: QUEZON CITY
ADF Test statistic:  -3.846997454400206
p-value:  0.002461458093812482
Critical Values: {'1%': -3.456360306409983, '5%': -2.8729872043802356, '10%': -2.572870232500465}
```

In [33]:
```
print(f"Stationary Cities for variable {variable}:")
for city in stationary_cities:
    print(city)
```

```
Stationary Cities for variable cases:
BULACAN
RIZAL
QUEZON CITY
```

In [34]:
```
print(f"Non-Stationary Cities for variable {variable}:")
for city in non_stationary_cities:
    print(city)
```

```
Non-Stationary Cities for variable cases:
```

1.3.3.3 KPSS (Kwiatkowski-Phillips-Schmidt-Shin) Test [...]

1.4 Climate Data

1.4.1 Content

In [35]:
```
climate_df
```

Out[35]:

	loc	date	MAX__AIRMASS	MIN__AIRMASS	MEAN__AIRMASS	SUM__AIRMASS	STD__AIRMASS	MAX__ALLSKY_
0	BULACAN	2016-01-10	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
1	BULACAN	2016-01-17	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
2	BULACAN	2016-01-24	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
3	BULACAN	2016-01-31	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
4	BULACAN	2016-02-07	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
...	...	...	...	...	...	...	...	
781	RIZAL	2020-12-13	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
782	RIZAL	2020-12-20	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
783	RIZAL	2020-12-27	-999.00	-999.00	-999.000000	-6993.00	0.000000	0
784	RIZAL	2021-01-03	18.18	-999.00	-563.070000	-3941.49	543.700235	0
785	RIZAL	2021-01-10	18.15	18.09	18.122857	126.86	0.019760	0

786 rows × 502 columns

In [36]:
```
climate_parameters_df
```

Out[36]:

	Name	Abbreviation	Recommended	Treatment	Unit	Definition	Group
0	Air Mass	AIRMASS	False	NaN	dimensionless	Air Mass according to Perez et al. 1990.	1
1	All Sky Insolation Clearness Index	ALLSKY_KT	True	MEAN	dimensionless	A fraction representing clearness of the atmos...	1
2	All Sky Surface Longwave Downward Irradiance	ALLSKY_SFC_LW_DWN	True	MEAN	MJ/m^2/day	The downward thermal infrared irradiance under...	1
3	All Sky Surface Longwave Upward Irradiance	ALLSKY_SFC_LW_UP	False	NaN	MJ/m^2/day	The upward thermal infrared irradiance under a...	1
4	All Sky Surface PAR Total	ALLSKY_SFC_PAR_TOT	False	NaN	MJ/m^2/day	The total Photosynthetically Active Radiation ...	1
...	...	...	...	...	...	...	...
95	Wind Speed at 50 Meters	WS50M	True	MEAN	m/s	The average of wind speed at 50 meters above t...	5
96	Wind Speed at 50 Meters Maximum	WS50M_MAX	True	MAX	m/s	The maximum hourly wind speed at 50 meters abo...	5
97	Wind Speed at 50 Meters Minimum	WS50M_MIN	True	MIN	m/s	The minimum hourly wind speed at 50 meters abo...	5
98	Wind Speed at 50 Meters Range	WS50M_RANGE	False	NaN	m/s	The minimum and maximum hourly wind speed rang...	5
99	Surface Roughness	Z0M	False	NaN	m	The height above the displacement plane at whi...	5

100 rows × 7 columns

1.4.2 Preparation

Filter Climate Data Based on which parameters to use

In [37]:
```
def filter_climate_df(climate_df, parameters_df):
    # Create a list of columns to drop based on 'Recommended' column
    columns_to_drop = []

    # Create a dictionary to map 'Treatment' to the corresponding column name prefix
    treatment_mapping = {
        'MEAN': 'MEAN__',
        'MAX': 'MAX__',
        'MIN': 'MIN__',
        'SUM': 'SUM__',
        'STD': 'STD__'
    }

    # Iterate over rows in parameters_df to get columns based on 'Treatment'
    for index, row in parameters_df.iterrows():
        if row['Recommended'] == False:
            abbreviation = row['Abbreviation']
#             print("abbreviation: ", abbreviation)
            columns_to_drop.extend([col for col in climate_df.columns if col.endswith(abbreviation)])

        if row['Recommended'] == True:
            treatment = row['Treatment']
            abbreviation = row['Abbreviation']
            column_prefix = treatment_mapping[treatment]
            column_name = f"{column_prefix}{abbreviation}"

            columns_to_drop.extend([col for col in climate_df.columns if col.endswith(abbreviation) and co


    # Create a new DataFrame with the selected columns
    filtered_df = climate_df.drop(columns=columns_to_drop, errors='ignore')

    return filtered_df
```

In [38]:
```
final_climate_df = filter_climate_df(climate_df, climate_parameters_df)
features_climate_df = final_climate_df.drop(columns=['loc', 'date'])
```

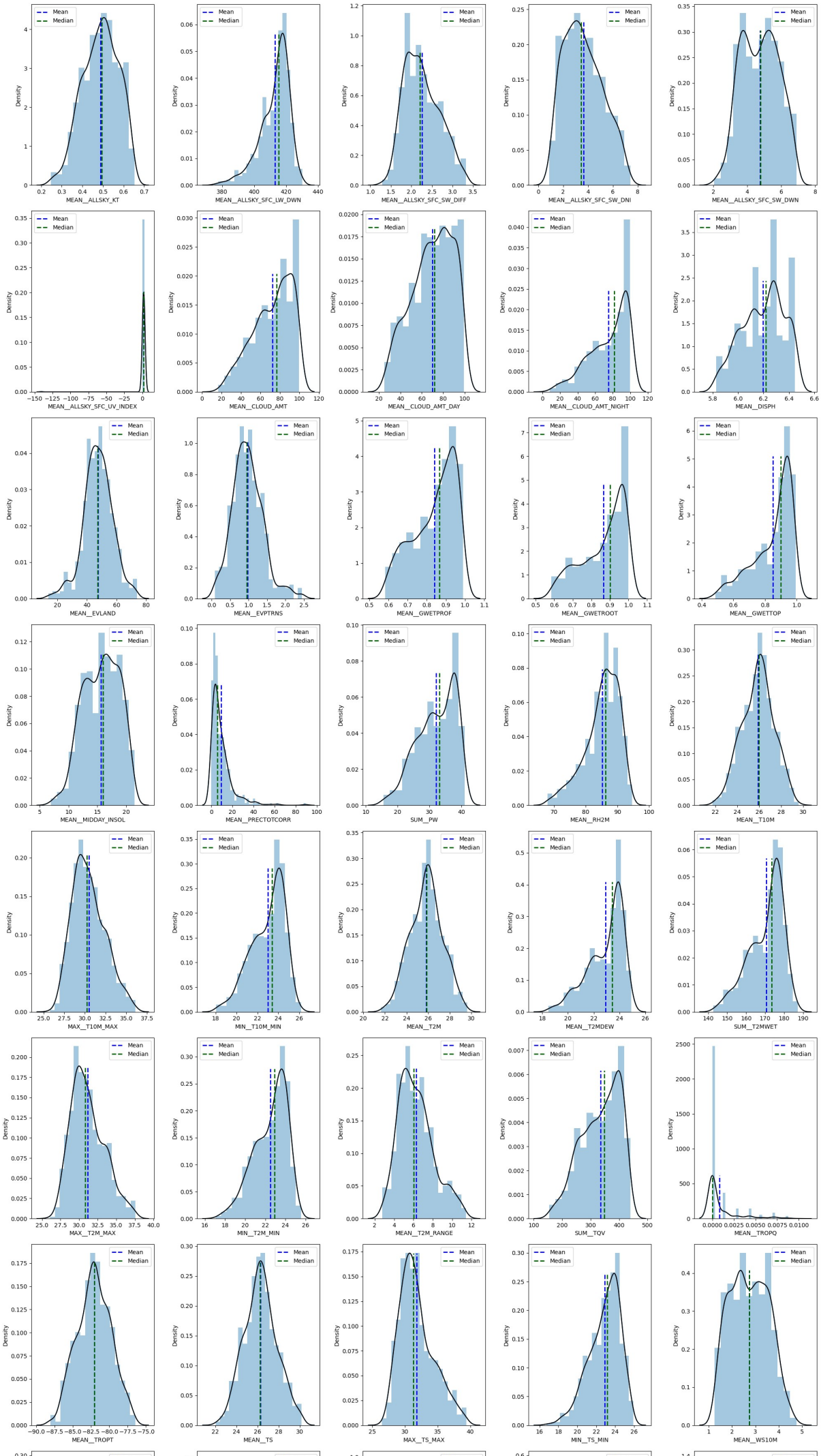
In [39]:
```
final_climate_df
```

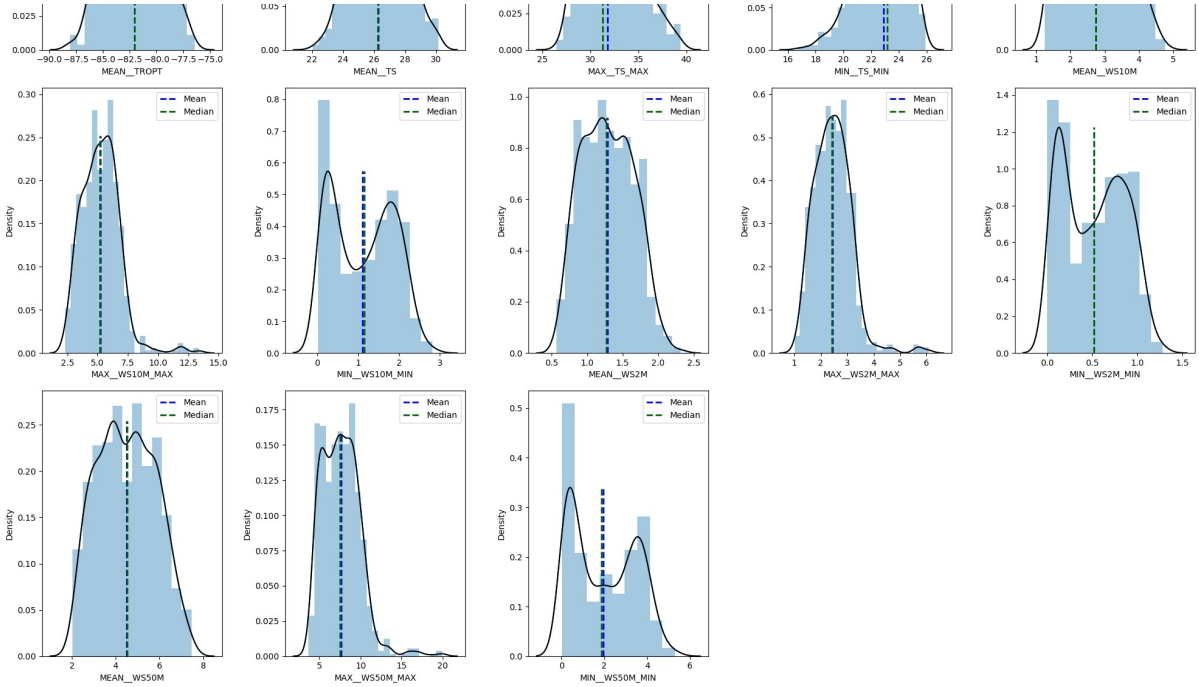
Out[39]:

	loc	date	MEAN__ALLSKY_KT	MEAN__ALLSKY_SFC_LW_DWN	MEAN__ALLSKY_SFC_SW_DIFF	MEAN__ALLSKY_
0	BULACAN	2016-01-10	0.601429	392.362857	1.388571	
1	BULACAN	2016-01-17	0.592857	397.481429	1.620000	
2	BULACAN	2016-01-24	0.528571	400.065714	1.607143	
3	BULACAN	2016-01-31	0.464286	403.874286	1.911429	
4	BULACAN	2016-02-07	0.464286	406.680000	1.890000	
...	...	...	...	...	...	
781	RIZAL	2020-12-13	0.402857	415.035714	1.825714	
782	RIZAL	2020-12-20	0.402857	412.948571	1.722857	
783	RIZAL	2020-12-27	0.417143	410.381429	1.655714	
784	RIZAL	2021-01-03	0.381429	412.025714	1.801429	
785	RIZAL	2021-01-10	0.441429	411.470000	2.080000	

786 rows × 45 columns

In [40]:
```
dist_plot(size_x=20, size_y=45, row=9, col=5, df=features_climate_df)
```

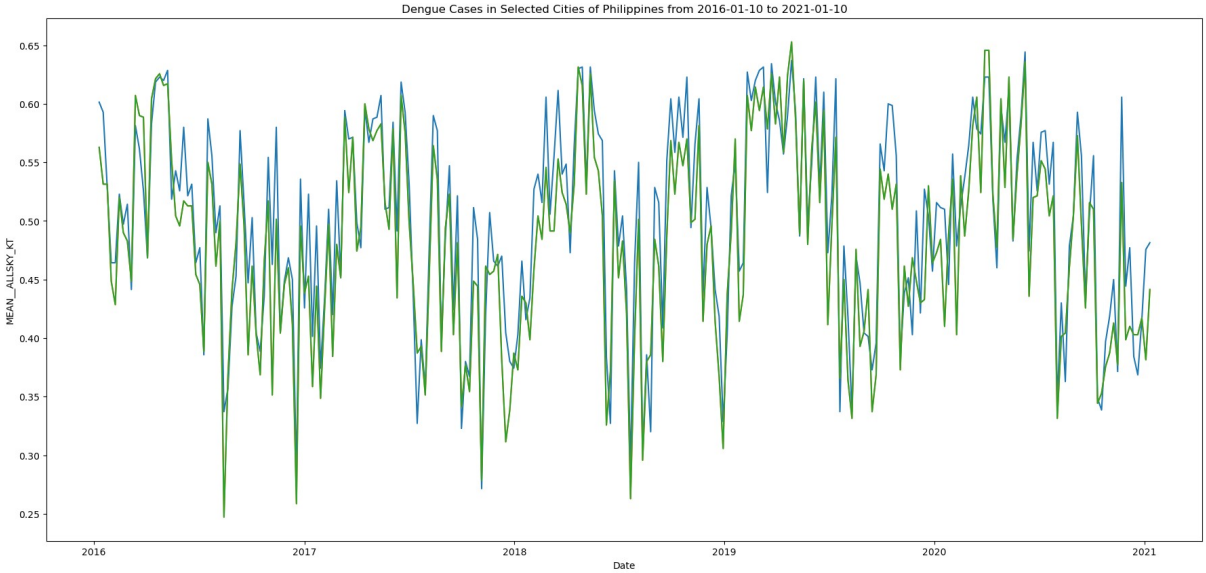

Mean
Median
Density
MEAN__ALLSKY_KT
MEAN__ALLSKY_SFC_LW_DWN
MEAN__ALLSKY_SFC_SW_DIFF
MEAN__ALLSKY_SFC_SW_DNI
MEAN__ALLSKY_SFC_SW_DWN
MEAN__ALLSKY_SFC_UV_INDEX
MEAN__CLOUD_AMT
MEAN__CLOUD_AMT_DAY
MEAN__CLOUD_AMT_NIGHT
MEAN__DISPH
MEAN__EVLAND
MEAN__EVPTRNS
MEAN__GWETPROF
MEAN__GWETROOT
MEAN__GWETTOP
MEAN__MIDDAY_INSOL
MEAN__PRECTOTCORR
SUM__PW
MEAN__RH2M
MEAN__T10M
MAX__T10M_MAX
MIN__T10M_MIN
MEAN__T2M
MEAN__T2MDEW
SUM__T2MWET
MAX__T2M_MAX
MIN__T2M_MIN
MEAN__T2M_RANGE
SUM__TQV
MEAN__TROPQ
MEAN__TROPT
MEAN__TS
MAX__TS_MAX
MIN__TS_MIN
MEAN__WS10M

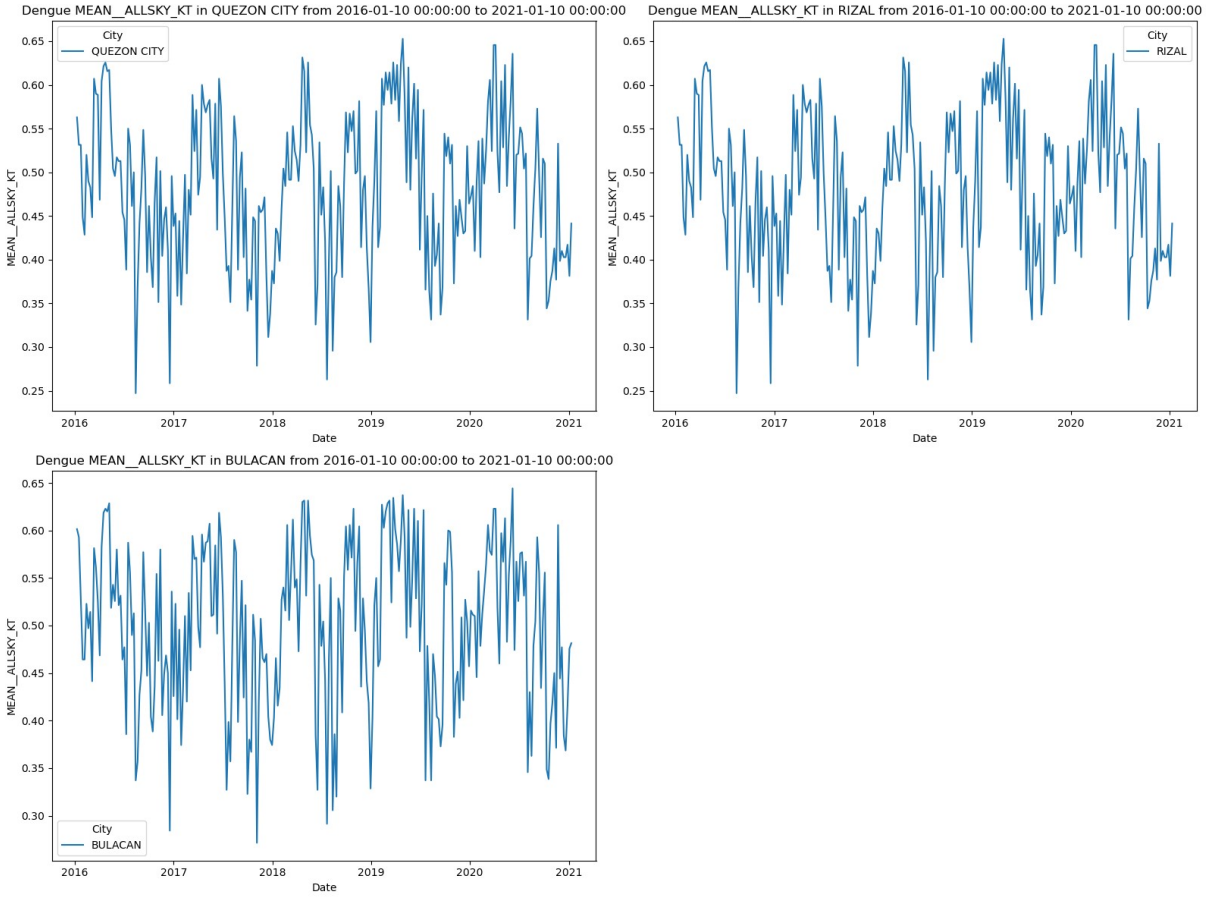


1.4.3 Visualization

Some Normally Distributed Variables

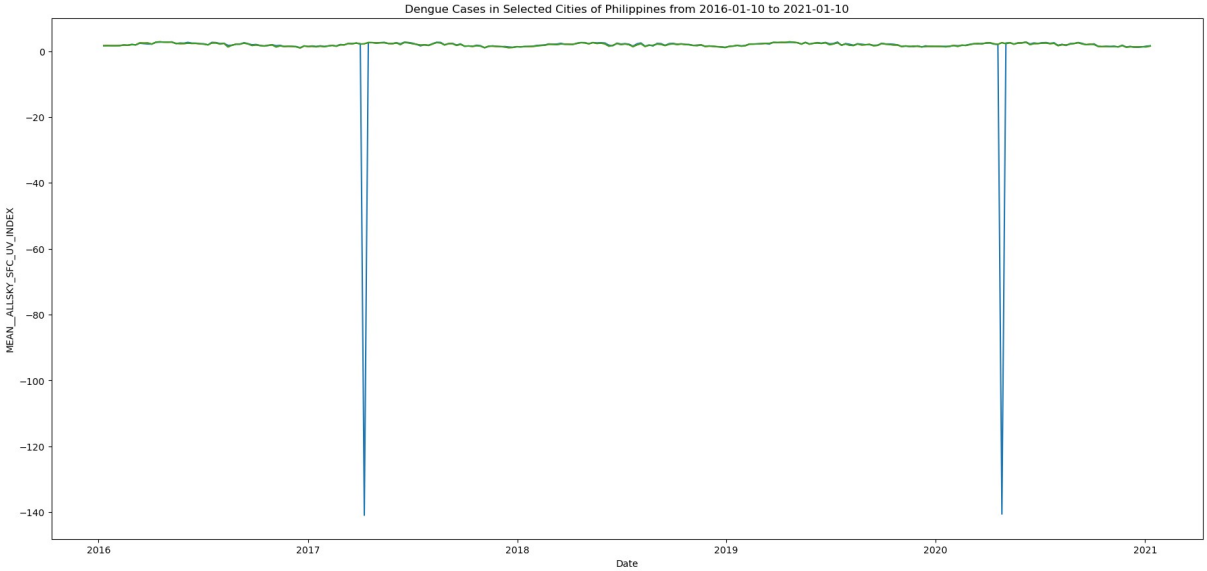
In [41]:
```
plot_top_(0, 'MEAN__ALLSKY_KT', final_climate_df)
separate_plot_top(4, 'MEAN__ALLSKY_KT', final_climate_df)
```

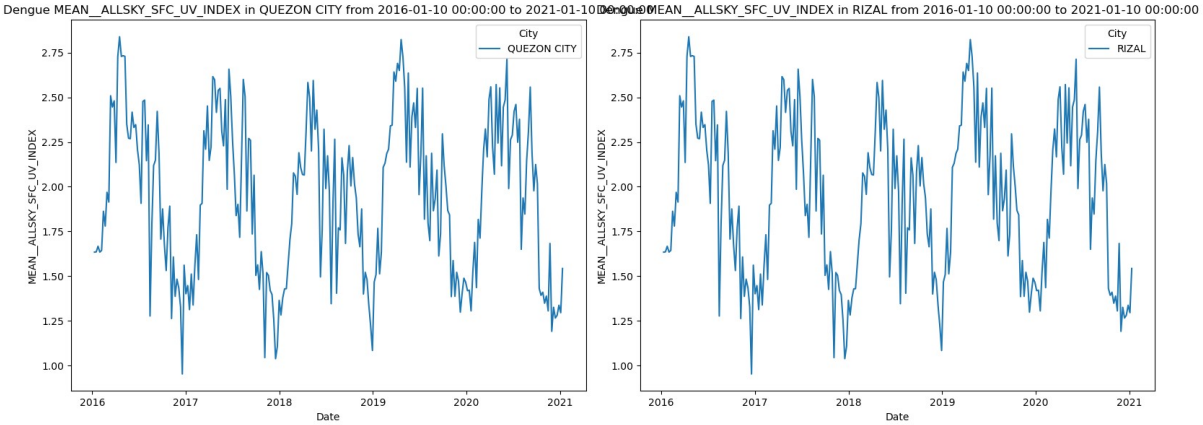


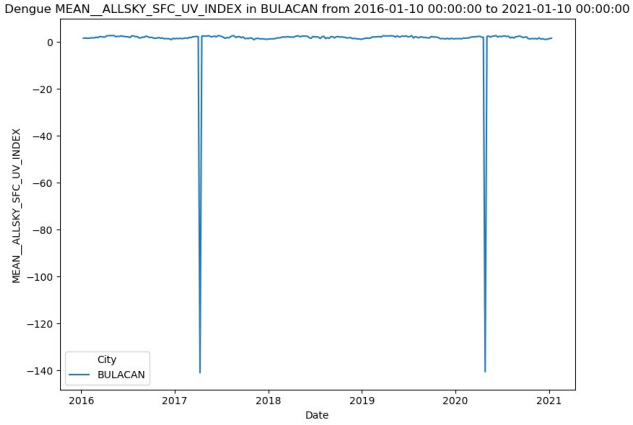


Some Left-Skewed Variables

In [42]:
```
plot_top_(0, 'MEAN__ALLSKY_SFC_UV_INDEX', final_climate_df)
separate_plot_top(4, 'MEAN__ALLSKY_SFC_UV_INDEX', final_climate_df)
```

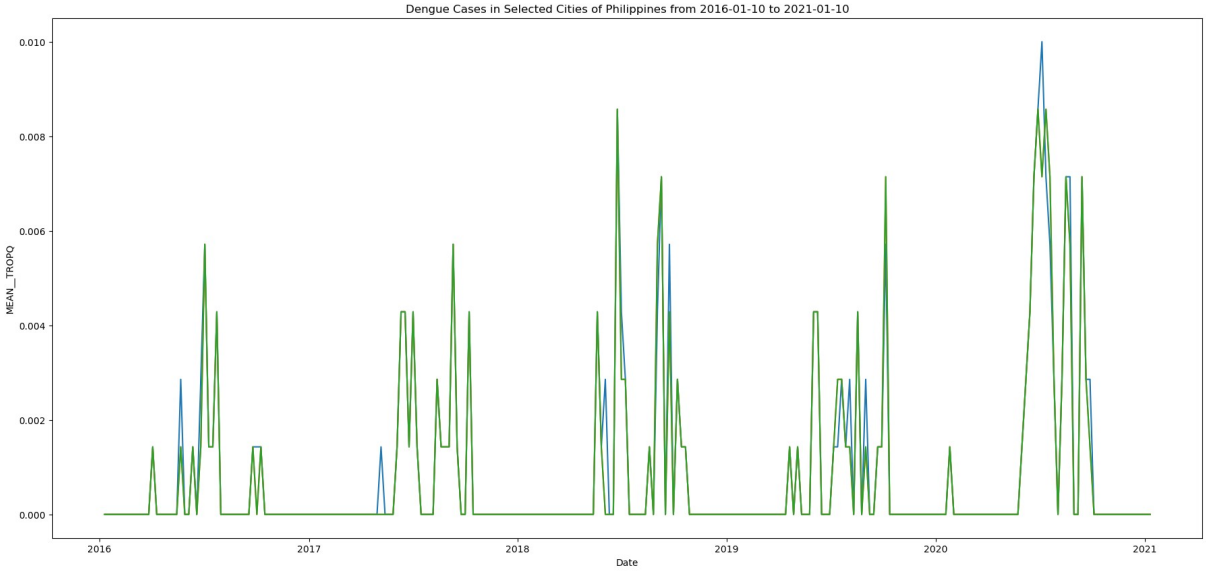


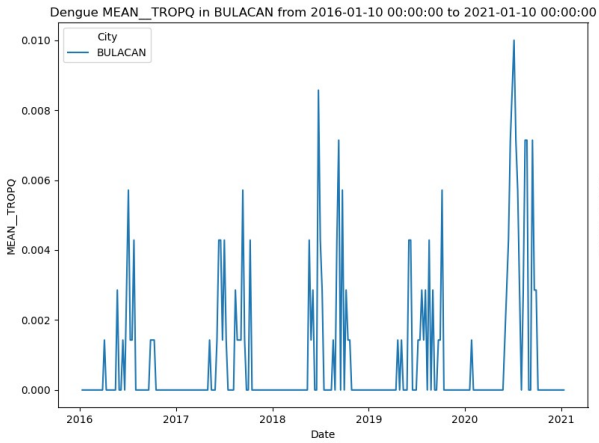


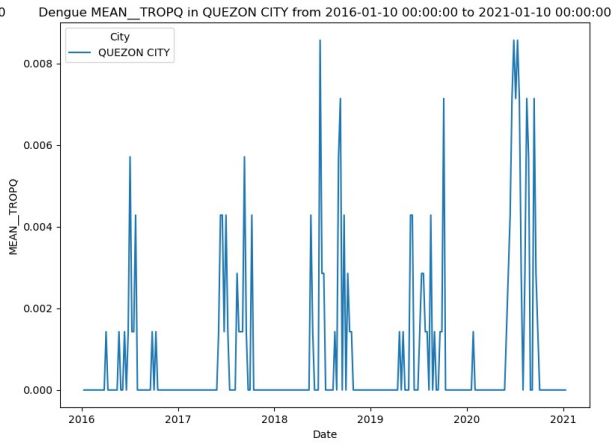


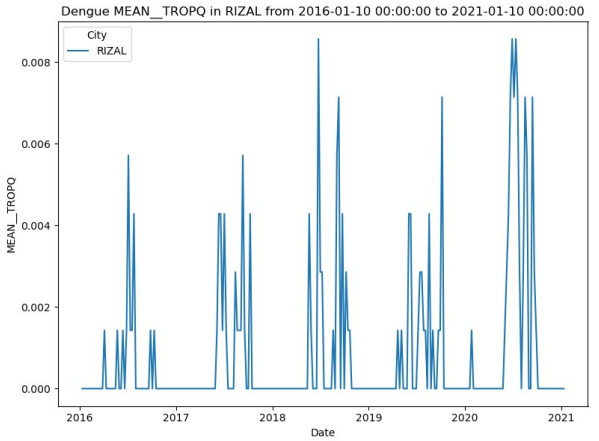
Some Right-Skewed Variables

In [43]:
```
plot_top_(0, 'MEAN__TROPQ', final_climate_df)
separate_plot_top(3, 'MEAN__TROPQ', final_climate_df)
```









General Insights

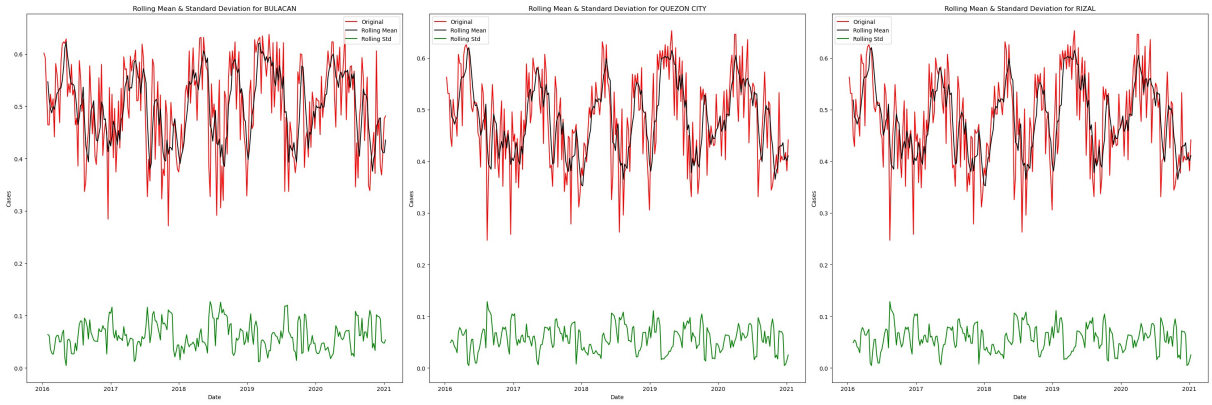
- Shows that behavior of timeseries data of most variables are not dependent on spatial property (city location)

1.4.4 Stationarity

1.4.4.1 Rolling Statistics

Sample

In [44]:
```
variable = 'MEAN__ALLSKY_KT'
dataframe = final_climate_df
window_size = 4

fig, axs = plt.subplots(1, 3, figsize=(30, 10), sharex=True)
for i, city in enumerate(dataframe['loc'].unique()):
    cases_ts = create_timeseries(dataframe, variable, city)
    ax = axs[i // 1] #i % 3]
    check_mean_std(cases_ts, city, window_size, ax=ax)

plt.tight_layout()
plt.show()
```



1.4.4.2 Augmented Dickey-Fuller

In [45]:
```
# List of columns to exclude from the analysis
exclude_columns = ['loc', 'date']

# Get the list of variables (columns) to process
variables = [col for col in final_climate_df.columns if col not in exclude_columns]

stationary_city_pairs = []
non_stationary_city_pairs = []
```

In [46]:
```
# Loop through each variable and perform the analysis
for variable in variables:
    print("Current Variable: ", variable, "\n")
    for city in dataframe['loc'].unique():
        # Create time series for the city
        variable_ts = create_timeseries(dataframe, variable, city)

        # Perform Dickey-Fuller test
        test_statistic, critical_values = check_adfuller(variable_ts[variable], city)

        # Check if the test statistic is less than the critical value
        if test_statistic < critical_values['5%']:
            stationary_city_pairs.append((variable, city))
        else:
            non_stationary_city_pairs.append((variable, city))
```

```
Current Variable:  MEAN__ALLSKY_KT

Results for: BULACAN
ADF Test statistic:  -7.070013426475254
p-value:  4.965228751712654e-10
Critical Values: {'1%': -3.4557539868570775, '5%': -2.8727214497041422, '10%': -2.572728476331361}

Results for: QUEZON CITY
ADF Test statistic:  -5.014200787903754
p-value:  2.080088854417232e-05
Critical Values: {'1%': -3.4558530692911504, '5%': -2.872764881778665, '10%': -2.572751643088207}

Results for: RIZAL
ADF Test statistic:  -5.014200787903754
p-value:  2.080088854417232e-05
Critical Values: {'1%': -3.4558530692911504, '5%': -2.872764881778665, '10%': -2.572751643088207}

Current Variable:  MEAN__ALLSKY_SFC_LW_DWN

Results for: BULACAN
```

In [47]:
```
# Print the results
print("Stationary City Pairs:")
for pair in stationary_city_pairs:
    print(pair[0], "-", pair[1])

print("\nNon-Stationary City Pairs:")
for pair in non_stationary_city_pairs:
    print(pair[0], "-", pair[1])
```

```
Stationary City Pairs:
MEAN__ALLSKY_KT - BULACAN
MEAN__ALLSKY_KT - QUEZON CITY
MEAN__ALLSKY_KT - RIZAL
MEAN__ALLSKY_SFC_LW_DWN - BULACAN
MEAN__ALLSKY_SFC_LW_DWN - QUEZON CITY
MEAN__ALLSKY_SFC_LW_DWN - RIZAL
MEAN__ALLSKY_SFC_SW_DIFF - BULACAN
MEAN__ALLSKY_SFC_SW_DIFF - QUEZON CITY
MEAN__ALLSKY_SFC_SW_DIFF - RIZAL
MEAN__ALLSKY_SFC_SW_DNI - BULACAN
MEAN__ALLSKY_SFC_SW_DNI - QUEZON CITY
MEAN__ALLSKY_SFC_SW_DNI - RIZAL
MEAN__ALLSKY_SFC_SW_DWN - BULACAN
MEAN__ALLSKY_SFC_SW_DWN - QUEZON CITY
MEAN__ALLSKY_SFC_SW_DWN - RIZAL
MEAN__ALLSKY_SFC_UV_INDEX - BULACAN
MEAN__ALLSKY_SFC_UV_INDEX - QUEZON CITY
MEAN__ALLSKY_SFC_UV_INDEX - RIZAL
```

General Insight

- Seems likes climate data are stationary in nature

1.5 Google Trend

1.5.1 Content

In [66]:
```
trend_dengue_df
```

Out[66]:

	index	loc	date	trend
0	880.0	Quezon	2016-01-10	1.0
1	880.0	Quezon	2016-01-17	0.0
2	880.0	Quezon	2016-01-24	2.0
3	880.0	Quezon	2016-01-31	1.0
4	880.0	Quezon	2016-02-07	4.0
...	...	...	...	...
775	NaN	BULACAN	2020-11-29	64.0
776	NaN	BULACAN	2020-12-06	55.0
777	NaN	BULACAN	2020-12-13	48.0
778	NaN	BULACAN	2020-12-20	8.0
779	NaN	BULACAN	2020-12-27	56.0

780 rows × 4 columns

In [67]:
```
trend_denguesympt_df
```

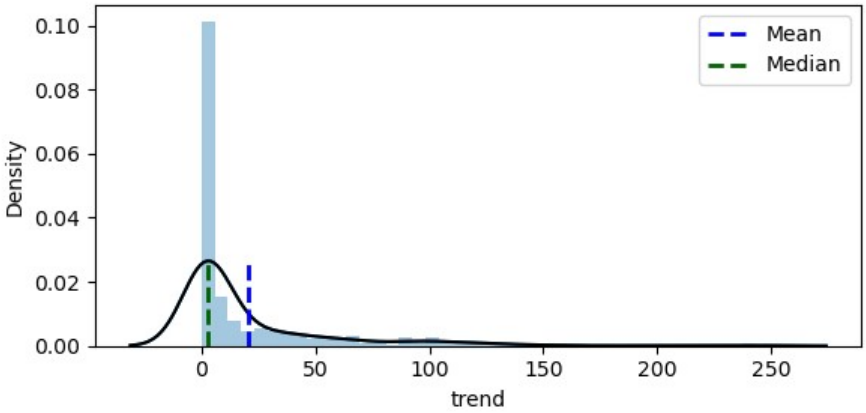
Out[67]:

	index	loc	date	trend
0	36.0	Quezon	2016-01-10	4.0
1	36.0	Quezon	2016-01-17	7.0
2	36.0	Quezon	2016-01-24	2.0
3	36.0	Quezon	2016-01-31	5.0
4	36.0	Quezon	2016-02-07	5.0
...	...	...	...	...
775	NaN	BULACAN	2020-11-29	22.0
776	NaN	BULACAN	2020-12-06	0.0
777	NaN	BULACAN	2020-12-13	87.0
778	NaN	BULACAN	2020-12-20	0.0
779	NaN	BULACAN	2020-12-27	0.0

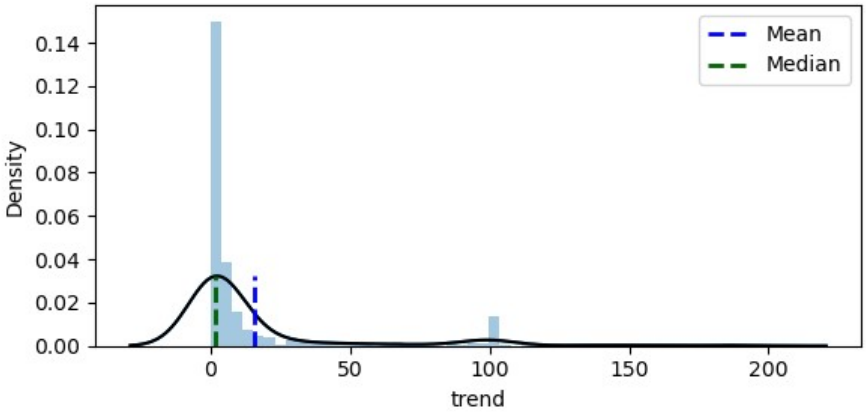
780 rows × 4 columns

1.5.2 Visualization

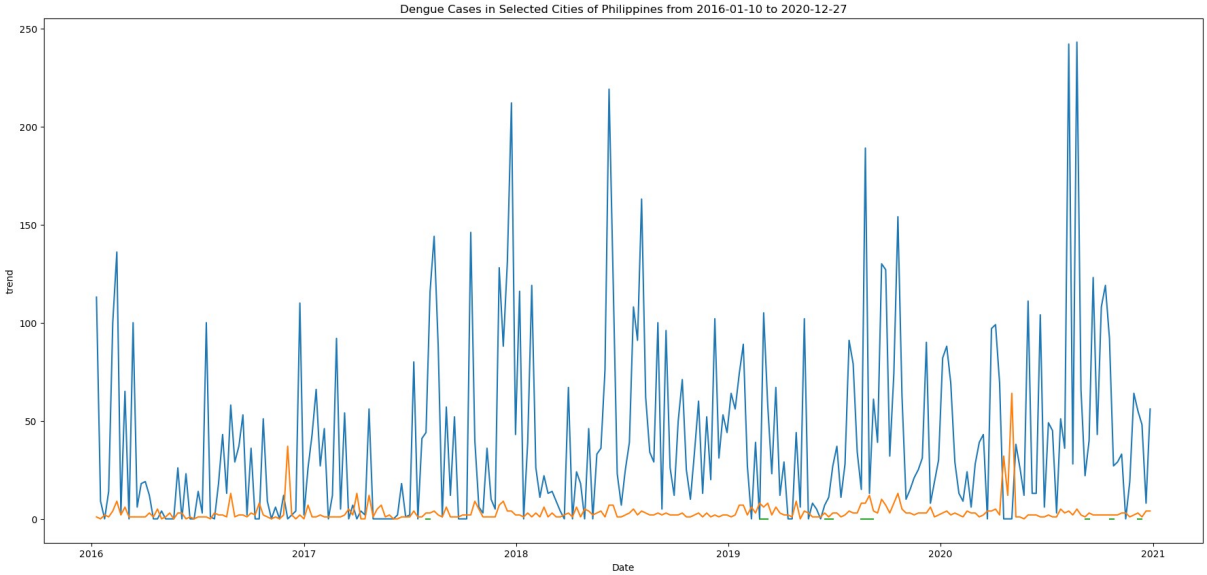
In [68]:
```
dist_plot(size_x=6, size_y= 3, row=1, col=1, df=trend_dengue_df[['trend']])
```

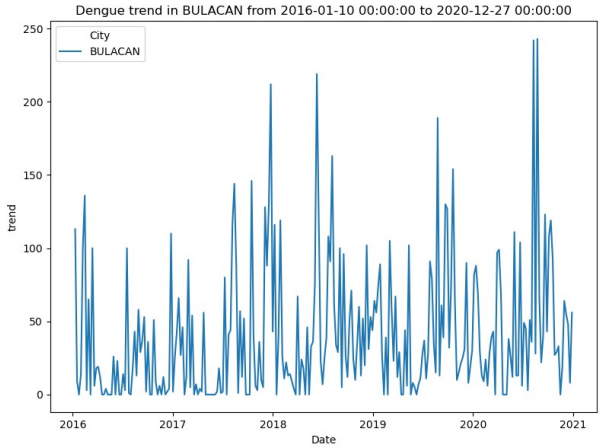


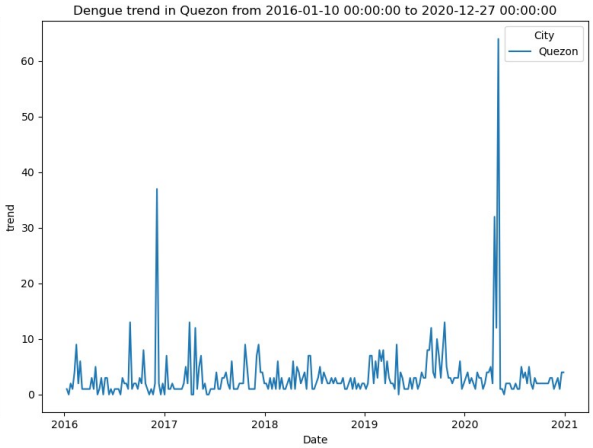
In [69]:
```
dist_plot(size_x=6, size_y= 3, row=1, col=1, df=trend_denguesympt_df[['trend']])
```

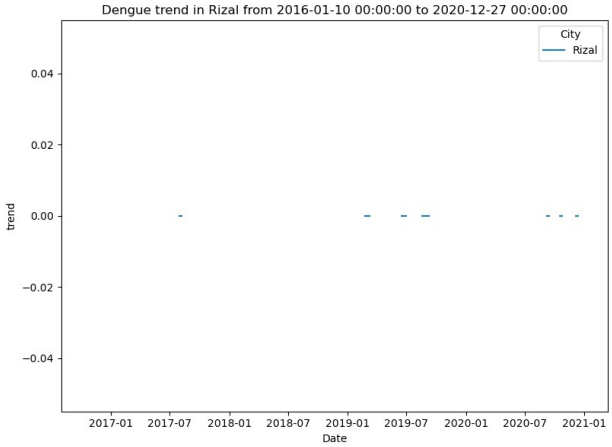


In [70]:
```
plot_top_(0, 'trend', trend_dengue_df)
separate_plot_top(4, 'trend', trend_dengue_df)
```

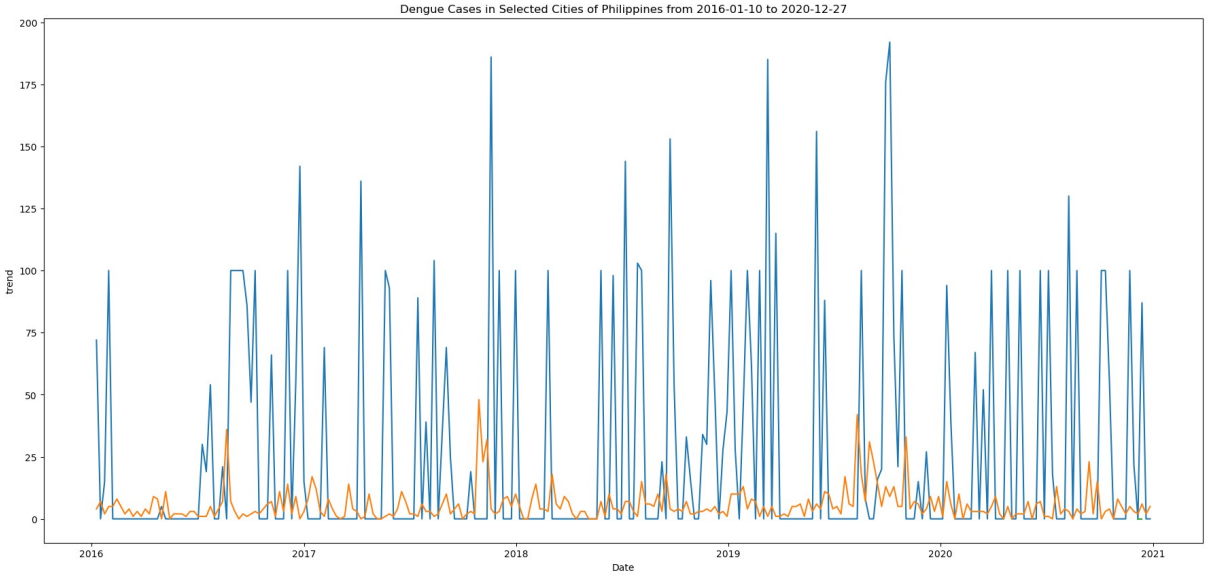


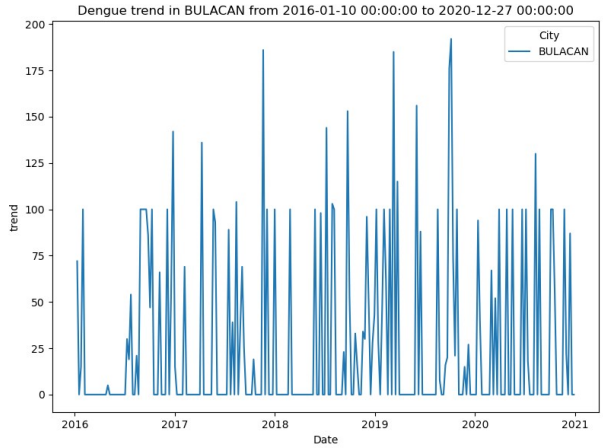


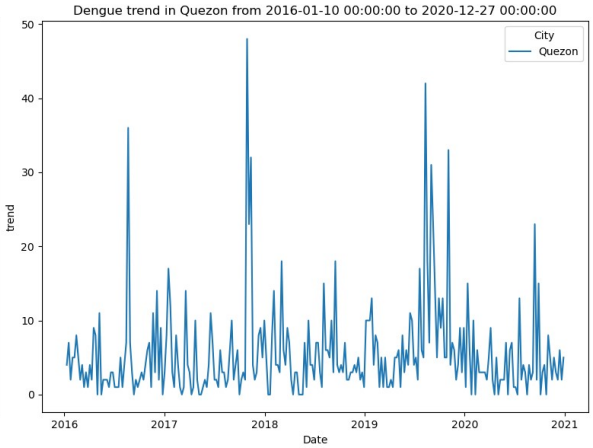


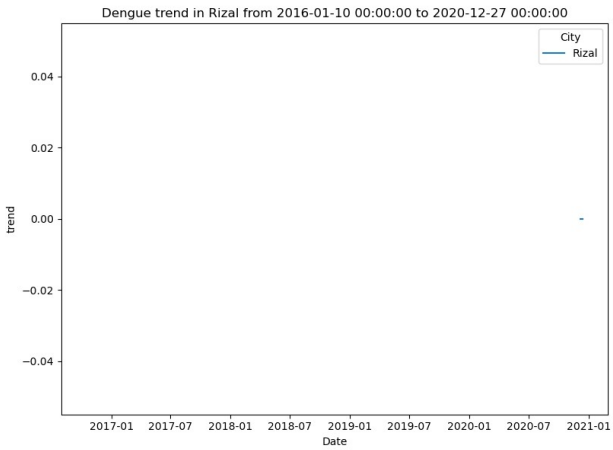


In [71]:
```
plot_top_(0, 'trend', trend_denguesympt_df)
separate_plot_top(4, 'trend', trend_denguesympt_df)
```









1.5.3 Stationarity

In [72]:
```
# Group by 'loc' and compute the sum of the 'trend' column for each group
city_trend_sums = trend_dengue_df.groupby('loc')['trend'].sum()

# Filter out the cities where the sum of 'trend' is 0
cities_to_keep = city_trend_sums[city_trend_sums > 0].index

# Filter the original dataframe to keep only the rows for the cities to keep
trend_dengue2_df = trend_dengue_df[trend_dengue_df['loc'].isin(cities_to_keep)]
```

In [73]:
```
variable = 'trend'
dataframe = trend_dengue2_df

stationary_cities = []
non_stationary_cities = []

# Loop through unique cities
for city in dataframe['loc'].unique():
#     print('current city:', city)
    # Create time series for the city
    variable_ts = create_timeseries(dataframe, variable,city)

    # Perform Dickey-Fuller test
    test_statistic, critical_values = check_adfuller(variable_ts[variable], city)

    # Check if the test statistic is less than the critical value
    if test_statistic < critical_values['5%']:
        stationary_cities.append(city)
    else:
        non_stationary_cities.append(city)

print(f"Stationary Cities for variable {variable}:")
for city in stationary_cities:
    print(city)

print(f"Non-Stationary Cities for variable {variable}:")
for city in non_stationary_cities:
    print(city)
```

```
Results for: Quezon
ADF Test statistic:  -8.508427279584065
p-value:  1.185399100982125e-13
Critical Values: {'1%': -3.455952927706342, '5%': -2.8728086526320302, '10%': -2.572774990685656}

Results for: BULACAN
ADF Test statistic:  -8.442900109580117
p-value:  1.7437491245531152e-13
Critical Values: {'1%': -3.455952927706342, '5%': -2.8728086526320302, '10%': -2.572774990685656}

Stationary Cities for variable trend:
Quezon
BULACAN
Non-Stationary Cities for variable trend:
```

In [74]:
```
# Group by 'loc' and compute the sum of the 'trend' column for each group
city_trend_sums = trend_denguesympt_df.groupby('loc')['trend'].sum()

# Filter out the cities where the sum of 'trend' is 0
cities_to_keep = city_trend_sums[city_trend_sums > 0].index

# Filter the original dataframe to keep only the rows for the cities to keep
trend_denguesympt2_df = trend_dengue_df[trend_dengue_df['loc'].isin(cities_to_keep)]
```

In [75]:
```
variable = 'trend'
dataframe = trend_denguesympt2_df

stationary_cities = []
non_stationary_cities = []

# Loop through unique cities
for city in dataframe['loc'].unique():
#     print('current city:', city)
    # Create time series for the city
    variable_ts = create_timeseries(dataframe, variable,city)

    # Perform Dickey-Fuller test
    test_statistic, critical_values = check_adfuller(variable_ts[variable], city)

    # Check if the test statistic is less than the critical value
    if test_statistic < critical_values['5%']:
        stationary_cities.append(city)
    else:
        non_stationary_cities.append(city)

print(f"Stationary Cities for variable {variable}:")
for city in stationary_cities:
    print(city)

print(f"Non-Stationary Cities for variable {variable}:")
for city in non_stationary_cities:
    print(city)
```

```
Results for: Quezon
ADF Test statistic:  -8.508427279584065
p-value:  1.185399100982125e-13
Critical Values: {'1%': -3.455952927706342, '5%': -2.8728086526320302, '10%': -2.572774990685656}

Results for: BULACAN
ADF Test statistic:  -8.442900109580117
p-value:  1.7437491245531152e-13
Critical Values: {'1%': -3.455952927706342, '5%': -2.8728086526320302, '10%': -2.572774990685656}

Stationary Cities for variable trend:
Quezon
BULACAN
Non-Stationary Cities for variable trend:
```

In [ ]:

In [ ]: