## **NOISE POLLUTION MONITORING**

------------------------------------------------------------------------------------------------------------------------------

**Phase 4:Development part 2**

**Topic: Different activities like feature engineering and model training .**

Noise pollution monitoring involves the measurement and assessment of excessive or disruptive sounds in the environment. Here's a brief overview of its definition and designing:

1. **Definition**:

Noise pollution monitoring refers to the systematic process of measuring, recording, and analyzing noise levels in a given area to evaluate their impact on human health and the environment. It aims to identify sources of noise pollution, quantify noise levels, and assess compliance with noise regulations.

2. **Designing for Noise Pollution Monitoring:**

**a. Sensor Selection**: Choose appropriate noise sensors (e.g., microphones, sound level meters) capable of accurately capturing sound levels across various frequencies and decibel ranges.

**b. Data Acquisition System:** Develop a data acquisition system to collect, process, and store noise data. This may include analog-to-digital converters, microcontrollers, and data storage devices.

**c. Power Supply**: Ensure a stable power source for continuous monitoring. Consider using batteries, solar panels, or a reliable electrical connection.

**d. Data Transmission**: Implement a means to transmit data to a central database or monitoring station, such as wireless communication (Wi-Fi, cellular, or LoRa) or wired connections.

e. **Location Planning**: Strategically position monitoring devices to cover relevant areas, considering factors like proximity to noise sources and the community affected.

**f. Calibration:** Regularly calibrate sensors to maintain accuracy and reliability in noise level measurements.

**g. Data Analysis Software**: Develop or choose software for real-time data analysis, visualization, and reporting. This software should be capable of identifying noise trends and anomalies.

**h**. **Noise Mapping**: Create noise maps using geographical information systems (GIS) to visualize noise levels spatially and understand how noise pollution varies across different locations.

**i. Alerting System**: Implement an alerting mechanism to notify relevant authorities or individuals when noise levels exceed predefined thresholds or regulations.

j. **Maintenance Plan**: Establish a maintenance schedule for sensor upkeep, data system checks, and software updates.

k**. Privacy and Data Security**: Ensure data privacy and security measures are in place, especially if the monitoring system captures audio or location data.

l. **Community Engagement**: Involve local communities and stakeholders in the design process to gather input and address concerns.

**m.** **Regulatory Compliance**: Ensure that the monitoring system complies with local noise regulations and standards.

**n**. **Long-Term Sustainability**: Consider the long-term sustainability of the monitoring system, including funding, maintenance, and scalability.

Designing an effective noise pollution monitoring system involves a multidisciplinary approach, combining engineering, environmental science, data analysis, and community engagement to address the challenges associated with noise pollution. Noise pollution monitoring involves the measurement and assessment of excessive or disruptive sounds in the environment. Here's a brief overview of its definition and designing:

1. **Definition**:

Noise pollution monitoring refers to the systematic process of measuring, recording, and analyzing noise levels in a given area to evaluate their impact on human health and the environment. It aims to identify sources of noise pollution, quantify noise levels, and assess compliance with noise regulations.

2. **Designing for Noise Pollution Monitoring:**

**a. Sensor Selection**: Choose appropriate noise sensors (e.g., microphones, sound level meters) capable of accurately capturing sound levels across various frequencies and decibel ranges.

**b. Data Acquisition System:** Develop a data acquisition system to collect, process, and store noise data. This may include analog-to-digital converters, microcontrollers, and data storage devices.

**c. Power Supply**: Ensure a stable power source for continuous monitoring. Consider using batteries, solar panels, or a reliable electrical connection.

**d. Data Transmission**: Implement a means to transmit data to a central database or monitoring station, such as wireless communication (Wi-Fi, cellular, or LoRa) or wired connections.

e. **Location Planning**: Strategically position monitoring devices to cover relevant areas, considering factors like proximity to noise sources and the community affected.

**f. Calibration:** Regularly calibrate sensors to maintain accuracy and reliability in noise level measurements.

**g. Data Analysis Software**: Develop or choose software for real-time data analysis, visualization, and reporting. This software should be capable of identifying noise trends and anomalies.

**h**. **Noise Mapping**: Create noise maps using geographical information systems (GIS) to visualize noise levels spatially and understand how noise pollution varies across different locations.

**i. Alerting System**: Implement an alerting mechanism to notify relevant authorities or individuals when noise levels exceed predefined thresholds or regulations.

j. **Maintenance Plan**: Establish a maintenance schedule for sensor upkeep, data system checks, and software updates.

k**. Privacy and Data Security**: Ensure data privacy and security measures are in place, especially if the monitoring system captures audio or location data.

l. **Community Engagement**: Involve local communities and stakeholders in the design process to gather input and address concerns.

**m.** **Regulatory Compliance**: Ensure that the monitoring system complies with local noise regulations and standards.

**n**. **Long-Term Sustainability**: Consider the long-term sustainability of the monitoring system, including funding, maintenance, and scalability.

Designing an effective noise pollution monitoring system involves a multidisciplinary approach, combining engineering, environmental science, data analysis, and community engagement to address the challenges associated with noise pollution.

**Program for Noise Pollution Moniterig**

const int pingPin = 7;

const int red=11;

const int blue=10;b

int green=9;

void setup() {

// initialize serial communication:

Serial.begin(9600);

pinMode(red,OUTPUT);

pinMode(blue,OUTPUT);

pinMode(green,OUTPUT);

pinMode(3, OUTPUT);

}

void loop()

{

digitalWrite(3, HIGH);

delay(1000); // Wait for 1000 millisecond(s)

digitalWrite(3, LOW);

delay(1000); // Wait for 1000 millisecond(s)

// establish variables for duration of the ping, and the distance result

// in inches and centimeters:

long duration, inches, cm;

// The PING))) is triggered by a HIGH pulse of 2 or more microseconds.

// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

pinMode(pingPin, OUTPUT);

digitalWrite(pingPin, LOW);

delayMicroseconds(2);

digitalWrite(pingPin, HIGH);

delayMicroseconds(5);

digitalWrite(pingPin, LOW);

// The same pin is used to read the signal from the PING))): a HIGH pulse

// whose duration is the time (in microseconds) from the sending of the ping

// to the reception of its echo off of an object.

pinMode(pingPin, INPUT);

duration = pulseIn(pingPin, HIGH);

// convert the time into a distance

inches = microsecondsToInches(duration);

cm = microsecondsToCentimeters(duration);

Serial.print(inches);

Serial.print("in, ");

Serial.print(cm);

Serial.print("cm");

Serial.println();

if(cm<256){

analogWrite(red,cm);

analogWrite(blue,255-cm);

analogWrite(green,inches);

}

else{

analogWrite(red,0);

analogWrite(blue,0);

analogWrite(green,0);}

delay(100);

}

long microsecondsToInches(long microseconds) {

// According to Parallax's datasheet for the PING))), there are 73.746

// microseconds per inch (i.e. sound travels at 1130 feet per second).

// This gives the distance travelled by the ping, outbound and return,

// so we divide by 2 to get the distance of the obstacle.

// See: <http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf>

return microseconds / 74 / 2;

}

long microsecondsToCentimeters(long microseconds) {

// The speed of sound is 340 m/s or 29 microseconds per centimeter.

// The ping travels out and back, so to find the distance of the object we

// take half of the distance travelled.

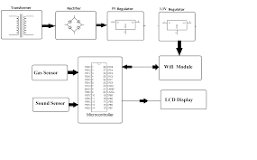
return microseconds / 29 / 2;

}

**Introduction:**

Greetings from the Kaggle bot! This is an automatically-generated kernel with starter code demonstrating how to read in the data and begin exploring. If you're inspired to dig deeper, click the blue "Fork Notebook" button at the top of this kernel to begin editing.

**Monitor noise pollution in IoT**



System uses air sensors to sense presence of harmful gases/compounds in the air and constantly transmit this data to microcontroller. Also system keeps measuring sound level and reports it to the online server over IOT. The sensors interact with microcontroller which processes this data and transmits it over internet.

## **Exploratory Analysis**

To begin this exploratory analysis, first import libraries and define functions for plotting the data using matplotlib. Depending on the data, not all plots will be made. (Hey, I'm just a simple kerneling bot, not a Kaggle Competitions Grandmaster!)

In [1]:

from mpl\_toolkits.mplot3d import Axes3D  
from sklearn.preprocessing import StandardScaler  
import matplotlib.pyplot as plt *# plotting*  
*import* numpy as np *# linear algebra*  
*import* os *# accessing directory structure*  
*import* pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

There are 2 csv files in the current version of the dataset:

In [2]:

for dirname, \_, filenames **in** os.walk('/kaggle/input'):  
 for filename **in** filenames:  
 print(os.path.join(dirname, filename))

/kaggle/input/stations.csv  
/kaggle/input/station\_month.csv

The next hidden code cells define functions for plotting data. Click on the "Code" button in the published kernel to reveal the hidden code.

unfold\_moreShow hidden code

unfold\_moreShow hidden code

unfold\_moreShow hidden code

Now you're ready to read in the data and use the plotting functions to visualize the data.

### Let's check 1st file: /kaggle/input/station\_month.csv

In [6]:

nRowsRead = 1000 *# specify 'None' if want to read whole file*  
*# station\_month.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows*  
*df1* = pd.read\_csv('/kaggle/input/station\_month.csv', delimiter=',', nrows = nRowsRead)  
df1.dataframeName = 'station\_month.csv'  
nRow, nCol = df1.shape  
print(f'There are **{nRow}** rows and **{nCol}** columns')

There are 840 rows and 7 columns

Let's take a quick look at what the data looks like:

In [7]:

df1.head(5)

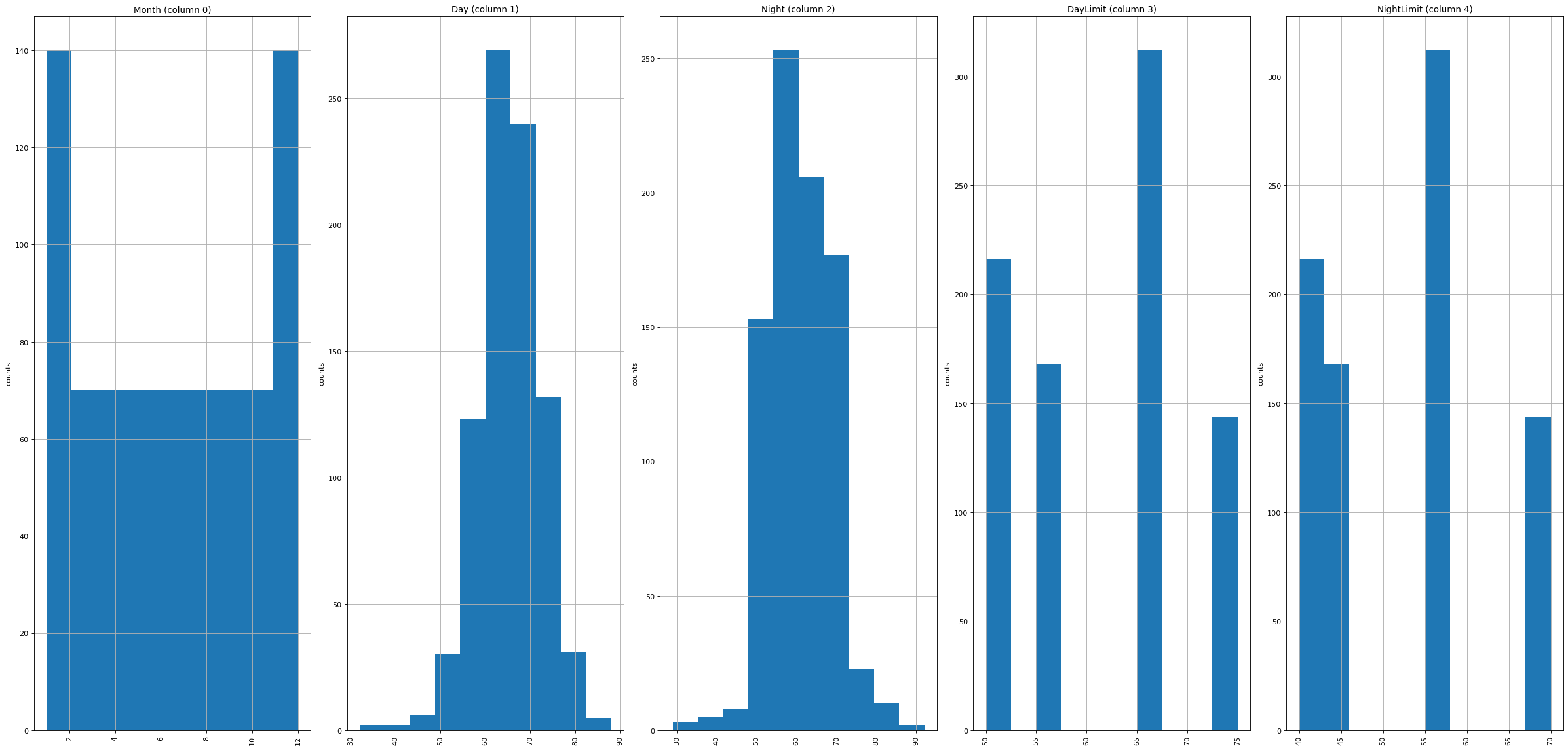
Out[7]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Station | Year | Month | Day | Night | DayLimit | NightLimit |
| 0 | DEL01 | 2018 | 1 | 53 | 52 | 50 | 40 |
| 1 | DEL01 | 2018 | 2 | 53 | 53 | 50 | 40 |
| 2 | DEL01 | 2018 | 3 | 53 | 53 | 50 | 40 |
| 3 | DEL01 | 2018 | 4 | 55 | 58 | 50 | 40 |
| 4 | DEL01 | 2018 | 5 | 55 | 57 | 50 | 40 |

**Distribution graphs (histogram/bar graph) of sampled columns:**

In [8]:

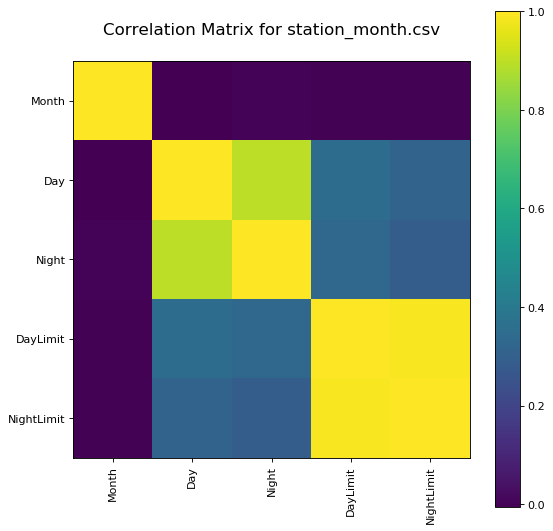
plotPerColumnDistribution(df1, 10, 5)



Correlation matrix:

In [9]:

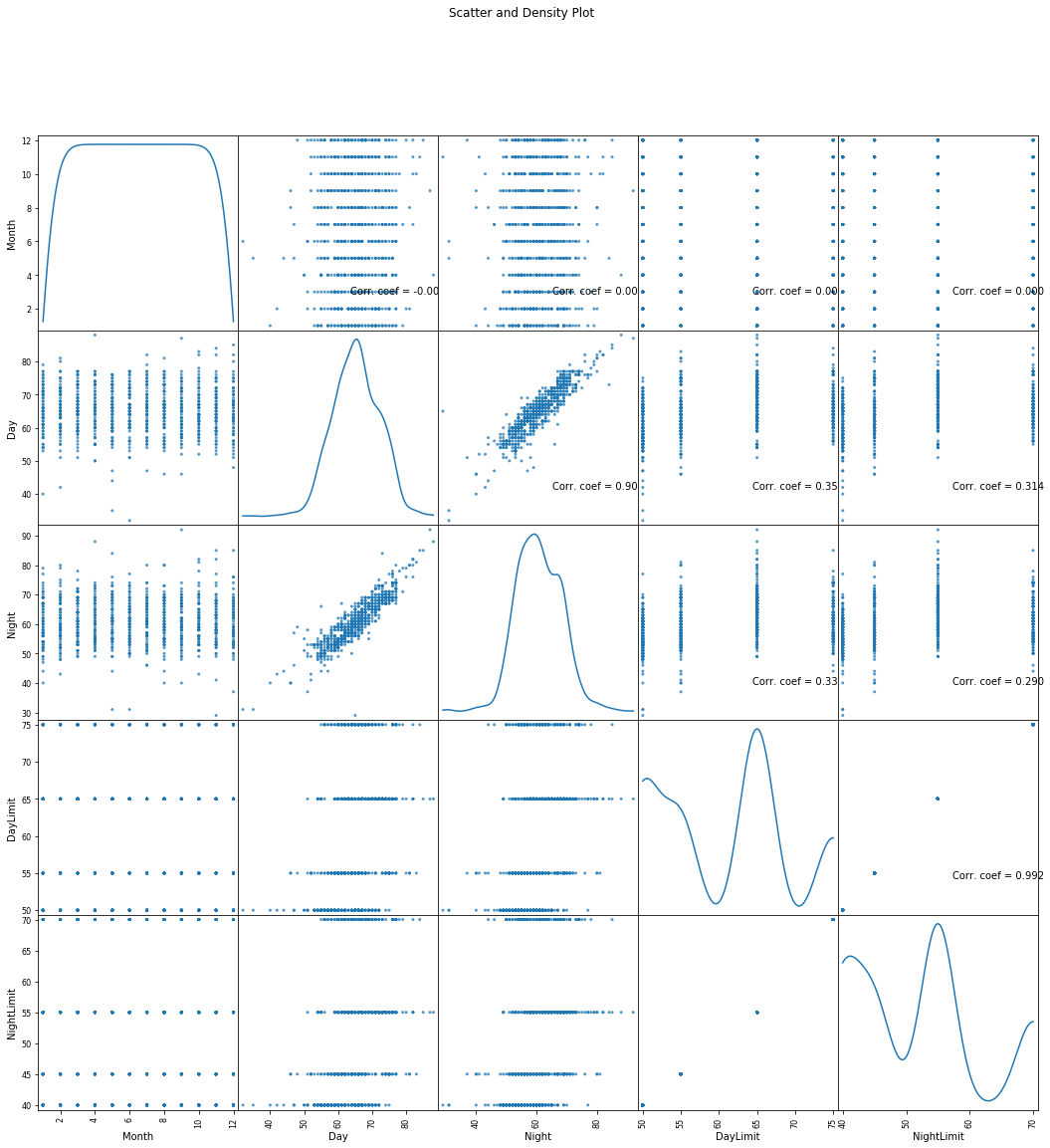
plotCorrelationMatrix(df1, 8)



Scatter and density plots:

In [10]:

plotScatterMatrix(df1, 18, 10)



### Let's check 2nd file: /kaggle/input/stations.csv

In [11]:

nRowsRead = 1000 *# specify 'None' if want to read whole file*  
*# stations.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows*  
*df2* = pd.read\_csv('/kaggle/input/stations.csv', delimiter=',', nrows = nRowsRead)  
df2.dataframeName = 'stations.csv'  
nRow, nCol = df2.shape  
print(f'There are **{nRow}** rows and **{nCol}** columns')

There are 70 rows and 5 columns

Let's take a quick look at what the data looks like:

In [12]:

df2.head(5)

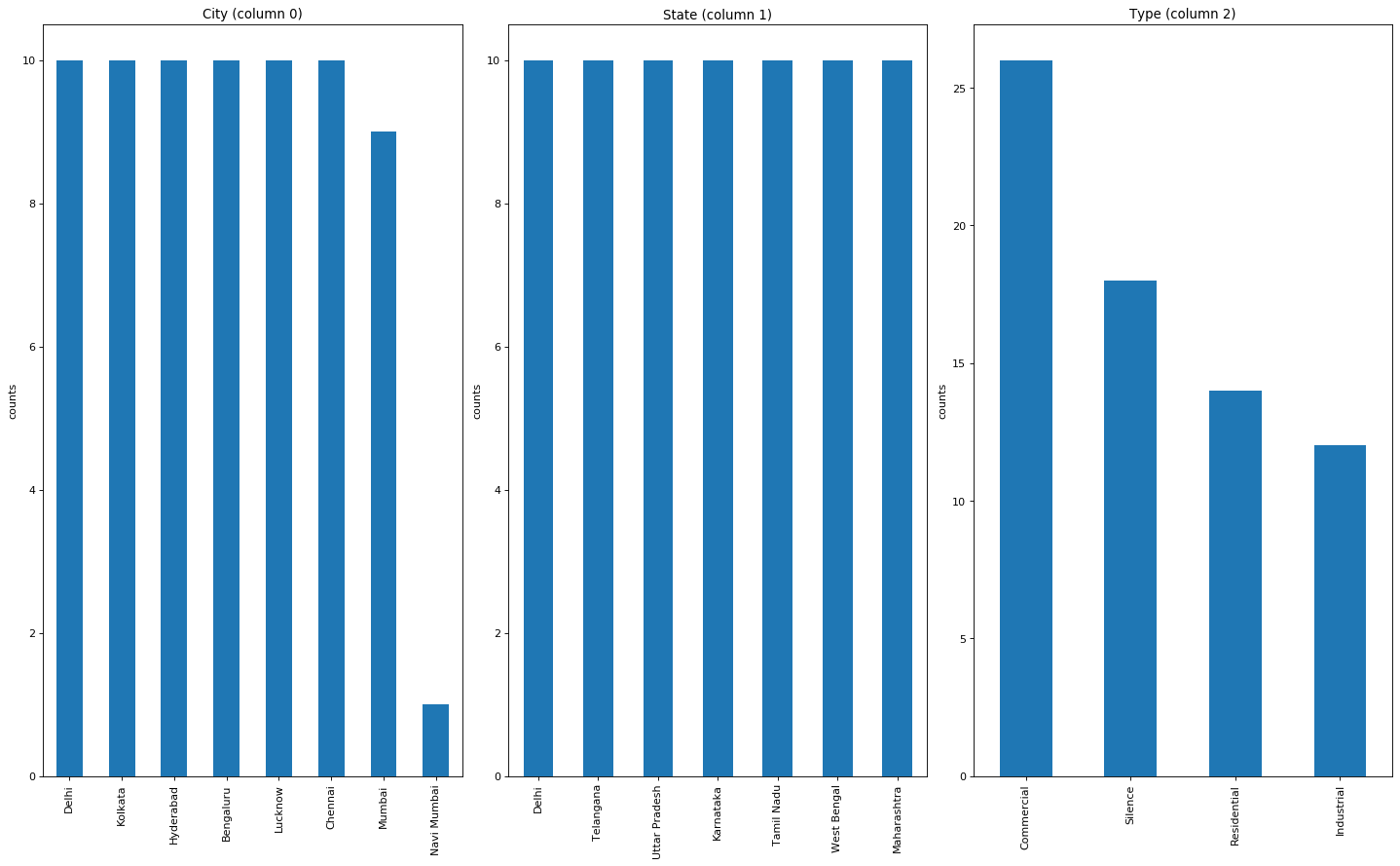
Out[12]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Station | Name | City | State | Type |
| 0 | DEL01 | Dilshad Garden | Delhi | Delhi | Silence |
| 1 | DEL02 | CPCB, HQ | Delhi | Delhi | Commercial |
| 2 | DEL03 | DCE, Bawana | Delhi | Delhi | Silence |
| 3 | DEL04 | ITO | Delhi | Delhi | Commercial |
| 4 | DEL05 | NSIT, Dwarka | Delhi | Delhi | Silence |

Distribution graphs (histogram/bar graph) of sampled columns:

In [13]:

plotPerColumnDistribution(df2, 10, 5)



Code :

from machine import Pin, ADC

from time import sleep

pot = ADC(Pin(2))

pot.atten(ADC.ATTN\_11DB) #Full range: 3.3v

#ADC.ATTN\_0DB: Maximum voltage of 1.2V

#ADC.ATTN\_2\_5DB: Maximum voltage of 1.5V

#ADC.ATTN\_6DB: Maximum voltage of 2.0V

#ADC.ATTN\_11DB: Maximum voltage of 3.3V

while True:

pot\_value = pot.read()

print(pot\_value)

sleep(0.1)

'''

import machine, time

a = machine.ADC(machine.Pin(32))

while True:

sample = a.read() # we want 16 bits, a.read() returns 10 bits

print(sample)

time.sleep(1/44100)

## Certainly! Different activities are involved in the process of developing and deploying machine learning models. These activities include:

## 

## **1. Data Collection:** Gathering and obtaining relevant data for your machine learning project. This can involve web scraping, using APIs, or working with pre-existing datasets.

## 

## **2.** **Preprocessing**: Cleaning and preparing the data for analysis. This may include handling missing values, removing outliers, and converting data types.

## 

## **3. Feature Engineering:**Creating new features or transforming existing ones to make them more suitable for model training. This can involve techniques like one-hot encoding, feature scaling, and dimensionality reduction.

## 

## **4. Data Splitting:** Dividing the dataset into training, validation, and test sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the test set is used for the final model evaluation.

## 

## **5.Model Selection:**Choosing an appropriate machine learning algorithm or model architecture based on the problem you are trying to solve. This could be decision trees, neural networks, support vector machines, etc.

## 

## **6. Model Training:** Using the training data to train the chosen machine learning model. This involves optimization algorithms that adjust the model's parameters to minimize the error.

## 

## **7. Hyperparameter Tuning:** Fine-tuning the model's hyperparameters to improve its performance. Techniques like grid search or random search can be used.

## 

## **8. Model Evaluation:**Assessing the model's performance on the validation or test data using appropriate metrics. Common evaluation metrics include accuracy, precision, recall, F1 score, and mean squared error, among others.

## 

## **9. Model Deployment:** Integrating the trained model into a production environment where it can make predictions on new data. This may involve creating APIs, containers, or web applications to serve the model.

## 

## **10. Monitoring and Maintenance:**Continuously monitoring the deployed model's performance, retraining it with new data, and maintaining it to ensure it remains accurate and relevant over time.

## 

## **11.Explainability and Interpretability:** Understanding and explaining how the model makes predictions, especially in critical applications where model decisions need to be justified.

## 

## **12. Documentation:** Creating clear and comprehensive documentation for the entire machine learning pipeline to facilitate collaboration and future work.

## 

## **13. Ethical Considerations:** Addressing ethical issues related to data collection, model biases, fairness, and privacy throughout the project.

## 

## Each of these activities is crucial for the successful development and deployment of machine learning models, and they often require careful planning and iteration to achieve the desired results.

**Programe:**

import machine

import time

import urequests

import ujson

import network

import math

# Define your Wi-Fi credentials

wifi\_ssid = 'Wokwi-GUEST'

wifi\_password = '' # Replace with the actual Wi-Fi password

# Connect to Wi-Fi

wifi = network.WLAN(network.STA\_IF)

wifi.active(True)

wifi.connect(wifi\_ssid, wifi\_password)

# Wait for Wi-Fi connection

while not wifi.isconnected():

pass

# Define ultrasonic sensor pins (Trig and Echo pins)

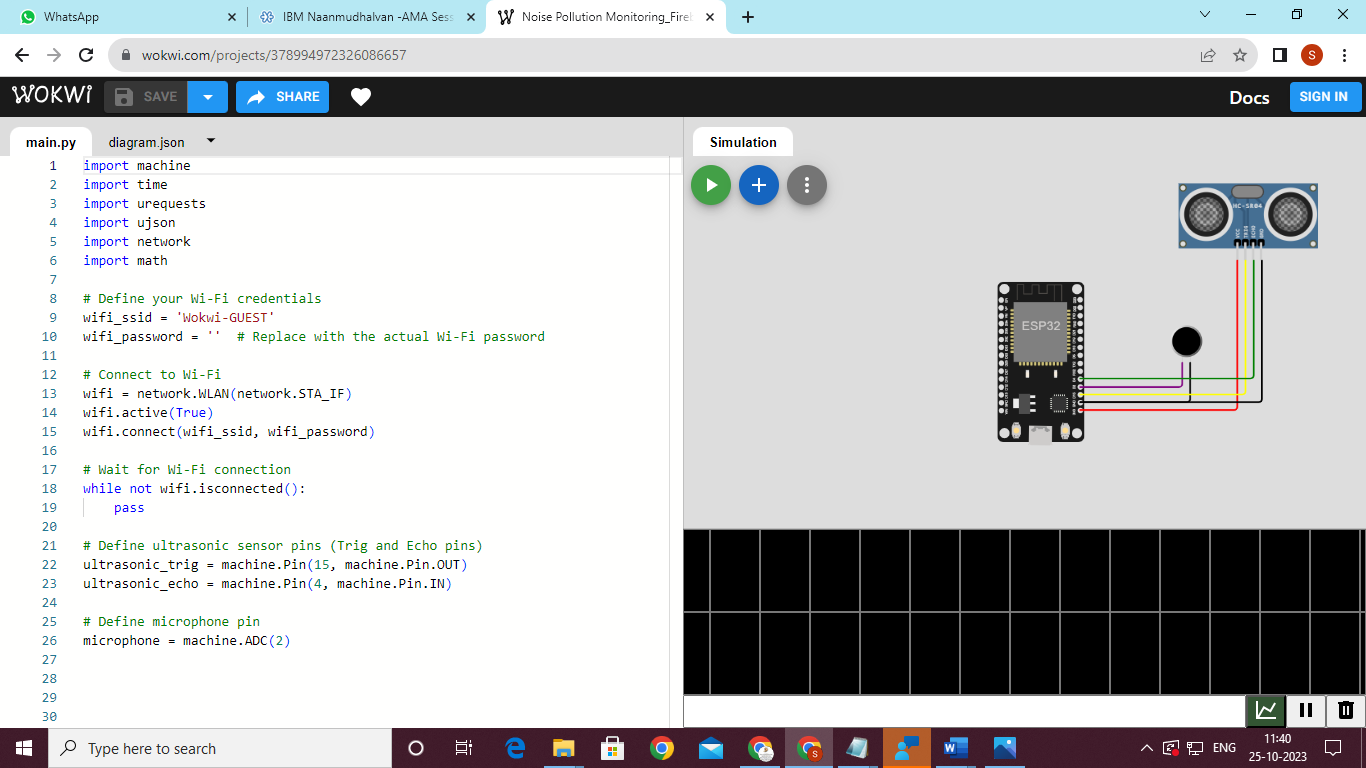
ultrasonic\_trig = machine.Pin(15, machine.Pin.OUT)

ultrasonic\_echo = machine.Pin(4, machine.Pin.IN)

# Define microphone pin

micl: rophone = machine.ADC(2)

**Model:**



## **Conclusion:**

This concludes your starter analysis! To go forward from here, click the blue "Fork Notebook" button at the top of this kernel. This will create a copy of the code and environment for you to edit. Delete, modify, and add code as you please. Happy Kaggling!