LAB EXAM END SEM-23

MDSC-102 Inferential Statistics(P)

Dataset: Air Quality Decode

Source: Unlocking City Air Quality: Decode AQI with Us!

Register number: 23913

Subject Code: MDSC-102(P)

Date: 17/10/23

INDEX

About the dataset:

Air pollution is a major global issue that affects the health and well-being of millions of people. The Air Quality Index (AQI) is a widely-used measure of air pollution that provides information on the quality of air in a city on a daily basis.

The AQI is divided into six levels of air quality, ranging from "good" to "hazardous".

1. **Good (0-50)** - This indicates minimal impact.
2. **Satisfactory (51-100)** - This category can cause minor breathing difficulties in susceptible individuals.
3. **Moderately polluted (101-200)** - This category can cause breathing difficulties in people with lung diseases, such as asthma, as well as discomfort for heart disease patients, children, and older adults.
4. **Poor (201-300)** - This category can cause breathing difficulties in people who are exposed to it for prolonged periods, as well as discomfort for individuals with heart disease.
5. **Very Poor (301-400)** - This category can cause respiratory illness in people who are exposed to it for prolonged periods.
6. **Severe (401-500)** - This category can cause respiratory issues in otherwise healthy people, and may result in very severe health problems for those with lung or heart disease.
7. **Very Severe (500 and above)** - This category is considered uninhabitable.

The dataset contains observations on the following variables:

City: the name of the city where the air quality is measured

Datetime: the date and time of the air quality measurement

PM2.5: concentration of particulate matter less than 2.5 micrometers in diameter (unit: micrograms per cubic meter, $\mu g/m^3$)

PM10: concentration of particulate matter less than 10 micrometers in diameter (unit: micrograms per cubic meter, $\mu g/m^3$)

NO: concentration of nitrogen monoxide (unit: parts per billion, ppb)

NO2: concentration of nitrogen dioxide (unit: parts per billion, ppb)

NOx: concentration of nitrogen oxides (unit: parts per billion, ppb)

NH3: concentration of ammonia (unit: parts per billion, ppb)

CO: concentration of carbon monoxide (unit: parts per million, ppm)

SO2: concentration of sulfur dioxide (unit: parts per billion, ppb)

O3: concentration of ozone (unit: parts per billion, ppb)

Benzene: concentration of benzene (unit: micrograms per cubic meter, $\mu g/m^3$)

Toluene: concentration of toluene (unit: micrograms per cubic meter, $\mu g/m^3$)

Xylene: concentration of xylene (unit: micrograms per cubic meter, $\mu g/m^3$)

AQI_Bucket: the Air Quality Index (AQI) bucket to which the observation belongs. The AQI bucket represents the level of air quality, with higher values indicating poorer air quality.

Introduction:

Before importing the dataset and starting preprocessing and plotting we will discuss some of the main tools which I used in this process.

```
import numpy as np
import pandas as pd
```

NumPy provides fundamental numerical operations and multidimensional array support, while Pandas offers data structures and data analysis tools to efficiently work with data. These libraries are crucial for data preprocessing, cleaning, and initial data exploration.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

These libraries enable the creation of visualizations that help to understand data patterns and convey insights effectively. Matplotlib is a versatile plotting library, while Seaborn builds on it, making it easier to create informative and aesthetically pleasing plots.

```
import scipy as sp
from scipy import stats
```

SciPy extends NumPy by offering a wide range of statistical functions, optimization tools, and additional scientific capabilities. It's particularly useful for advanced statistical analysis, hypothesis testing, and scientific research .

Importing and preprocessing:

```
#importing data using pandas
df = pd.read_csv("/content/drive/MyDrive/MDSC/1 semester/102/lap_test_102/train.csv")
```

1. After importing the data from Google drive using pandas. Now the csv file is in df variable which is now pandas dataframe.

```
df.info() # info about data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495512 entries, 0 to 495511
Data columns (total 15 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   City        495512 non-null  object
 1   Datetime    495512 non-null  object
 2   PM2.5       488851 non-null  float64
 3   PM10        484693 non-null  float64
 4   NO          469994 non-null  float64
 5   NO2         489898 non-null  float64
 6   NOx         485090 non-null  float64
 7   NH3         474977 non-null  float64
 8   CO          490495 non-null  float64
 9   SO2         483354 non-null  float64
 10  O3          471898 non-null  float64
 11  Benzene     486993 non-null  float64
 12  Toluene     484508 non-null  float64
 13  Xylene      483352 non-null  float64
 14  AQI_Bucket  495512 non-null  int64
dtypes: float64(12), int64(1), object(2)
memory usage: 56.7+ MB
```

Now, we will see the basic information like Dtypes and Non-Null values and Column names etc:

In this Dataset,

- **Total 15 columns are there.**
- **Total 495512 values (rows)**
- **DTypes:**
  - **Float64 - 12 columns**
  - **Int64 - 1 column**
  - **Object - 2 columns**

```
df.head(20) # displaying first 20 data
```

	City	Datetime	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI_Bucket
0	Jaipur	2019-05-04 01:00:00	60.8	91.5	41.6	37.3	66.7	27.0	1.0	10.5	26.9	2.0	7.6	3.1	5
1	Amritsar	2017-04-27 19:00:00	23.8	119.1	14.9	17.1	NaN	28.4	0.0	6.7	19.1	3.1	8.7	3.1	5
2	Amaravati	2018-11-12 10:00:00	93.8	182.0	5.5	63.9	NaN	1.8	1.3	23.1	41.7	0.1	0.6	3.1	5
3	Kochi	2020-06-19 09:00:00	16.0	26.9	17.4	0.9	0.0	23.6	1.6	27.4	34.8	3.1	0.0	3.1	5
4	Ahmedabad	2019-05-16 03:00:00	29.5	138.4	NaN	31.6	26.0	23.6	9.3	NaN	36.9	14.5	86.8	11.5	1
5	Lucknow	2020-06-04 05:00:00	91.7	119.1	10.5	9.2	14.0	13.0	1.2	5.5	31.9	NaN	1.7	3.1	3
6	Thiruvananthapuram	2019-05-02 09:00:00	39.2	95.5	0.9	3.5	2.6	2.8	1.2	1.8	46.6	3.1	8.7	3.1	1
7	Talcher	2019-06-08 08:00:00	41.4	89.4	37.0	2.3	39.3	8.2	0.8	23.5	2.4	0.0	8.7	3.1	1
8	Guwahati	2020-02-08 19:00:00	175.8	309.9	49.4	37.5	121.3	18.2	1.2	12.4	18.2	1.5	8.7	3.1	4
9	Lucknow	2020-03-01 17:00:00	48.6	119.1	3.6	17.8	12.4	19.9	1.2	4.8	66.0	17.0	16.6	3.1	5
10	Lucknow	2018-01-01 22:00:00	130.5	119.1	8.8	30.4	22.9	53.4	1.6	9.0	53.4	5.1	1.2	3.1	6
11	Ahmedabad	2015-10-30 05:00:00	44.5	119.1	1.0	8.1	9.0	23.6	1.0	6.6	39.8	0.0	0.8	0.0	0
12	Amritsar	2017-08-14 01:00:00	6.4	79.2	6.4	16.4	32.3	7.8	0.7	5.6	4.4	13.5	0.9	19.9	3
13	Jaipur	2018-03-15 12:00:00	40.2	188.5	7.6	23.7	29.9	19.0	0.7	6.1	43.8	0.3	0.8	3.1	2
14	Delhi	2017-01-04 06:00:00	362.0	438.1	312.1	96.9	102.6	75.3	2.5	30.7	51.6	9.2	20.7	3.1	4

```
df.isnull().sum() # count null values
```

```
City             0
Datetime         0
PM2.5         6661
PM10         10819
NO           25518
NO2           5614
NOx          10422
NH3          20535
CO            5017
SO2          12158
O3           23614
Benzene       8519
Toluene      11004
Xylene       12160
AQI_Bucket       0
dtype: int64
```

2. **NULL VALUES:** There are Null values in the feature variables like PM2.5, PM10, NO etc:

   So, to fill the null values I used a technique, which is to count the cities in the dataset and the null values present in the particular variable . like this,

```
#in that cities count and number of NUll values are there in NO2
for col in city:
  null_count = df[df['City'] == col]['NO2'].isnull().sum()
  print(col,'total count',(30-len(col)-10)*' ',city[col],'Missing value of NO2',null_count)
```

```
Jaipur total count               18600 Missing value of NO2 192
Amritsar total count             20503 Missing value of NO2 215
Amaravati total count            15887 Missing value of NO2 177
Kochi total count                2699 Missing value of NO2 31
Ahmedabad total count            33662 Missing value of NO2 391
Lucknow total count              33699 Missing value of NO2 414
```

3. After finding those values now we can infer from the values that there are more number values as rows and less number of null values in the particular column. So, for variables related to air pollutants (e.g., PM2.5, PM10, NO, NO2, NOx, CO, SO2, O3, Benzene, Toluene, Xylene), it's more appropriate to fill missing values with the **mean**, as these are continuous numerical measurements. Filling with the mean will preserve the overall statistical characteristics of the data.

   This is achieved by **groupby operation with respective cities**.

```
# Fill missing values with mode for each column within their respective city
df = df.groupby('City',group_keys=True).apply(fill_with_mean).reset_index(drop=True)
```

   By doing this we are filling the null values in the dataset.

4. **DateTIme** column is separated to make more inference.

   By Changing the datatype of the column from object to datetime and separating the Hour, date, month and year.

```
df['Datetime'] = pd.to_datetime(df['Datetime'])

df['Year'] = df['Datetime'].dt.year
df['Month'] = df['Datetime'].dt.month
df['Day'] = df['Datetime'].dt.day
df['Hour'] = df['Datetime'].dt.hour
```

Data Visualisation:

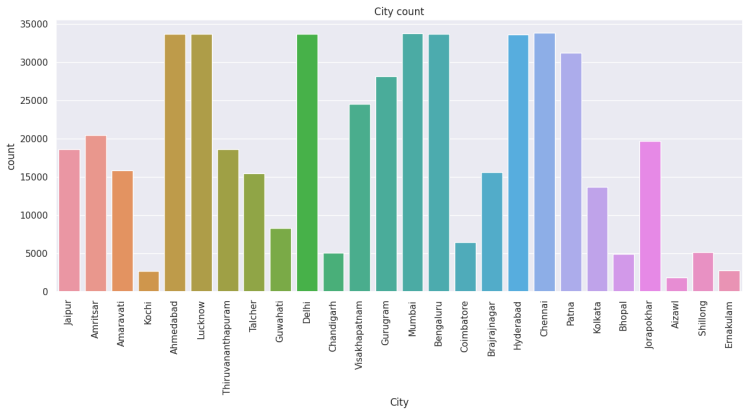
1. Since our dataset is very big (containing 50000 rows plus). so , I dropped rows with one condition. That is, each city records I'm keeping 10 entries for one city and dropping others so that I will have 10 or 11 entries of all cities' records. This will be useful to do hypothesis testing and data understanding.

```
entities = 10
# Group the data by 'City'
df = df.groupby('City', group_keys=False).apply(lambda x: x.sample(n=entities, random_state=1))
df.reset_index(drop=True, inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260 entries, 0 to 259
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   City         260 non-null    object
 1   Datetime     260 non-null    object
 2   PM2.5        260 non-null    float64
 3   PM10         260 non-null    float64
 4   NO           260 non-null    float64
 5   NO2          260 non-null    float64
 6   NOx          260 non-null    float64
 7   NH3          260 non-null    float64
 8   CO           260 non-null    float64
 9   SO2          260 non-null    float64
 10  O3           260 non-null    float64
 11  Benzene      260 non-null    float64
 12  Toluene      260 non-null    float64
 13  Xylene       260 non-null    float64
 14  AQI_Bucket   260 non-null    int64
dtypes: float64(12), int64(1), object(2)
memory usage: 30.6+ KB
```

2. Plotting the count of cities, which is presented in the dataset.

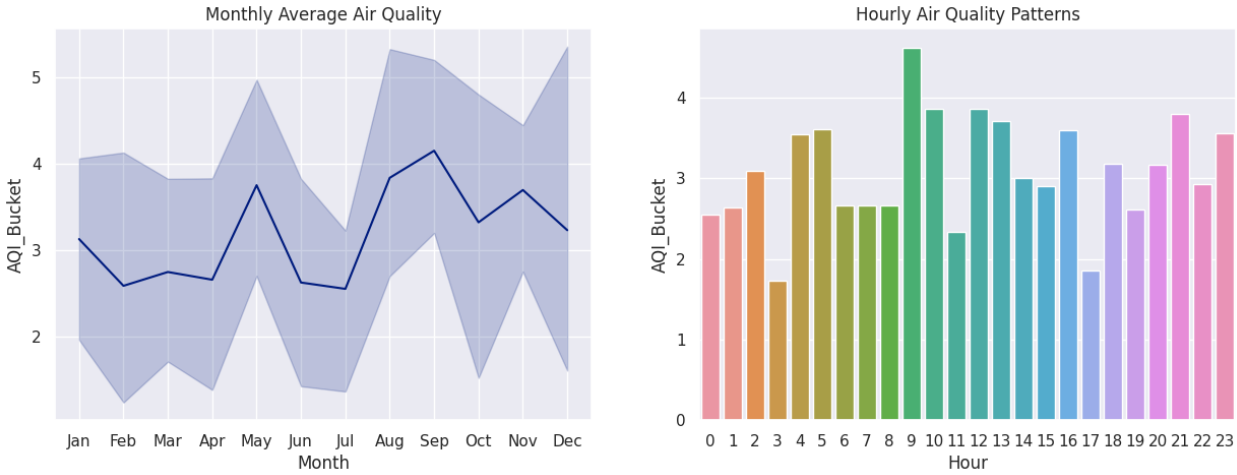


The above plotting shows that in the dataset Ahmedabad, Lucknow, Mumbai, Bengaluru and Chennai are having the highest entries of rows among all the cities.

3. Plotting in line plot and bar plot of monthly average and hourly average of the air quality.
Monthly average is calculated by grouping year and month and AQI_Bucket and by taking the mean of it.
Hourly average is calculated by grouping hour and AQI_Bucket and by taking the mean of it.

```
monthly_avg = df.groupby(['Year', 'Month'])['AQI_Bucket'].mean().reset_index()
hourly_avg = df.groupby('Hour')['AQI_Bucket'].mean().reset_index()

plt.figure(figsize=(15, 6))

plt.subplot(1,2,1)
sns.lineplot(x='Month', y='AQI_Bucket', data=monthly_avg)
```

```
plt.title("Monthly Average Air Quality")
plt.xticks(range(1, 13), [calendar.month_abbr[i] for i in range(1, 13)])

plt.subplot(1,2,2)
sns.barplot(x="Hour", y="AQI_Bucket", data=hourly_avg)
plt.title("Hourly Air Quality Patterns")
```



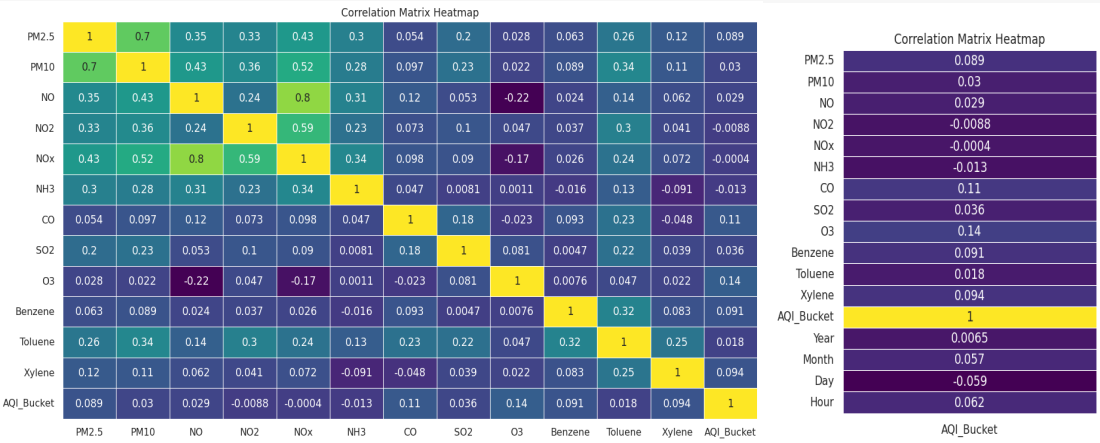
The above two plots show the air quality over the month and in one day.

The 1st plot shows that on average the air quality is bad after the august month in all cities . Highest may be in September and lowest in July. In the month of May the air quality is bad .

The 2nd plot shows that in a day during the peak hours for example 9AM is the time which records more bad air quality whereas at the time 3AM air quality is very good.

4. A correlation plot, by using corr() function, visualizes the relationships between variables in our dataset by quantifying the strength and direction of their linear associations.
Heatmap is used to provide an intuitive way to identify both positive and negative correlations.

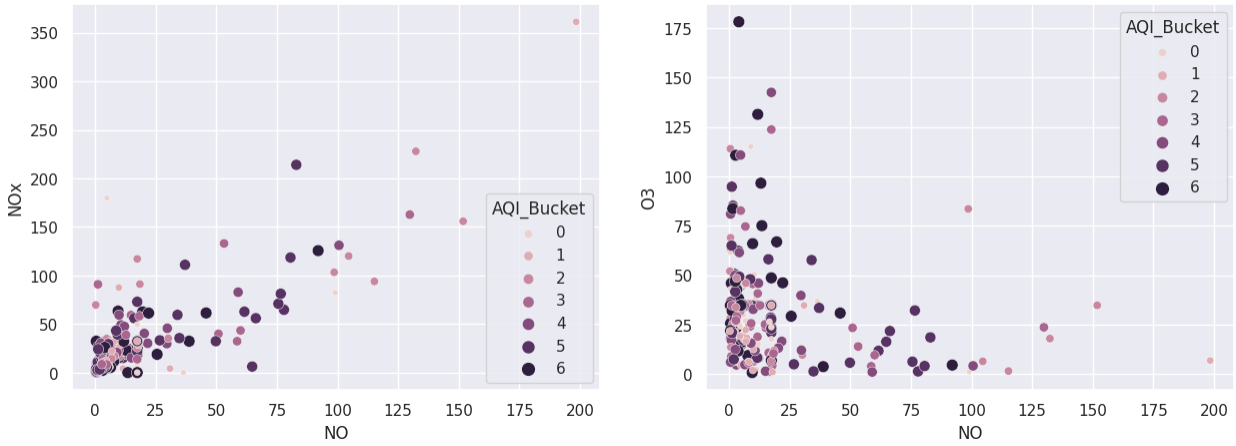
```
correlation_matrix = df.corr()
plt.figure(figsize=(15, 8))

sns.heatmap(correlation_matrix, annot=True, cmap="viridis", linewidths=0.5, cbar=False)
plt.title("Correlation Matrix Heatmap")
```



From the above heatmap we infer that

- **The features O3 and NO are highly Negatively correlated and the features O3 and NOX are also highly negatively correlated, by -0.22 and -0.17 respectively.**
- **The features NOX and NO are highly positively correlated by 0.8.**
- **The features like O3 and NO3 have no correlation because its relation is almost 0.**
- **The right side heat map shows the relation between the target variable (AQI_Bucket) and the features.**

5. Scatter plot is useful to see the relation between variables. I used sns to scatter plots and it is helpful in visualizing and understanding the relation between the two data features in one plot with some HUE.



The above plot shows us the relationship between the Feature NO with NOx and O3 with AQI_bucket (quality level). We can infer that

- **In the left side of the picture we can see the x axis as NO and y axis as NOx. we infer that Most of the time of recording Air quality, the content level of NO and NOx is very low. Wherever the NOx is low, NO is also low.**

  But both features are contributing to increase the AQI_Bucket level because from the plot we can see that when the NOx and NO content are high ie: above 100, the AQI_Bucket is very good.
- **In the right side of the picture we can see the x axis as NO and y axis as O3. we infer that Most of the time of recording Air quality, the content level of NO is very low and O3 is High. Wherever the NOx is low, O3 is High.**

  These features are negatively highly correlated because when NO is high and O3 is Low it is contributing to increase the Air quality level. When NO is low and O3 is high, it is observed to negatively contribute to the air quality.

Skewness and Normalization.

In this section, we will use the **BOX PLOT** to find the outliers and skewness and we use HISTPLOTto confirm the skewness and will measure the skewness value. And with the help of **PROBABILITY PLOT**, the plot is like a visual reality check for data and then we will normalize the data using boxcox and again will check the resulting plot and skewness of the features.

FIrst step will see the skewness of the dataset and plot the features to infer skewness from that

1. Skewness of the dataset before normalization:

```
df.skew()
```

```
<ipython-input-64-9e0b1e29!
  df.skew()
PM2.5           3.736473
PM10            1.776554
NO              3.420528
NO2             5.782236
NOx             3.974168
NH3             3.002169
CO             12.563817
SO2             5.011781
O3              2.071430
Benzene        14.198384
Toluene         4.039373
Xylene          3.087387
AQI_Bucket     -0.132105
Year           -0.825943
Month           0.141175
Day             0.087023
Hour           -0.130335
dtype: float64
```

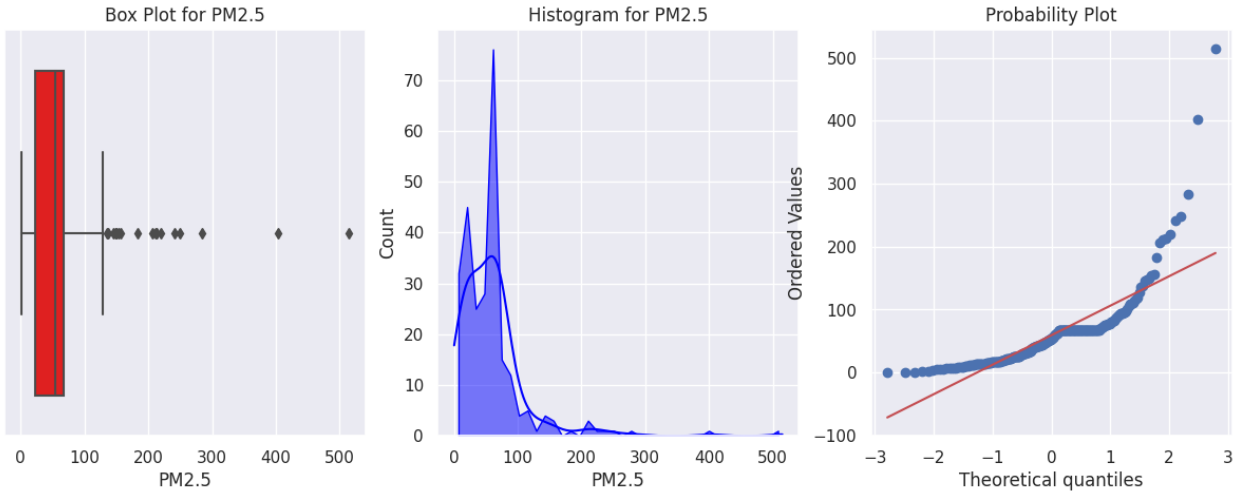
From the code we can infer that all the features are positively skewed. In a positively skewed distribution, the tail on the right side (the upper tail) is longer or fatter than the left side (the lower tail). The majority of the data points are concentrated on the left side of the distribution, and the tail extends to the right. This indicates that the data is right-tailed, and the mean is typically greater than the median.

2. Plotting before normalization:

```
def Plotting(features):

    plt.figure(figsize=(15,5))

    #BOX PLOT
    plt.subplot(1,3,1)
    plt.xlabel('X Label')
    sns.boxplot(data= df, x= features, color='red').set(title="Box Plot for "+features)

    #HISTPLOT
    plt.subplot(1,3,2)
    sns.histplot(df[features], kde=True, color='blue', element="poly").set(title="Histogram for " + features)

    #PROBABILITY PLOT
    plt.subplot(1,3,3)
    plt.title('Probability Plot for' + features)
    sp.stats.probplot(df[features], plot=plt)
```

Using the function plotting , we can plot all the features hisplot and box plot and probability plot for both before normalization and after normalization.

PM2.5



BOXPLOT -

Both whiskers are the same size. So we can't make any decision out of it.

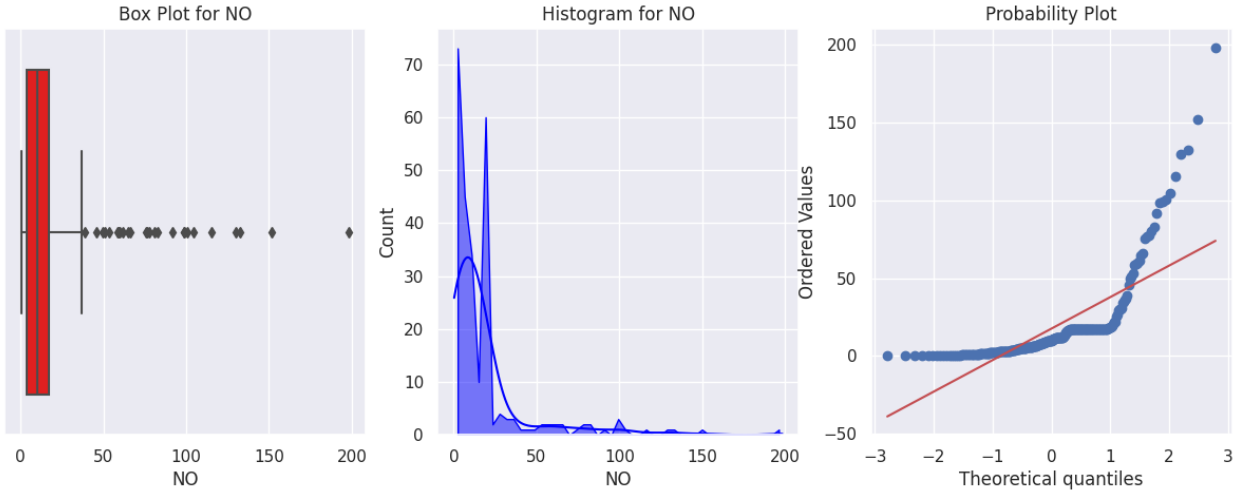
Most of the value lies between 1st quartile and 2nd quartile.

Many outliers are also there on the right side of the plot.

HISPLOT -

By seeing histplot we can infer that the plot is right skewed.

NO



BOXPLOT -

Left is comparatively lesser than the right one so, we can guess that it may be right skewed.

Most of the value lies between 2nd quartile and 3rd quartile.

Many outliers are also there on the right side of the plot.

HISPLOT -

By seeing histplot we can confirm that the plot is right skewed.

3. Normalization using Boxcox

```
lambdaList = {}
for column in features:
    # Perform the Box-Cox transformation
    transformed_data, lambda_val = stats.boxcox(df[column] + 1)  # Adding 1 to avoid
zero values
    lambdaList[column] = lambda_val
    df[column] = transformed_data
```

The Box-Cox transformation is a mathematical technique used to stabilize variance and make data more closely resemble a normal distribution by applying a power transformation.
By using the Box-Cox transformation we try to normalize the data and try to plot the graph.
The lambda Values:

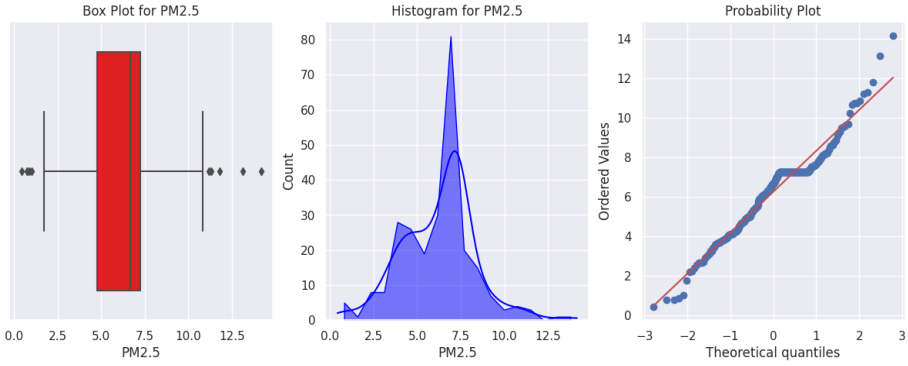
```
{'PM2.5': 0.228850044621929,
 'PM10': 0.3784306192807244,
 'NO': -0.06239116793656803,
 'NO2': 0.17334152228094482,
 'NOx': 0.13224403576793106,
 'NH3': 0.24013180252232177,
 'CO': -0.6722351409028531,
 'SO2': -0.05947462290516178,
 'O3': 0.3443353348262985,
 'Benzene': -0.21638150710197543,
 'Toluene': 0.10932050143949244,
 'Xylene': 0.3587035765717658,
 'AQI_Bucket': 0.8236808129506861}
```

4. Skewness after Normalization:

```
PM2.5         0.050411
PM10          0.021827
NO            0.007757
NO2           0.019850
NOx           0.015702
NH3           0.031282
CO            0.011979
SO2          -0.011742
O3            0.014891
Benzene       0.014350
Toluene      -0.030328
Xylene        0.142259
AQI_Bucket   -0.248229
Year         -0.825943
Month         0.141175
Day           0.087023
Hour         -0.130335
dtype: float64
```

5. Plotting After normalization:

PM2.5



BOXPLOT -

After normalizing the data we can see that both whiskers are the same size and the mean lies at the center.

HISPLOT -

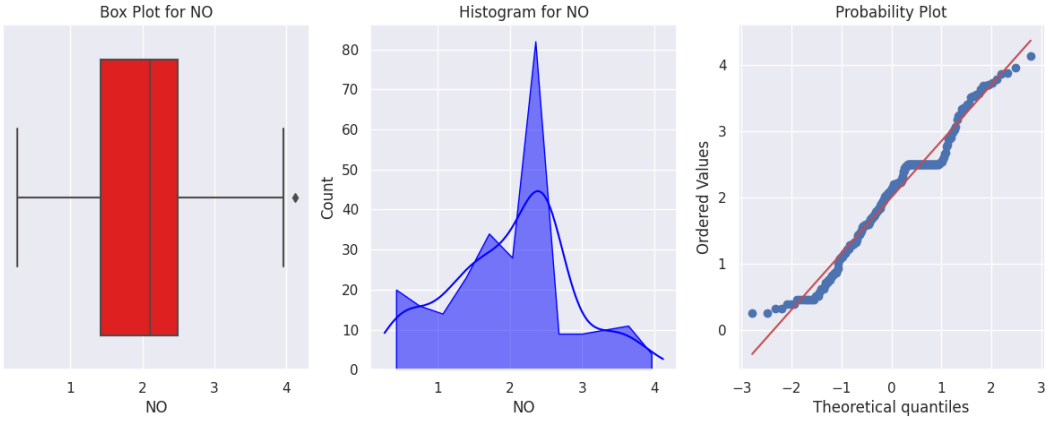
By seeing histplot we can confirm the data is almost normalized.

PP -

All data points lie in almost linear fashion .

.

NO



BOXPLOT -

After normalizing the data we can see that both whiskers are the same size and the mean lies at the center.

HISPLOT -

By seeing histplot we can confirm the data is almost normalized.

PP -

All data points lie in almost linear fashion .

Hypothesis Testing:

Hypothesis testing is a common statistical technique used to make inferences about population parameters, such as the population mean (μ) and variance (σ^2).

- **Hypothesis Testing for μ with Known Population Variance (σ^2):**

  Null Hypothesis (H0): μ = μ0
  Alternative Hypothesis (H1): μ ≠ μ0

  For this test, the Z Test will be used.

- **Hypothesis Testing for μ with Unknown Population Variance (σ^2):**

  Null Hypothesis (H0): μ = μ0
  Alternative Hypothesis (H1): μ ≠ μ0

  For this test, the T test will be used. If the population is more than 25, then the Z test will be used.

- **Hypothesis Testing for σ^2 with Known Population Mean (μ):**

  Null Hypothesis (H0): σ^2 = σ0^2
  Alternative Hypothesis (H1): σ^2 ≠ σ0^2

  For this test, the Chi-Square (χ^2) test will be used.

1. Now we created a common universal function accepting the data when **σ^2 is known and σ^2 is unknown.**

```
def hypothesis_test(mu0, alpha, data, test_type="t"):
    if test_type == "t" and len(data) >= 25:
        print('performing z test because data is suffiecently large')
        z_stat = (data.mean() - mu0) / (data.std() / (len(data)**0.5))
        p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))
    elif test_type == "z":
        print('performing z test')
        z_stat = (data.mean() - mu0) / (data.std() / (len(data)**0.5))
        p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))
    elif test_type == "t":
        print('performing t test')
        t_stat, p_value = stats.ttest_1samp(data, mu0)

    if p_value < alpha:
        print("Reject the null hypothesis (H0)")
    else:
        print("Do not reject the null hypothesis (H0)")

alpha = 0.01
mu0 = 6

# Perform a t-test ( n = 260, so it will perform z test only)
hypothesis_test(mu0, alpha, df['PM2.5'], test_type="t")

# Perform a z-test (assuming known population variance)
hypothesis_test(mu0, alpha, df['PM2.5'], test_type="z")
```

In this program, in that function we are accepting the mu0, alpha value, data and test type weather t or z. Inside the function,

If it is a t test then we will check the population size.

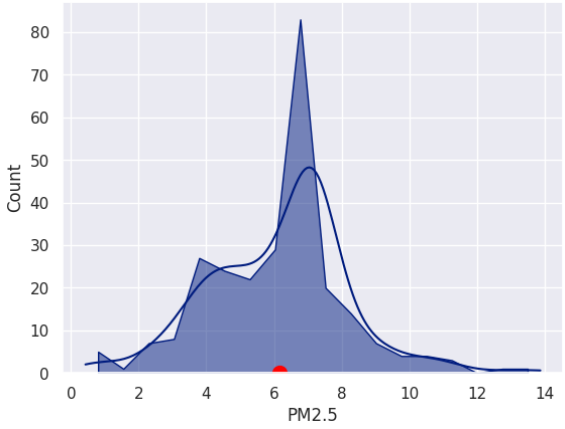
If it is more than 25, then will perform that z test with std(). Otherwise T test.

If it is a Z test then, will perform z test. Both tests are performed using **stats.norm.cdf and stats.ttest.1samp.**

Example:

- **Hypothesis Testing for μ with Known Population Variance (σ^2) for PM2.5 feature:**

  Null Hypothesis (H0): μ = 6
  Alternative Hypothesis (H1): μ ≠ 6
  TEST - Z



ORIGINAL MEAN:

```
df['PM2.5'].mean()
```

```
6.159823100448004
```

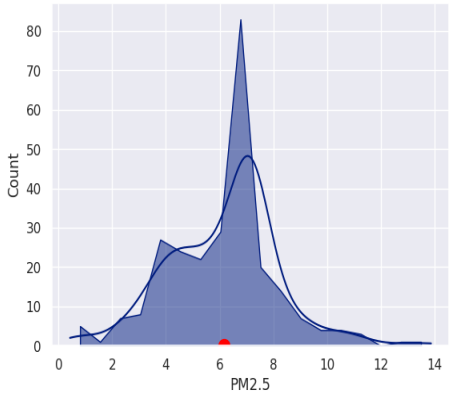
RESULT:

```
performing z test
Do not reject the null hypothesis (H0)
```

- **Hypothesis Testing for μ with unKnown Population Variance (σ^2) for PM2.5 feature:**

  Null Hypothesis (H0): μ = 6
  Alternative Hypothesis (H1): μ ≠ 6

  TEST - Z



ORIGINAL MEAN:

```
df['PM2.5'].mean()
```

```
6.159823100448004
```

RESULT:

```
performing z test because data is suffiecently large
Do not reject the null hypothesis (H0)
```

2. Now we created a function accepting the data when **μ is known.**

```
def variance_hypothesis_test(data, sigma0_squared, alpha):
    n = len(data)
    sample_variance = ((data - data.mean())**2).sum() / (n - 1)

    # Hypothesis Testing for σ^2 with Known Population Mean
    chi_squared_stat = (n - 1) * sample_variance / sigma0_squared
    p_value = 1 - stats.chi2.cdf(chi_squared_stat, df=n - 1)


    # Compare the p-value to the significance level alpha
    if p_value < alpha:
        print("Reject the null hypothesis")
    else:
        print("Do not reject the null hypothesis")

alpha = 0.05
sigma0_squared = 0.8

variance_hypothesis_test(df['NO'], sigma0_squared, alpha)
```

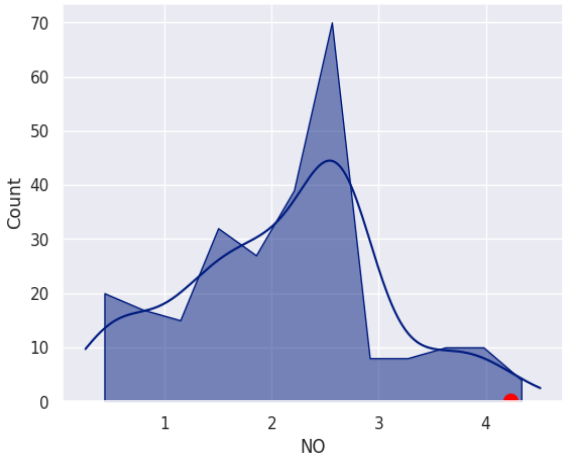
In this program, in that function we are accepting the sigma0, alpha value, data.
Inside the function,
We are calculating sample variance and chi-squared_stat and rejecting the H0 if p_value is less than alpha value.
Test is performed using **stats.chi2.cdf.**

Example:

- **Hypothesis Testing for σ^2 with Known Population Mean (μ) for NO feature:**

Null Hypothesis (H0): σ^2 = 0.6
Alternative Hypothesis (H1): σ^2 ≠ 0.6

TEST - Chi-Square (χ^2)



ORIGINAL Variance:

```
df['NO'].var()
```

```
0.8726753523731399
```

RESULT:

```
Reject the null hypothesis
```