Project 2 report

In [ ]:
```
Github repository: https://github.com/pypr-2122/pp-project-2-group-2.git
```

Introduction

Air quality is one of the most popular topics in the world at present. Every country, or every region might be affected by air pollution.

In our project, we focus our investigation on PM2.5 pollution in India, and we complete our investigation in three aspects:

- Provide an overview of how values of PM2.5 differ in different Indian cities by plotting a map, and show the city with the worst air quality.
- Study if there is a relationship between population size and air quality (i.e. PM2.5 values).
- Investigate whether changes of weather will play a role in changes of air quality.

Data Set

- The air quality data is provided by [OpenAQ (https://openaq.org/#/)](https://openaq.org/#/), an NGO which aggregates and distributes open data on air quality in many different countries, sourced mainly from government sources. From OpenAQ, 153,678,944 air quality measurements have been collected from 8,247 locations in 64 countries as of Jan 2018. Data are aggregated from 101 government level and research-grade sources. The data set in this project consists of the mean values of PM2.5 of 155 Indian cities and of different locations in Delhi over the year 2021.

- The population data is provided by [Geocoding API (https://open-meteo.com/en/docs/geocoding-api)](https://open-meteo.com/en/docs/geocoding-api), which is used to obtain latitude coordinate, longitude coordinate and population for 11 million different cities and towns in the world. The data set in this project consists of the population of 129 Indian cities.

- The weather data is provided by [Meteostat API (https://dev.meteostat.net/guide.html)](https://dev.meteostat.net/guide.html), which is used to obtain historical weather data. The data set in this project consists of the average air temperature in °C, the daily precipitation total in mm and the average wind speed in km/h of the weather station in Delhi.

PART 1: Visualize Air Pollution (PM2.5) Situation in India

The air quality data we are interested in is the values of PM2.5 for Indian cities.

We create a dataframe called `geo_data`, containing the names of displayed Indian cities (from **OpenAQ** API), along with their correponding latitudes and longitudes, and the mean value of PM2.5 in each city.

To have a direct visualization of PM2.5 pollution in India, we decide to draw a map using **folium** package.

In [1]:
```
%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import openaq
import folium
import os
from folium import plugins
import branca.colormap as cm

# Set major seaborn asthetics
sns.set("notebook", style = 'ticks', font_scale = 1.0)

# Increase the quality of inline plots
mpl.rcParams['figure.dpi'] = 500
```

In [2]:
```
# Initiate an instance of the openaq.OpenAQ class so we can begin looking at data
api = openaq.OpenAQ()

# Grab the past 10000 data points for PM2.5 in cities of India
res = api.measurements(country = 'IN', parameter = 'pm25', limit = 10000, df = True)

# Get the statistics on a per-city basis
city_with_pm25 = res.groupby(['city'])['value'].describe()

# Get all the Indian cities into a list
IN_cities = city_with_pm25.index.to_list()

# Get the mean value of PM2.5 in each city into a list
mean = city_with_pm25['mean'].to_list()
```

```
/Users/candice/anaconda3/envs/pp-proj2-2/lib/python3.8/site-packages/openaq/decorators.p
y:57: FutureWarning: pandas.io.json.json_normalize is deprecated, use pandas.json_normali
ze instead
  data = pd.io.json.json_normalize(resp)
```

In [3]:
```
# Create two empty lists to store latitude and longitude for each city respectively
latitude = []
longitude = []

# Loop over each city, collect corresponding latitude and longitude, and append both to the
for i in range(len(mean)):
    latitude.append(res.loc[res['city'] == IN_cities[i], 'coordinates.latitude'].values[0])
    longitude.append(res.loc[res['city'] == IN_cities[i], 'coordinates.longitude'].values[(

# Get the desired dataframe
geo_data = pd.DataFrame({'city': IN_cities,
                         'mean': mean,
                         'latitude': latitude,
                         'longitude': longitude})

# Export 'geo_data' to a csv file in order to avoid the influence of api real time updating
# outputpath = '/Users/wangyiyi/Desktop/pp-project-2-group-2/geo_data.csv'
# geo_data.to_csv(outputpath, sep = ',', index = False, header = False)
```

We check the Indian cities with PM2.5 values from OpenAQ and find that 155 Indian cities have their PM2.5 values recorded.

The dataframe `city_with_pm25` shows some statistics of PM2.5 values of 155 cities and we use the mean as PM2.5 value.

For easier use, we create a new dataframe called `geo_data` to save only the mean values of PM2.5, city names and their corresponding locations.

The 155 Indian cities are stored in a list called `IN_cities` .

In [4]:

```
# Read the 'geo_data' file we have saved
colnames = ['city','mean','latitude','longitude']
geo_data = pd.read_csv("geo_data.csv", names = colnames, header = None)

# Show the data
geo_data
```

Out[4]:

	city	mean	latitude	longitude
0	Agartala	48.948235	23.862828	91.288736
1	Agra	112.907928	27.198620	77.920660
2	Ahmedabad	28.200962	23.043070	72.562968
3	Aizawl	1.575946	23.717634	92.719284
4	Ajmer	45.805128	26.470859	74.646594
...	...	...	...	...
150	Vijayapura	14.960270	16.802639	75.722694
151	Visakhapatnam	65.325000	17.720000	83.300000
152	Vrindavan	139.690714	27.571409	77.655757
153	Yadgir	57.058824	16.760200	77.142800
154	Yamunanagar	173.452778	30.148057	77.289347

155 rows × 4 columns

In [5]:
```
# Lines 16-23: Nagasudhir

# URL: https://nagasudhir.blogspot.com/2021/08/create-bubble-map-from-excel-data-using.html

# "Create a bubble map from excel data using python folium and pandas", August 29,2021.

# Accessed on 6 Dec 2021.

# Create a base map, and also pass the starting coordinates to Folium
mapObj = folium.Map(location = [26.86328062676624, 80.71655273437501], zoom_start = 5.5, ti

# Import geospatial data of the border of India
border = os.path.join('states_india.geojson')

# Define border style
borderStyle = {'color': 'blue',
               'weight': 2,
               'fill': False}

# Pass the border layer to the map as an overlay
folium.GeoJson(data = border,
               name = 'Borders',
               style_function = lambda x: borderStyle).add_to(mapObj)

# Create a ColorMap based on linear interpolation of a set of colors over a given index
colormap = cm.LinearColormap(colors = ['greenyellow', 'red'], # Gradient colors
                             index = [0,1000], # The range of values of PM2.5
                             vmin = 0,
                             vmax = 1000,
                             caption = 'PM2.5 Levels')

# Add colormap object to the main map
mapObj.add_child(colormap)

# Create a FeatureGroup layer
# You can put things in it and handle them as a single layer
# For example, you can add a LayerControl to tick/untick the whole group

# Create a FeatureGroup layer and put mean levels of PM2.5 as a whole group in it
pm25Layer = folium.FeatureGroup("Mean of PM 2.5").add_to(mapObj)

# Store elements in the lists
latStr = list(geo_data['latitude'])
lngStr = list(geo_data['longitude'])
meanStr = list(geo_data['mean'])
cityStr = list(geo_data['city'])

# Loop over each city, add markers on the map with different colors indicating levels of PM
for loc, m, c in zip(zip(latStr, lngStr), meanStr, cityStr):
    folium.CircleMarker(location = loc, # The loc is the coordinates for a specific city.
                        radius = 6, # The size of the circle marker.
                        fill = True,
                        color = colormap(m), # The color will change from green to red as m
                        tooltip =(c, round(m, 2)) # Popups, display city and its mean level
                       ).add_to(pm25Layer)

# Add a minimap on the side
minimap = plugins.MiniMap()
mapObj.add_child(minimap)

# Allow multiple layers can be visualized on the same map, and one can tick/untick differen

folium.LayerControl().add_to(mapObj)

# Save it in a file
mapObj.save("index.html")

# Display it in the Jupyter notebook
```

```
mapObj
```

Out[5]: Make this Notebook Trusted to load map: File -> Trust Notebook

In the above map plot, each circle corresponds to the mean value of PM2.5 in each city contained in the `geo_data` dataframe. When we move the mouse near the cirle, we can see the name of the city of and the following PM2.5 value.

We set up a color map so that the mean values of PM2.5 can be displayed on the graph from green to red, from the smallest to the largest. In other words, the green circle can indicate the corresponding city has a relative good air quality (i.e. low PM2.5 value), and the reddest circle infers that the corresponding city has worst air quality (i.e. highest PM2.5 value).

From the map, we can easily have a general idea of the air pollution situation in Indian cities, and the plot displays that the city with the worst air quality is **Faridabad** as the corresponding circle marker is the reddest one.

In [6]:
```
# Find the city with the worst air condition
worstcity = geo_data[geo_data['mean'] == max(geo_data['mean'])]
print(worstcity)
```

```
         city       mean   latitude  longitude
43  Faridabad  932.45125  28.408842  77.309908
```

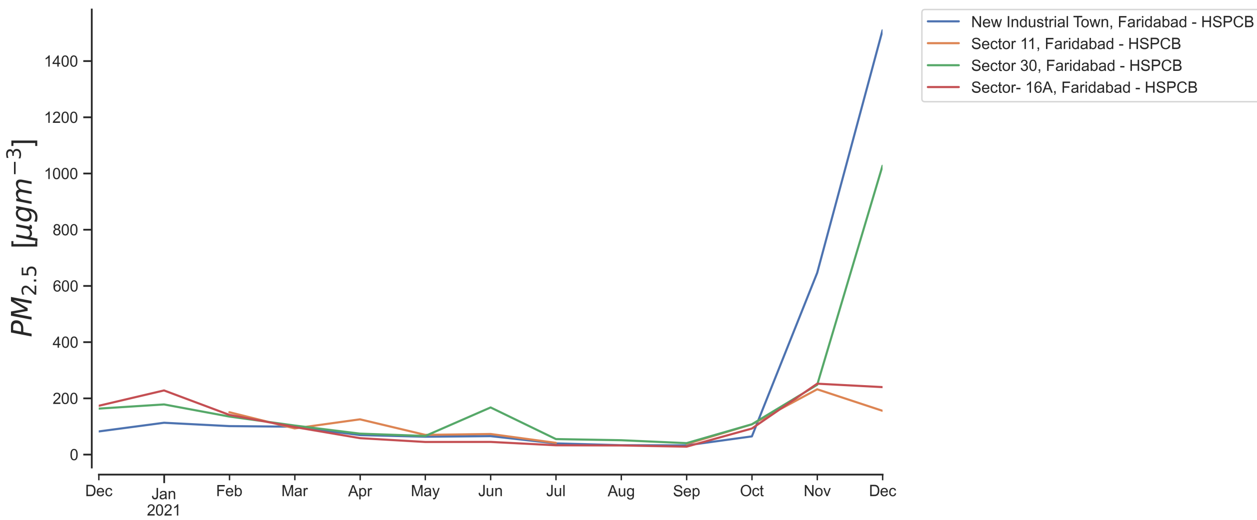
We can use `geo_data` to find the exact city with the highest mean PM 2.5 value, and we find the city with the worst air quality contained in the dataframe is **Faridabad**, which is same as we can see in the previous map.

As we have found the city with the worst air pollution in India, we want to do some analysis about the city **Faridabad**.

In [7]:
```
# Grab the past 10000 data points for PM2.5 in city Faridabad
worst = api.measurements(city = 'Faridabad', parameter = 'pm25', limit = 10000, df = True)

# Export 'worst' to a csv file in order to avoid the influence of api real time updating
# outputpath = '/Users/wangyiyi/Desktop/pp-project-2-group-2/worst_city.csv'
# worst.to_csv(outputpath, sep = ',', index = False, header = False)

# Read 'worst_city' file we have saved
worst_city = pd.read_csv('worst_city.csv',
                         sep = ',',
                         names = ['location', 'parameter', 'value', 'unit', 'country', 'cit
                                  'date.utc', 'coordinates.latitude', 'coordinates.longitud

# Change the dtype of 'date.utc' back from object to datetime
worst_city['date.utc'] = pd.to_datetime(worst_city['date.utc'])

# Ensure the dataframe has DatetimeIndex, not RangeIndex
worst_city = worst_city.set_index(worst_city['date.utc'])

# Show the information of the dataframe
worst_city.info()
```

```
/Users/candice/anaconda3/envs/pp-proj2-2/lib/python3.8/site-packages/openaq/decorators.p
y:57: FutureWarning: pandas.io.json.json_normalize is deprecated, use pandas.json_normali
ze instead
  data = pd.io.json.json_normalize(resp)

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 13085 entries, 2021-12-02 20:30:00+00:00 to 2020-12-27 23:15:00+00:00
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   location               13085 non-null  object
 1   parameter              13085 non-null  object
 2   value                  13085 non-null  float64
 3   unit                   13085 non-null  object
 4   country                13085 non-null  object
 5   city                   13085 non-null  object
 6   date.utc               13085 non-null  datetime64[ns, UTC]
 7   coordinates.latitude   13085 non-null  float64
 8   coordinates.longitude  13085 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(3), object(5)
memory usage: 1022.3+ KB
```

In [8]:
```
# Create the figure and axes
fig, ax = plt.subplots(1, figsize = (10, 6))

for group, df in worst_city.groupby('location'):
    # Query the data to only get positive values and resample to monthly
    _df = df.query("value >= 0.0").resample('1m').mean()
    _df.value.plot(ax = ax, label = group)

# Set the legend and the axis label
ax.legend(loc = 'best')
ax.set_ylabel("$PM_{2.5}$  [$\mu g m^{-3}$]", fontsize = 20)
ax.set_xlabel("")
sns.despine(offset = 5)
plt.legend(bbox_to_anchor = (1.05, 1), loc = 2, borderaxespad = 0.)

# Display the plot
plt.show()
```

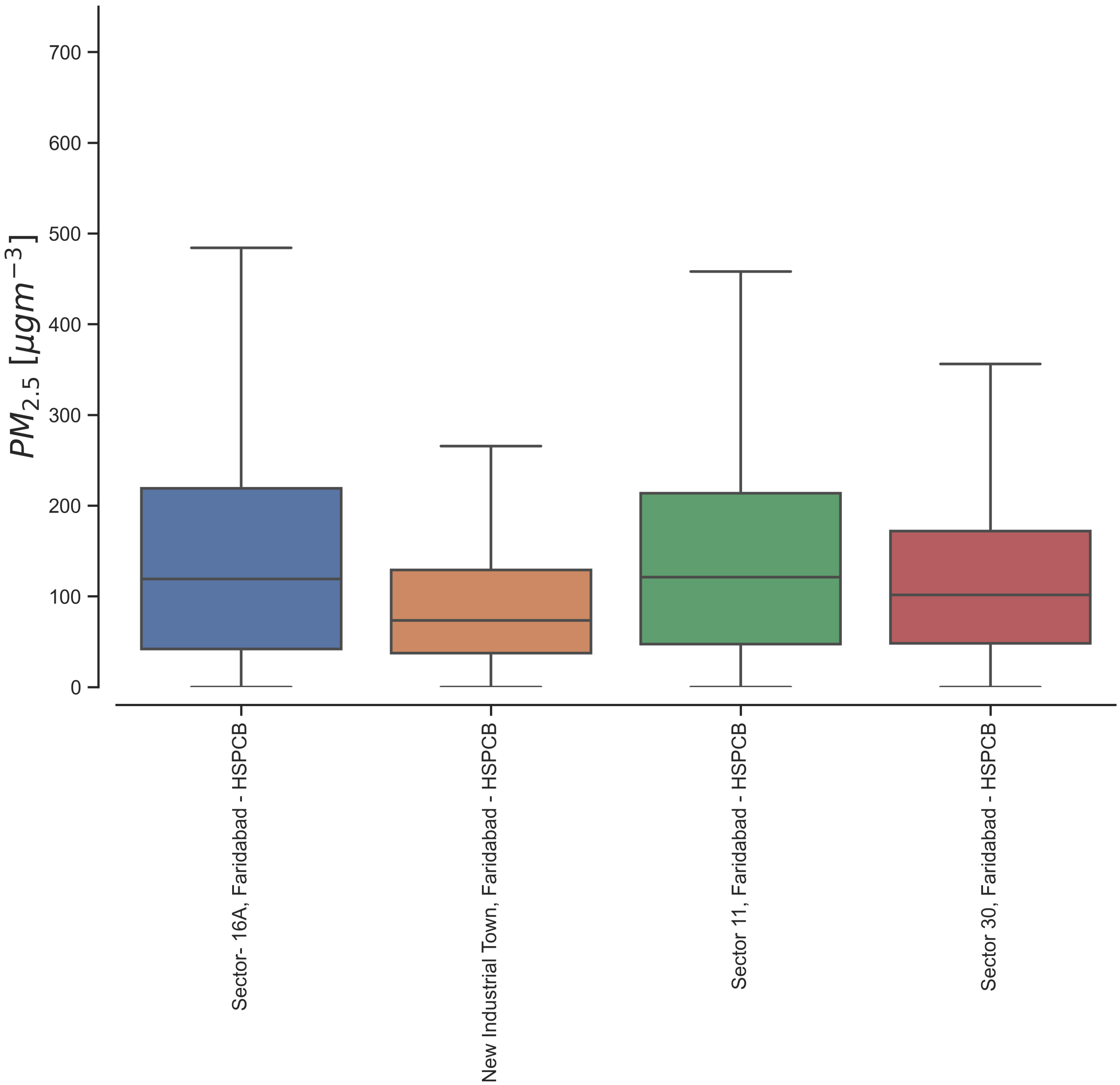


By splitting the data in each location, resampling the data to monthly and plotting the monthly data of each location, we can see the trend of PM2.5 value in 2021. In winter, PM2.5 value is much higher than other seasons, especially in November and December, PM2.5 value inceases dramatically.

Let’s go ahead and look at the distribution of PM2.5 values seen in `worst_city` by various sensors.

In [9]:

```
# Create the figure and axes
fig, ax = plt.subplots(1, figsize = (10, 7))

# Draw a box plot to show PM2.5 values according to location
ax = sns.boxplot(x = 'location',
                 y = 'value',
                 data = worst_city.query("value >= 0.0"),
                 fliersize = 0,
                 palette = 'deep',
                 ax = ax)

# Adjust the y-axis limits to tidy up the plot
ax.set_ylim([0, 750])

# Set the axis labels
ax.set_ylabel("$PM_{2.5}\;[\mu gm^{-3}]$", fontsize = 18)
ax.set_xlabel("")

# Tidy up the plot
sns.despine(offset = 10)
plt.xticks(rotation = 90)

# Display the plot
plt.show()
```



PART 2: The influence of Indian city size (population) on air pollution (PM2.5)

We focus on one aspect of city size and want to investigate the relationship between **PM2.5** and **city population**.

- Firstly, we create a dataframe called `df_pop` , containing city name, population and PM2.5 value.
- Secondly, we divide the PM2.5 pollution into six different levels, which are good, moderate, unhealthy for sensitive groups, unhealty, very unhealthy and hazardous.
- Finally, we fit a linear model to check the linear relationship between PM2.5 value and population.

In [10]:
```
# Import required packages
import requests
import scipy.stats as st
```

We write a function called `population(city_name)` , which takes one input argument `city_name` , a string representing the name of a city and returns its population.

In this function, **Geocoding API** is used to search for the population of the given city.

- If there is no result or no population data, it will return 'not found'.
- If there are many results, it will extract the population of the city belonging to India.

In [11]:
```
def population(city_name):
    '''
    Retrieves and displays the population of the given city.
    Input:
        city_name (string): a string representing the name of a city
    Output:
        population (int): a value representing the population of a given city
    '''

    # Request information from Geocoding API by using online URL builder interface
    # and parse the JSON data to a dictionary
    params_dict = {'name': city_name, 'count': 10}
    city_info = requests.get('https://geocoding-api.open-meteo.com/v1/search?', params = pa


    # Check whether there exists a list of results when serching the given city name in Geo
    # If the list exists, extract the result of Indian city in the list
    try:
        i = 0
        # For each result in the list, check if the city is in India
        while i < len(city_info['results']):
            if city_info['results'][i]['country'] == 'India':
                city_info = city_info['results'][i]
                break
            else:
                i += 1
    # If the list does not exist, let it be an empty dictionary
    except KeyError:
        city_info = {}


    # If the result of Indian city exists, check its population data exists
    if city_info != {}:
        # If the population data exists, let it be the value of population variable
        try:
            population = city_info['population']
        except KeyError:
            population = 'Not found'
    # If the result of Indian city dose not exist, let population be 'Not found'
    else:
        population = 'Not found'


    return population
```

In [12]:

```
# Create two empty lists for city name and population respectively
city_list = []
population_list = []

# Recreate the list 'IN_cities' using our saved 'geo_data' file
IN_cities = geo_data['city'].to_list()
IN_cities.remove('India') # We found that India appears in Indian cities incorrectly, so dr

# Check each city name in the city list
for city_name in IN_cities:
    # If the city has a population value, add its name and population into
    # the city name list and the population list respectively
    if population(city_name) != 'Not found':
        city_list.append(city_name)
        population_list.append(population(city_name))

# Create an empty list for PM2.5 values
pm25_list = []

# For each city with a population value, add its PM2.5 value into the PM2.5 list
for city_name in city_list:
    pm25_list.append(geo_data[geo_data['city'] == city_name]['mean'])

# Put the city names, population, PM2.5 values correspondingly into a dataframe
df_pop = pd.DataFrame({'City': city_list,
                       'Population': population_list,
                       'PM2.5': pm25_list})

# Export 'df_pop' to a csv file in order to avoid the influence of api real time updating
# outputpath = '/Users/wangyiyi/Desktop/pp-project-2-group-2/df_pop.csv'
# df_pop.to_csv(outputpath, sep = ',', index = False, header = False)

# Read the 'df_pop' file we have saved
col_name = ['City', 'Population', 'PM2.5']
df_pop = pd.read_csv('df_pop.csv', names = col_name, header = None)

# Show the dataframe
df_pop
```

Out[12]:

	City	Population	PM2.5
0	Agartala	203264	46.437188
1	Agra	1430055	122.526273
2	Ahmedabad	3719710	29.165598
3	Aizawl	265331	1.619722
4	Ajmer	517911	46.802632
...	...	...	...
123	Varanasi	1164404	116.687379
124	Visakhapatnam	1063178	66.256410
125	Vrindavan	60195	142.876154
126	Yadgir	65376	57.066667
127	Yamunanagar	208931	170.077143

128 rows × 3 columns

We use the function `population(city_name)` to get the population of 155 Indian cities in the list `IN_cities` and drop cities with no population data. We store the result city names and their correponding population data in lists `city_list` and `population_list` , repectively.

After getting all the cities with population data, we can get their PM2.5 mean values from our stored `geo_data` dataframe and store those values in a new list called `pm25_list` .

We can then put the three lists together, and we get a new data frame called `df_pop` , which has 128 rows and 3 columns:

- City: 128 Indian city names.
- Population: 128 population data.
- PM2.5: 128 PM2.5 mean values.

We write a function called `pol_level(value)` , which takes one input argument `value` , a float representing the mean value of PM2.5 and returns the pollution level of the city.

In this function, the pollution is divided into four different levels:

- No more than 12: good.
- 12.1 ~ 35.4: moderate.
- 35.5 ~ 55.4: unhealthy for sensitive groups.
- 55.5 ~ 150.4: unhealthy.
- 150.5 ~ 250.4: very unhealthy.
- More than 250.5: hazardous.

In [13]:
```
def pol_level(value):

    '''
    Retrieves and displays the pollution level of the city.
    Input:
        value (float): a float representing the mean value of PM2.5.
    Output:
        a string representing the pollution level of the city according to the given PM2.5
    '''

    # PM2.5 mean value no more than 12 indicates a good pollution level
    if value <= 12:
        return 'Good'

    # PM2.5 mean value between 12.1 and 35.5 indicates a moderate pollution level
    elif value > 12 and value <= 35.5:
        return 'Moderate'

    # PM2.5 mean value between 35.5 and 55.4 indicates an 'unhealthy for sensitive groups'
    elif value > 35.5 and value <= 55.4:
        return 'Unhealthy for Sensitive Groups'

    # PM2.5 mean value between 55.5 and 150.4 indicates an 'unhealthy' pollution level
    elif value > 55.4 and value <= 150.4:
        return 'Unhealthy'

    # PM2.5 mean value between 150.5 and 250.4 indicates an 'very unhealthy' pollution leve
    elif value > 150.4 and value <= 250.4:
        return 'Very unhealthy'

    # PM2.5 mean value larger than 250.5 indicates an 'hazardous' pollution level
    else:
        return 'Hazardous'
```

In [14]:
```
# Create an empty list for pollution level
level = []

# Recreate the 'pm25_list' using our saved 'df_pop' file
pm25_list = df_pop['PM2.5'].to_list()

# Using the PM2.5 values to check the pollution for 129 Indian cities
# and add them into the pollution level list
for i in pm25_list:
    level.append(pol_level(i))


# Add a new column representing the pollution level to the previous data frame
df_pop['Pollution level'] = level

# Show the modified dataframe
df_pop
```

Out[14]:

	City	Population	PM2.5	Pollution level
0	Agartala	203264	46.437188	Unhealthy for Sensitive Groups
1	Agra	1430055	122.526273	Unhealthy
2	Ahmedabad	3719710	29.165598	Moderate
3	Aizawl	265331	1.619722	Good
4	Ajmer	517911	46.802632	Unhealthy for Sensitive Groups
...	...	...	...	...
123	Varanasi	1164404	116.687379	Unhealthy
124	Visakhapatnam	1063178	66.256410	Unhealthy
125	Vrindavan	60195	142.876154	Unhealthy
126	Yadgir	65376	57.066667	Unhealthy
127	Yamunanagar	208931	170.077143	Very unhealthy

128 rows × 4 columns

According to the PM2.5 value, we use the function `pol_level(value)` to get the pollution level of 128 Indian cities. Then we get a new list called `level` and modify the previous data frame by adding the fourth column:

- Pollution level: the pollution level of 128 Indian cities according to their PM2.5 values.

Then our modified dataframe `df_pop` has 128 rows and 4 columns.

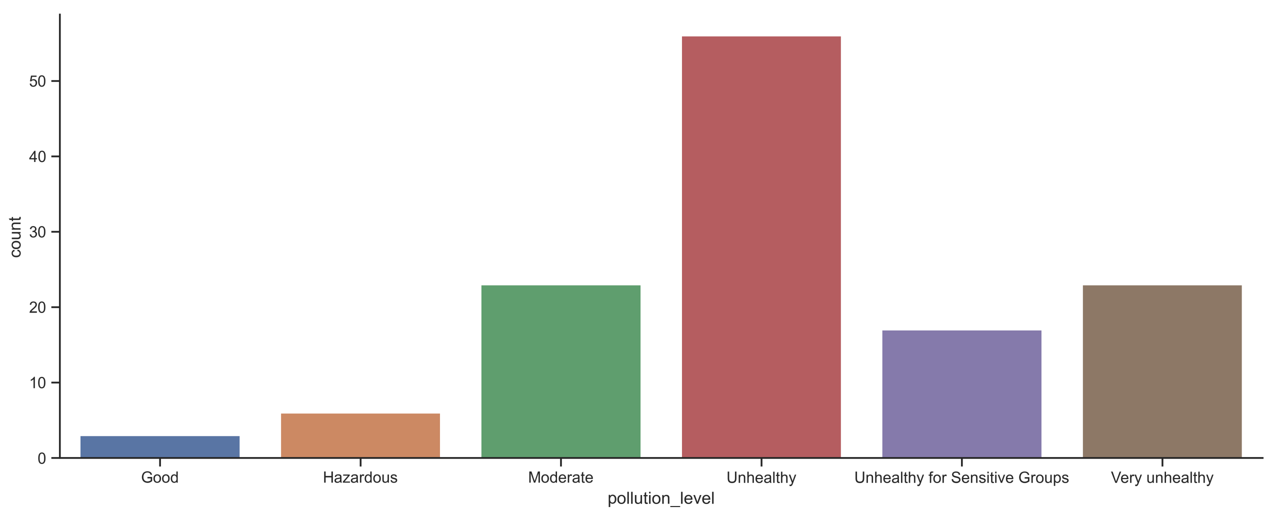
In [15]:
```
# Get the statistics on a per-level basis according to city
df_level = df_pop.groupby('Pollution level')['City'].describe()

# Get the index into a list
pollution_level = df_level.index.values

# Add the list to the dataframe
df_level['pollution_level'] = pollution_level

# Show the dataframe
df_level
```

Out[15]:

	count	unique	top	freq	pollution_level
Pollution level					
Good	3	3	Madikeri	1	Good
Hazardous	6	6	Buxar	1	Hazardous
Moderate	23	23	Pune	1	Moderate
Unhealthy	56	56	Jaipur	1	Unhealthy
Unhealthy for Sensitive Groups	17	17	Kolar	1	Unhealthy for Sensitive Groups
Very unhealthy	23	23	Chhapra	1	Very unhealthy

In [16]:
```
# Draw a bar plot to show the number of Indian cities with different pollution level
sns.catplot(data = df_level,
            x = 'pollution_level',
            y = 'count',
            kind = 'bar',
            aspect = 2.5)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x7ff17e1a22b0>



In [17]:
```
# Get the statistics on a per-level basis according to PM2.5 value
df_pop.groupby('Pollution level')['PM2.5'].describe()
```

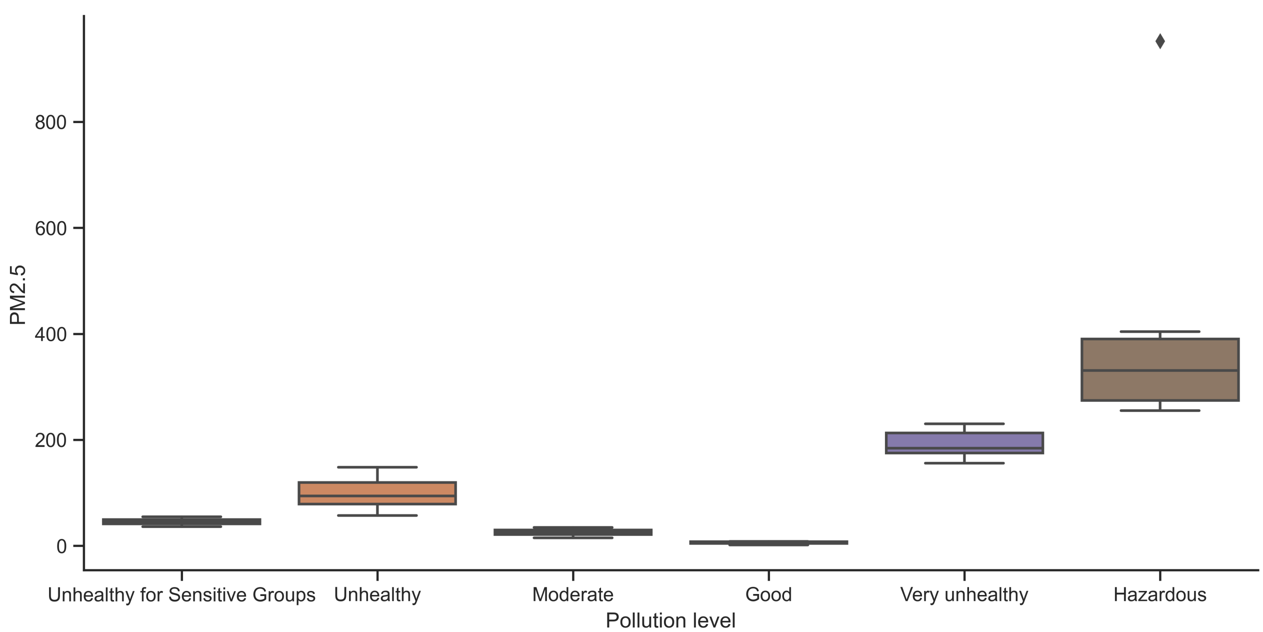
Out[17]:

Pollution level	count	mean	std	min	25%	50%	75%	max
Good	3.0	5.777112	3.617312	1.619722	4.563104	7.506486	7.855807	8.205128
Hazardous	6.0	422.402269	265.436607	255.200000	274.123001	330.916667	390.275508	952.126713
Moderate	23.0	25.809263	6.083930	14.870588	21.500652	25.635294	29.727132	34.706848
Unhealthy	56.0	98.753784	26.099257	57.066667	78.857200	93.991750	119.515247	148.103896
Unhealthy for Sensitive Groups	17.0	45.962597	5.619625	36.088108	41.310732	46.437188	49.715789	54.810811
Very unhealthy	23.0	190.477426	22.690576	156.030889	175.285833	184.230769	212.816667	230.027778

According to the `df_level` dateframe and the plot, we can find out that the around half of the city in India has a unhealthy PM2.5 level. Cities in good and moderate level only count for 26, representing only 20% of all the cities. So we can say that India is in a bad air condition.

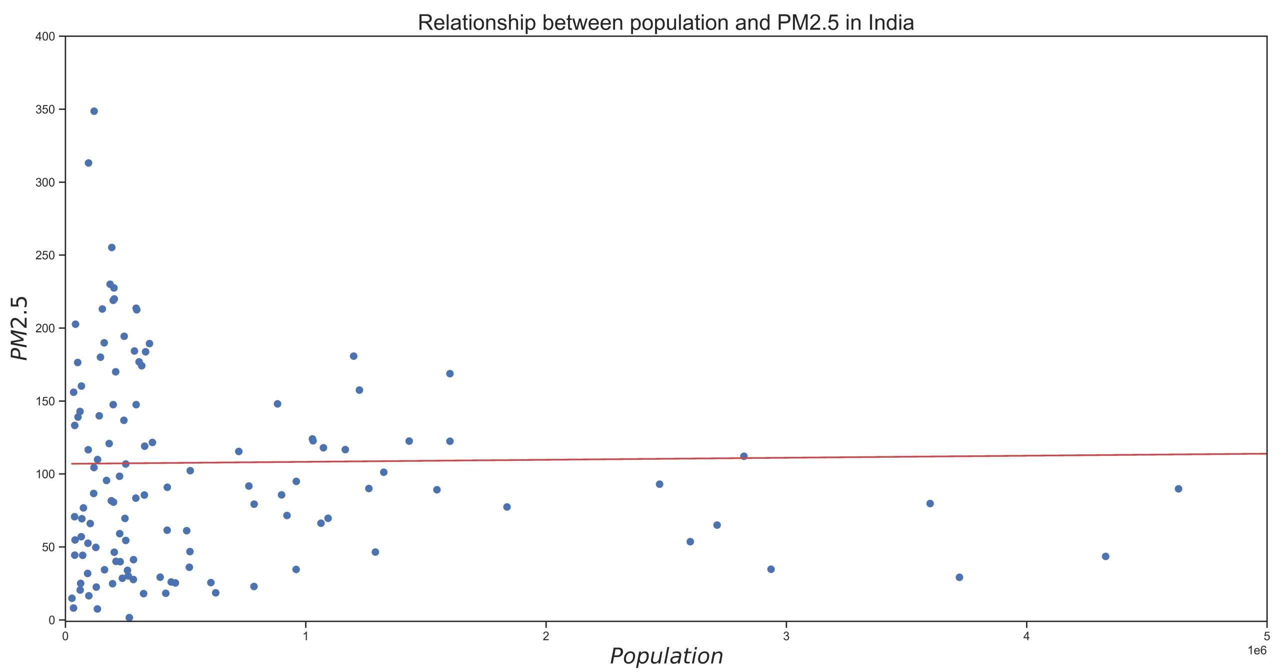
In [18]:
```
# Draw a box plot to show the statistical information of each polution level
sns.catplot(data = df_pop,
            x = 'Pollution level',
            y = 'PM2.5',
            kind = 'box',
            aspect = 2)
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x7ff17e2597f0>



As we have had the population data and mean PM2.5 values for Indian cities, we can check if there is any relationship between air quality and city size (i.e. population).

In [19]:
```
# Create the figure and axes
fig, ax = plt.subplots(figsize=(20, 10))

# Plot 'PM2.5' according to 'Population'
ax.plot(df_pop['Population'], df_pop['PM2.5'], 'bo')

# Label the figure
ax.set_title('Relationship between population and PM2.5 in India',fontsize = 20)
# Set the axis label
ax.set_xlabel(r'$Population$',fontsize = 20)
ax.set_ylabel(r'$PM2.5$',fontsize = 20)

# Fit a linear model and find the relationship between population and PM2.5
ln = st.linregress(df_pop['Population'], df_pop['PM2.5'])
y = ln.intercept + ln.slope * df_pop['Population']
# Plot the linear line on the figure
ax.plot(df_pop['Population'], y, 'r-')

# set limit to population and PM2.5
ax.set_xlim([0, 0.5e7])
ax.set_ylim([-1, 400])

# Display the plot
plt.show()
```



The fitted line is almost parallel, and the scatter is randomly distributed in the plot, so we can reach a conclusion that there is no significant relationship between population and PM2.5 value.

PART 3: The influence of weather conditions on air pollution (PM2.5) in Delhi

Some scholars believe that when the pollution source is relatively stable, meteorological conditions play a dominant role in the particle concentration. For example, temperature can affect particle formation because high temperature can promote the photochemical reaction between precursors. Furthermore, recent studies indicated that wind speed seems to play an influential role in modulating ground surface PM2.5 concentration.

For this part, we will explore the effect of weather conditions on air pollution (PM2.5) over the year 2021 in Delhi, one of the megacities in India. And we'll mainly focus on the following meteorological factors:

- Average temperature
- Precipitation
- Average wind speed

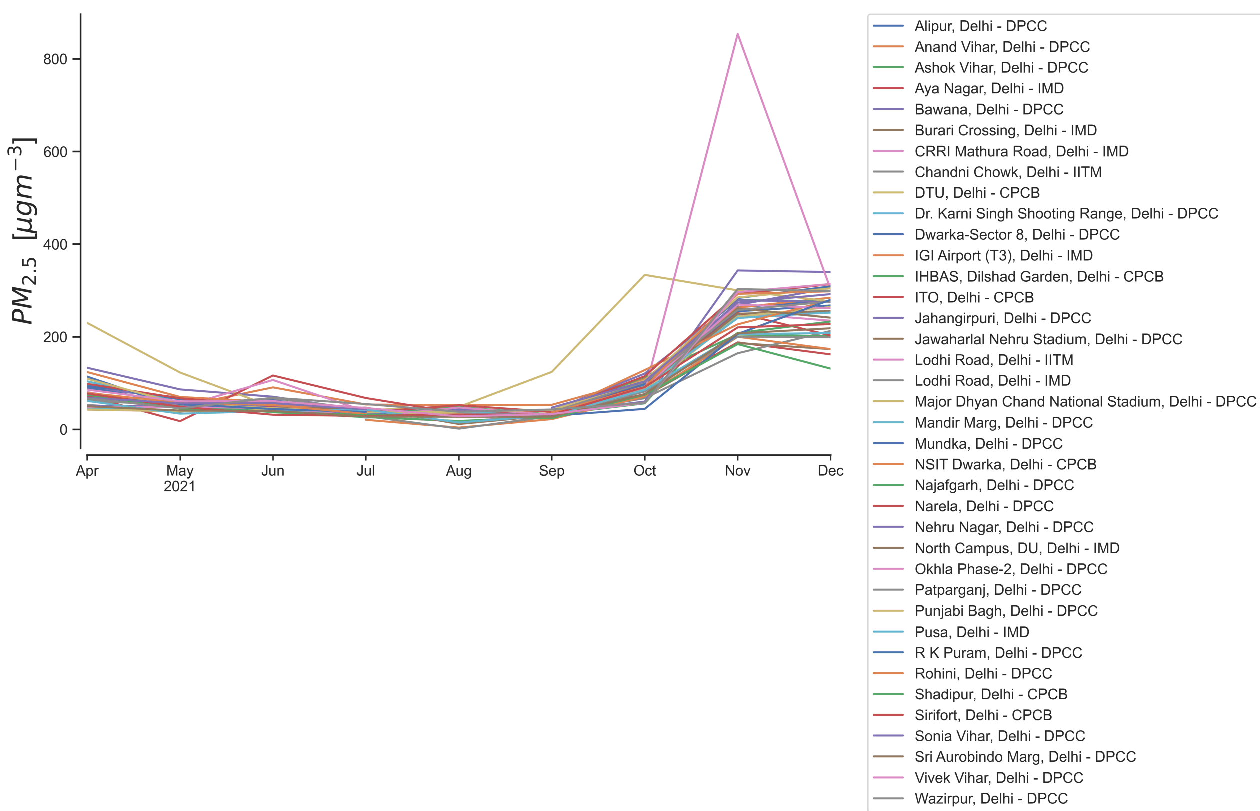
In [20]:
```
# Import required packages
from datetime import datetime
from meteostat import Stations, Daily
```

In [21]:
```
# Grab the past 100000 data points for PM2.5 values in Delhi.
Delhi_data = api.measurements(city = 'Delhi', parameter = 'pm25', limit = 100000, df = True
                              date_from = datetime(2021, 1, 1), date_to = datetime(2021, 12, 31))

# Export Delhi_data to a csv file in order to avoid the influence of api real time updating
# outputpath='/Users/wangyiyi/Desktop/pp-project-2-group-2/Delhi_data.csv'
# Delhi_data.to_csv(outputpath,sep=',',index=False,header=False)

# Let's first read PM2.5 data of Delhi from 'Delhi_data.csv'.
Delhi_data = pd.read_csv('Delhi_data.csv',
                         sep = ',',
                         names = ['location', 'parameter', 'value', 'unit', 'country', 'cit
                                  'date.utc', 'coordinates.latitude', 'coordinates.longitud

# Change the dtype of 'date.utc' back from object to datetime.
Delhi_data['date.utc'] = pd.to_datetime(Delhi_data['date.utc'])

# Ensure the dateframe has DatetimeIndex, not RangeIndex.
Delhi_data = Delhi_data.set_index(Delhi_data['date.utc'])

# Show the information of the dataframe
Delhi_data.info()
```

```
FutureWarning: pandas.io.json.json_normalize is deprecated, use pandas.json_normalize ins
tead

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 101198 entries, 2021-12-02 20:15:00+00:00 to 2021-04-02 13:30:00+00:00
Data columns (total 9 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   location               101198 non-null  object
 1   parameter              101198 non-null  object
 2   value                  101198 non-null  float64
 3   unit                   101198 non-null  object
 4   country                101198 non-null  object
 5   city                   101198 non-null  object
 6   date.utc               101198 non-null  datetime64[ns, UTC]
 7   coordinates.latitude   101198 non-null  float64
 8   coordinates.longitude  101198 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(3), object(5)
memory usage: 7.7+ MB
```

After loading the data needed, we plot the following figure to look at the distribution of **PM2.5** values seen in **Delhi** by various sensors. To do this, we first group the whole dataset by sensor locations, and then resample positive values of PM2.5 for each location to monthly.

In [22]:
```
# Create the figure and axes
fig, ax = plt.subplots(1, figsize = (10, 6))

for group, df in Delhi_data.groupby('location'):
    # Query the data to only get positive values and resample to monthly
    _df = df.query("value >= 0.0").resample('1m').mean()
    _df.value.plot(ax = ax, label = group)

# Set the legend and the axis label
ax.legend(loc = 'best')
ax.set_ylabel("$PM_{2.5}$  [$\mu g m^{-3}$]", fontsize = 20)
ax.set_xlabel("")
sns.despine(offset = 5)
plt.legend(bbox_to_anchor = (1.05, 1), loc = 2, borderaxespad = 0.)

# Display the plot
plt.show()
```



According to the plot above, we found a significant increase in PM2.5 values at all testing sites starting in September, and this upward trend continued into December.

Now, let's check out the weather conditions in Delhi over the year. To obtain historical weather data in Delhi, we will need the help of `Meteostat` Python library, which provides simple access to open weather and climate data using `Pandas` . We will mainly use the following classes provided in `Meteostat` :

- `Stations` - a simple interface to query weather stations using several filters..

- `Daily` - query daily weather data for one or multiple weather stations or a single geographical point.

To begin with, we find a weather station together with its station code in Delhi simply by providing the coordinates of Delhi in `nearby` method. And then pass the weather station identifiers returned by `fetch` method to get daily weather data needed. Once we've got all data points needed, we can plot them out.

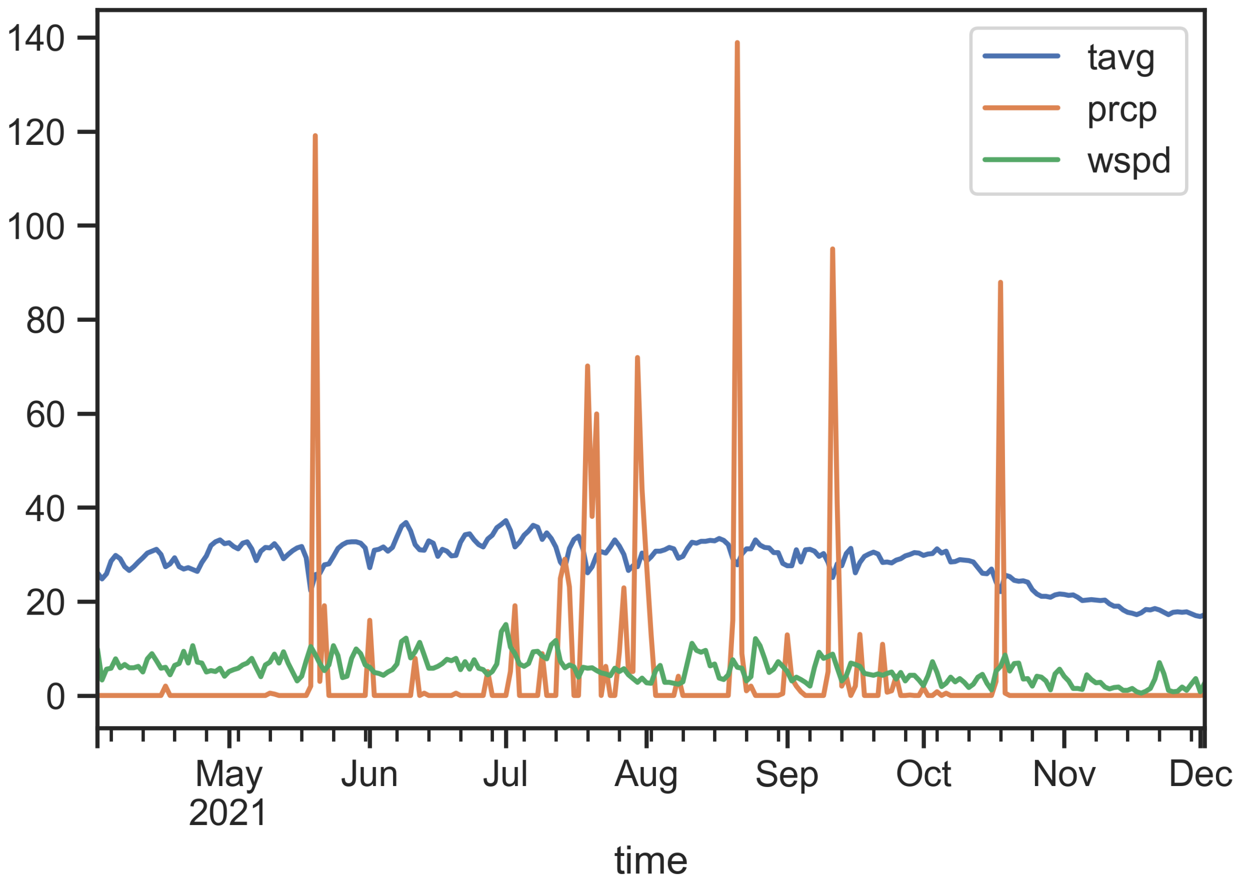
In [23]:

```
# Get the latitude and longitude of Delhi
latitude = Delhi_data['coordinates.latitude'][0]
longitude = Delhi_data['coordinates.longitude'][0]

# Get nearby weather stations
stations = Stations()
stations = stations.nearby(latitude, longitude)
station = stations.fetch(1)

station_code = station.index[0]
```

In [24]:

```
# Get the statistics on a per-location basis
Delhi_data.groupby(['location'])['value'].describe()

# Query the data to only get positive values and resample to daily
Delhi_data2 = Delhi_data.query("value >= 0.0").resample('1d').mean()

# Get the start date and end date of the data
start_month = Delhi_data2.index[0].month
start_day = Delhi_data2.index[0].day
end_month = Delhi_data2.index[-1].month
end_day = Delhi_data2.index[-1].day
```

In [25]:
```
# Set time period
start = datetime(2021, start_month, start_day)
end = datetime(2021, end_month, end_day)

# Get daily data
data = Daily(station_code, start, end)
data = data.fetch()

# Plot the line chart including average temperature, precipitation, average wind speed
data.plot(y = ['tavg', 'prcp', 'wspd'])

# Display the plot
plt.show()
```



As demonstrated in the plot above, we find that

- for average temperature, it does not vary much throughout the year in delhi, but it can be colder in the winter time;
- precipitation throughout the year is concentrated in summer, whilst it is almost close to zero in winter;
- the average wind speed hardly fluctuates throughout the year, and is only slightly lower in the winter months.

After visualizing data of PM2.5 and weather individually, we now try to fit a statistical relation between them for better inspection.

In [26]:

```
# Get precipitation, average temperature, average wind speed and PM2.5 value into lists
pcrp = data['prcp'].tolist()
tavg = data['tavg'].tolist()
wspd = data['wspd'].tolist()
PM25 = Delhi_data2['value'].tolist()

# Get the desired dataframe
weather_data = pd.DataFrame({'PM2.5': PM25,
                             'precipitation': pcrp,
                             'average air temperature': tavg,
                             'average wind speed': wspd})

# Show the dataframe
weather_data
```

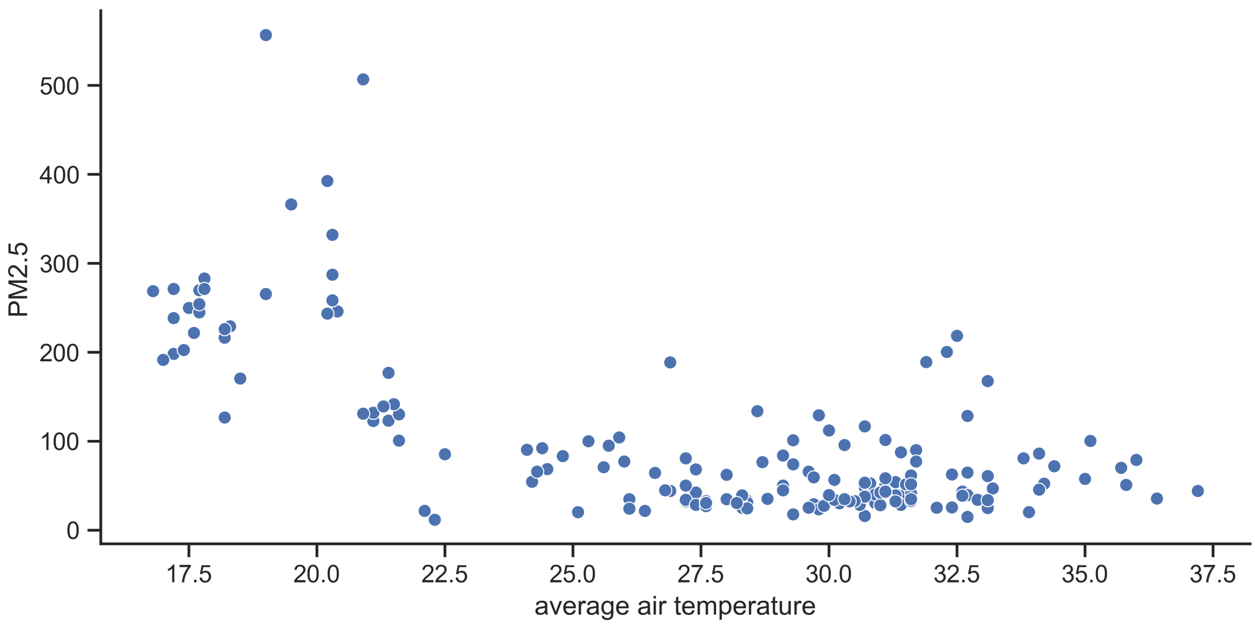
Out[26]:

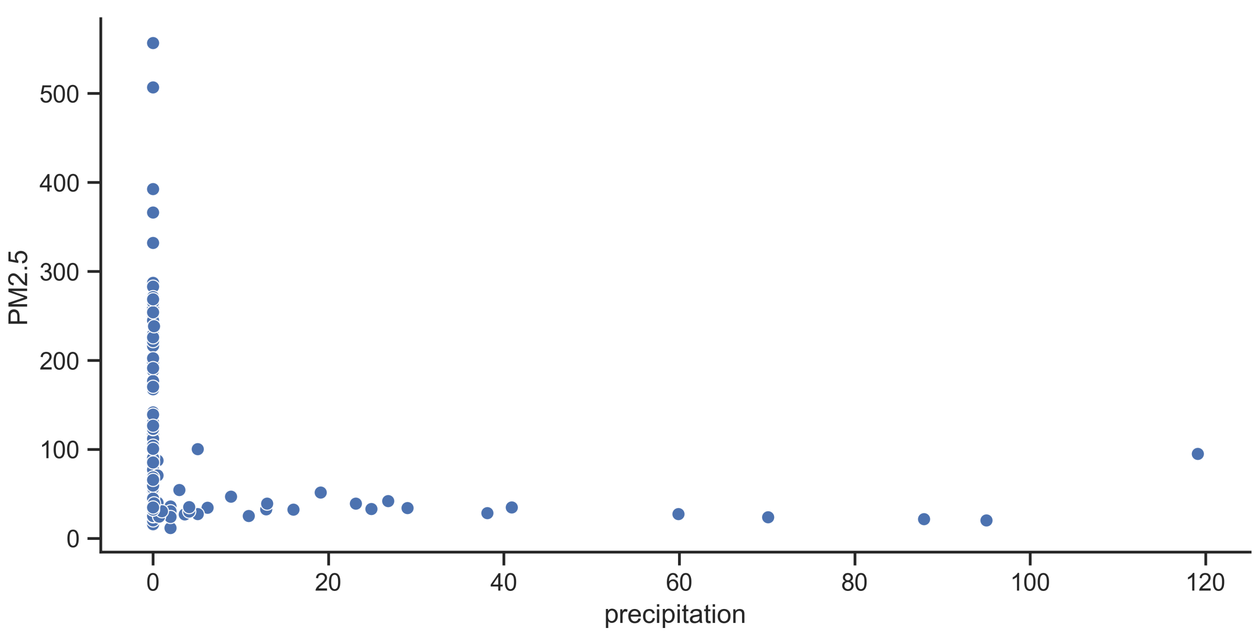
	PM2.5	precipitation	average air temperature	average wind speed
0	34.829091	0.0	26.1	9.9
1	83.285556	0.0	24.8	3.3
2	NaN	0.0	25.9	5.6
3	133.911786	0.0	28.6	5.8
4	129.127647	0.0	29.8	7.8
...	...	...	...	...
240	271.378591	0.0	17.8	1.1
241	202.523078	0.0	17.4	2.4
242	191.362754	0.0	17.0	3.6
243	268.717597	0.0	16.8	0.8
244	238.465397	0.1	17.2	2.7

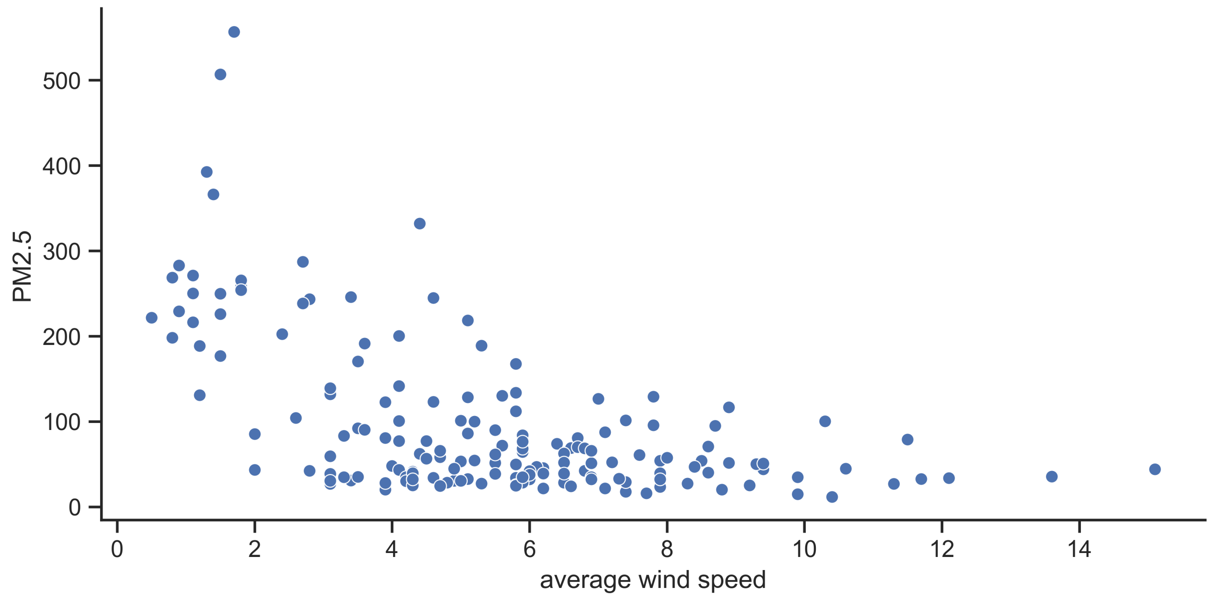
245 rows × 4 columns

In [27]:
```python
# Plot to see the relationship between weather measurements and PM2.5
sns.relplot(data = weather_data,
            x = 'average air temperature',
            y = 'PM2.5',
            height = 4,
            aspect = 2)

sns.relplot(data = weather_data,
            x = 'precipitation',
            y = 'PM2.5',
            height = 4,
            aspect = 2)

sns.relplot(data = weather_data,
            x = 'average wind speed',
            y = 'PM2.5',
            height  =4,
            aspect = 2)
```

Out[27]: <seaborn.axisgrid.FacetGrid at 0x7ff17d24df70>







By fitting the data of PM2.5 with 3 weather variables respectively, we get results listed below:

- The level of PM2.5 seems to have a negative trend with average temperature, which is weird. Because in a common sense, there should be a positive correlation between PM2.5 and temperature. Because high temperatures can promote photochemical reactions between precursors.
- There seems no particular relationship between precipitation and PM2.5. However, one thing to notice is that high levels of PM2.5 concentrate on zero precipitation. It is therefore reasonable to suspect that low rainfall is a favorable condition for the existence of high PM2.5.
- The third plot seems to infer a negative correlation between average wind speed and PM2.5. That is when the wind speed increases, the concentration of PM2.5 can be reduced.