PerformPreprocessingTask

September 11, 2022

1 Data Preprocessing

[1]:
```
# Importing the Required Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

1.1 Loading the CSV file into Pandas Dataframe

[2]:
```
df = pd.read_csv("data.csv",encoding='ISO-8859-1')
```

```
/tmp/ipykernel_6616/901186919.py:1: DtypeWarning: Columns (0) have mixed types.
Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("data.csv",encoding='ISO-8859-1')
```

[3]:
```
df.head()
```

[3]:

	stn_code	sampling_date	state	location	agency	\
0	150.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	
1	151.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	
2	152.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	
3	150.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	
4	151.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	

	type	so2	no2	rspm	spm	\
0	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	
1	Industrial Area	3.1	7.0	NaN	NaN	
2	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	
3	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	
4	Industrial Area	4.7	7.5	NaN	NaN	

	location_monitoring_station	pm2_5	date
0	NaN	NaN	1990-02-01
1	NaN	NaN	1990-02-01
2	NaN	NaN	1990-02-01
3	NaN	NaN	1990-03-01

```
4                          NaN    NaN  1990-03-01
```

1.2 1. Summarized details of the data.

1.2.1 Data Type Info

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435742 entries, 0 to 435741
Data columns (total 13 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   stn_code                     291665 non-null  object 
 1   sampling_date                435739 non-null  object 
 2   state                        435742 non-null  object 
 3   location                     435739 non-null  object 
 4   agency                       286261 non-null  object 
 5   type                         430349 non-null  object 
 6   so2                          401096 non-null  float64
 7   no2                          419509 non-null  float64
 8   rspm                         395520 non-null  float64
 9   spm                          198355 non-null  float64
 10  location_monitoring_station  408251 non-null  object 
 11  pm2_5                        9314 non-null    float64
 12  date                         435735 non-null  object 
dtypes: float64(5), object(8)
memory usage: 43.2+ MB
```

1.2.2 Statistical Summary

[5]:
```
df.describe()
```

[5]:

	so2	no2	rspm	spm	pm2_5
count	401096.000000	419509.000000	395520.000000	198355.000000	9314.000000
mean	10.829414	25.809623	108.832784	220.783480	40.791467
std	11.177187	18.503086	74.872430	151.395457	30.832525
min	0.000000	0.000000	0.000000	0.000000	3.000000
25%	5.000000	14.000000	56.000000	111.000000	24.000000
50%	8.000000	22.000000	90.000000	187.000000	32.000000
75%	13.700000	32.200000	142.000000	296.000000	46.000000
max	909.000000	876.000000	6307.033333	3380.000000	504.000000

1.3 2. Count the number of data

[6]:
```
df.size
```

[6]:
```
5664646
```

1.4 3. How Many rows and Columns are there in the dataset

[7]:
```
df.shape
```

[7]:
```
(435742, 13)
```

There are 435742 rows and 13 columns

1.5 4. How many null data are in each column

[8]:
```
df_null_count = pd.DataFrame(df.isna().sum()).reset_index()
df_null_count = df_null_count.rename(columns = {'index':'Column Name',0:
 ↪'NullCount'})
df_null_count['percentage'] = (df_null_count['NullCount']/df.shape[0])*100
df_null_count.sort_values(by=['percentage'],ascending=False)
```

[8]:

	Column Name	NullCount	percentage
11	pm2_5	426428	97.862497
9	spm	237387	54.478797
4	agency	149481	34.304933
0	stn_code	144077	33.064749
8	rspm	40222	9.230692
6	so2	34646	7.951035
10	location_monitoring_station	27491	6.309009
7	no2	16233	3.725370
5	type	5393	1.237659
12	date	7	0.001606
1	sampling_date	3	0.000688
3	location	3	0.000688
2	state	0	0.000000

From the datafame above we can see that column `pm2_5` has maximum percentage of null values

1.6 5. Drop out the following columns stn_code, agency, sampling_date, location_monitoring_agency

[9]:
```
df.
 ↪drop(columns=['stn_code','agency','sampling_date','location_monitoring_station'],␣
 ↪inplace=True)
```

[10]:
```
df.head()
```

[10]:

	state	location	type	so2	no2	\
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	

```
2  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  6.2  28.5
3  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  6.3  14.7
4  Andhra Pradesh  Hyderabad                      Industrial Area  4.7   7.5

   rspm  spm  pm2_5        date
0   NaN  NaN    NaN  1990-02-01
1   NaN  NaN    NaN  1990-02-01
2   NaN  NaN    NaN  1990-02-01
3   NaN  NaN    NaN  1990-03-01
4   NaN  NaN    NaN  1990-03-01
```

[11]:
```
df[df['date'].isna()].shape
```

[11]:
```
(7, 9)
```

[12]:
```
df.shape
```

[12]:
```
(435742, 9)
```

1.7 6. Drop out the row where the date is not available.

[13]:
```
df.dropna(subset=['date'],inplace=True)
```

[14]:
```
df.shape
```

[14]:
```
(435735, 9)
```

[15]:
```
df['type'].value_counts()
```

[15]:
```
Residential, Rural and other Areas    179013
Industrial Area                        96089
Residential and others                 86791
Industrial Areas                       51747
Sensitive Area                          8979
Sensitive Areas                         5536
RIRUO                                   1304
Sensitive                                495
Industrial                               233
Residential                              158
Name: type, dtype: int64
```

1.8 7. Making Values Uniform

Notice that the 'type' column has values such as 'Industrial Area' and 'Industrial Areas'. So, let's remove such type of stuff as changing the types to uniform format As:

1. "Residential": "R",
2. "Residential and others": "RO",

3. "Residential, Rural and other Areas": "RRO",
4. "Industrial Area": "I",
5. "Industrial Areas": "I",
6. "Industrial": "I",
7. "Sensitive Area": "S",
8. "Sensitive Areas": "S",
9. "Sensitive": "S",
10. np.nan: "RRO"

[16]:
```
df['type']=df['type'].replace({
    'Industrial Area':'I',
    'Industrial Areas':'I',
    'Industrial':'I',
    'Sensitive Area':'S',
    'Sensitive Areas':'S',
    'Sensitive':'S',
    'Residential':'R',
    'Residential and others':'RO',
    'Residential, Rural and other Areas':'PRO',
    np.nan:'PRO'
})
```

[17]:
```
df['type'].value_counts()
```

[17]:
```
PRO      184403
I        148069
RO        86791
S         15010
RIRUO      1304
R           158
Name: type, dtype: int64
```

[18]:
```
df.isna().sum()
```

[18]:
```
state           0
location        0
type            0
so2         34643
no2         16230
rspm        40219
spm        237380
pm2_5      426421
date            0
dtype: int64
```

1.9 8. Create additional column year

[19]:
```
df['year'] = pd.to_datetime(df['date']).dt.year
```

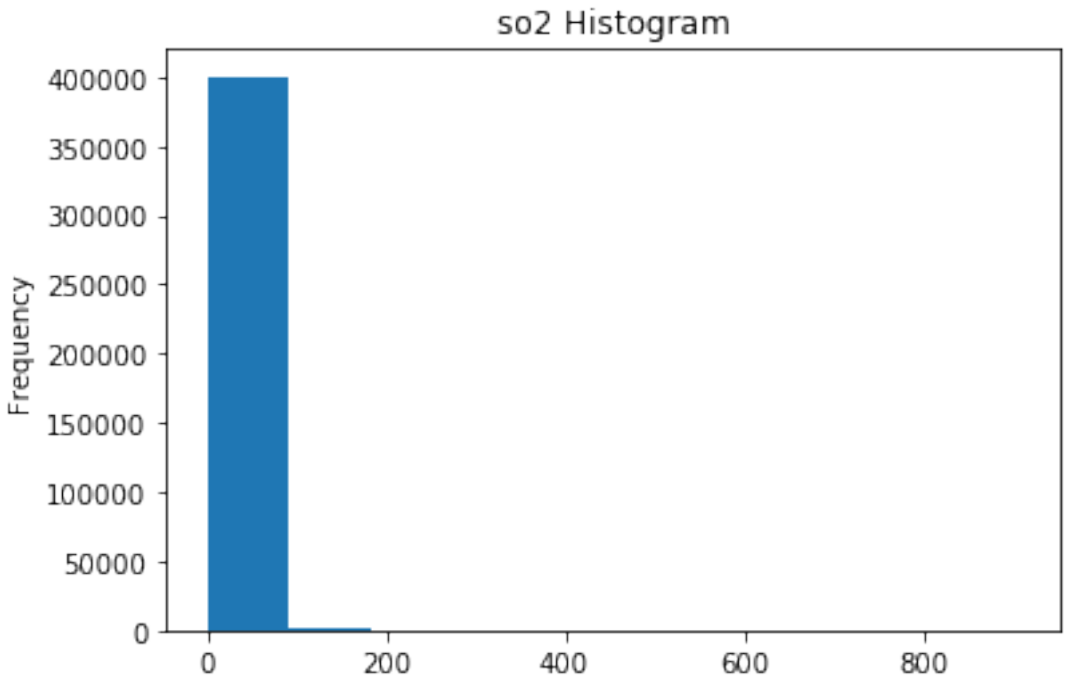
[20]:
```
df.head()
```

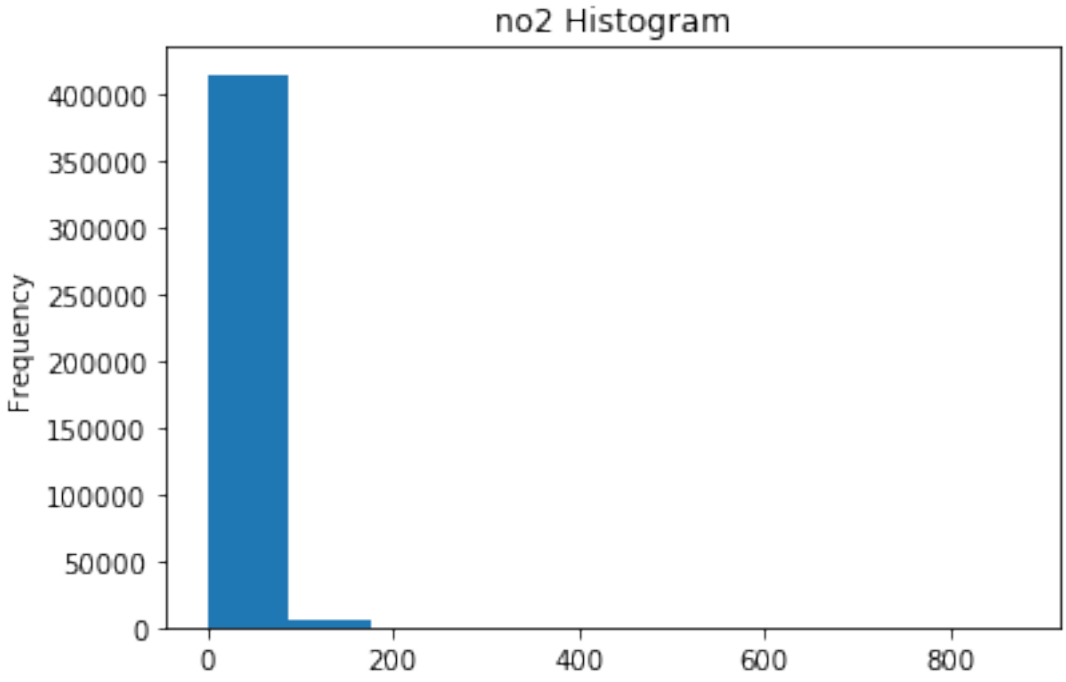
[20]:
```
            state   location type  so2   no2  rspm  spm  pm2_5        date  \
0  Andhra Pradesh  Hyderabad  PRO  4.8  17.4   NaN  NaN    NaN  1990-02-01
1  Andhra Pradesh  Hyderabad    I  3.1   7.0   NaN  NaN    NaN  1990-02-01
2  Andhra Pradesh  Hyderabad  PRO  6.2  28.5   NaN  NaN    NaN  1990-02-01
3  Andhra Pradesh  Hyderabad  PRO  6.3  14.7   NaN  NaN    NaN  1990-03-01
4  Andhra Pradesh  Hyderabad    I  4.7   7.5   NaN  NaN    NaN  1990-03-01

   year
0  1990
1  1990
2  1990
3  1990
4  1990
```

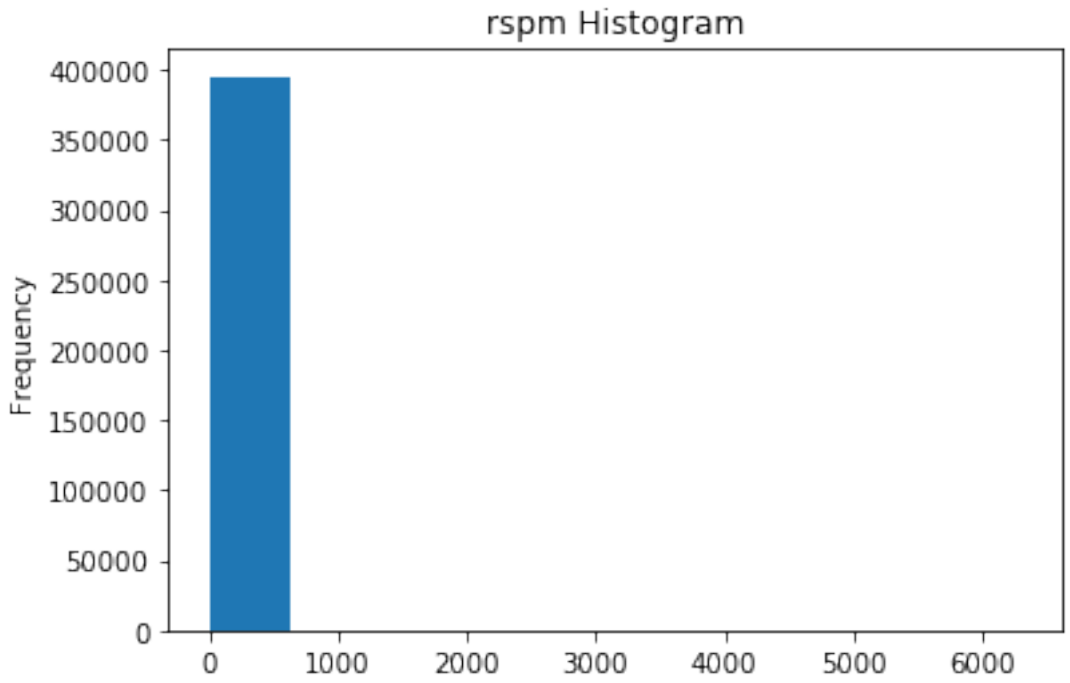
1.10 9. Remove null values from SO2, NO2, rspm, spm, pm2_5

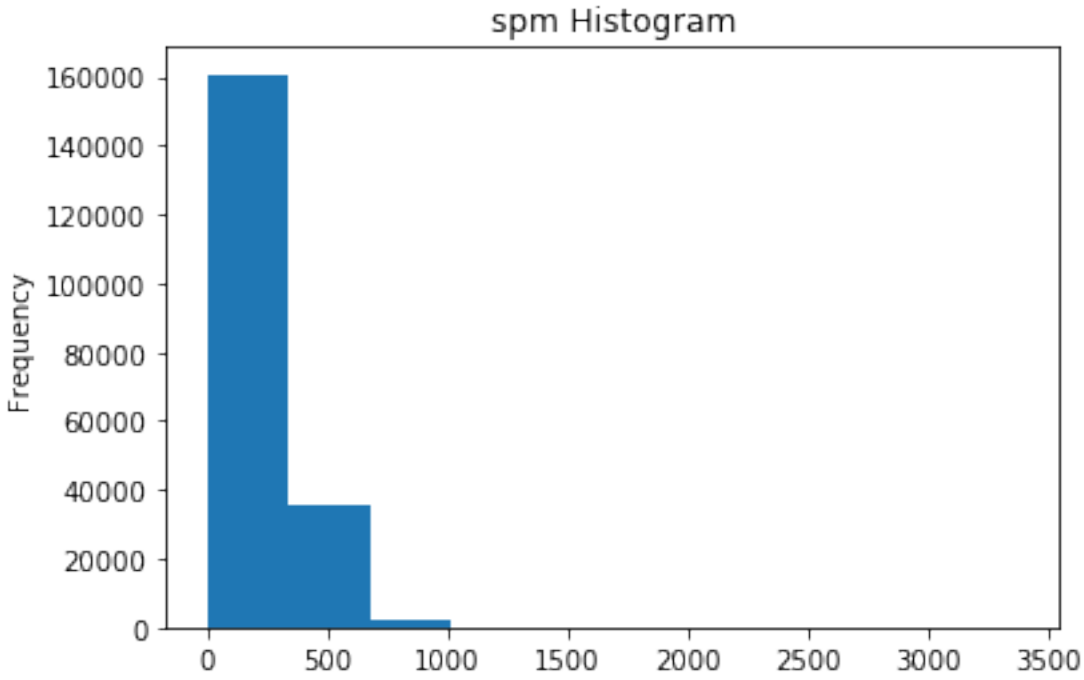
1.10.1 Plotting Histogram for Each Column

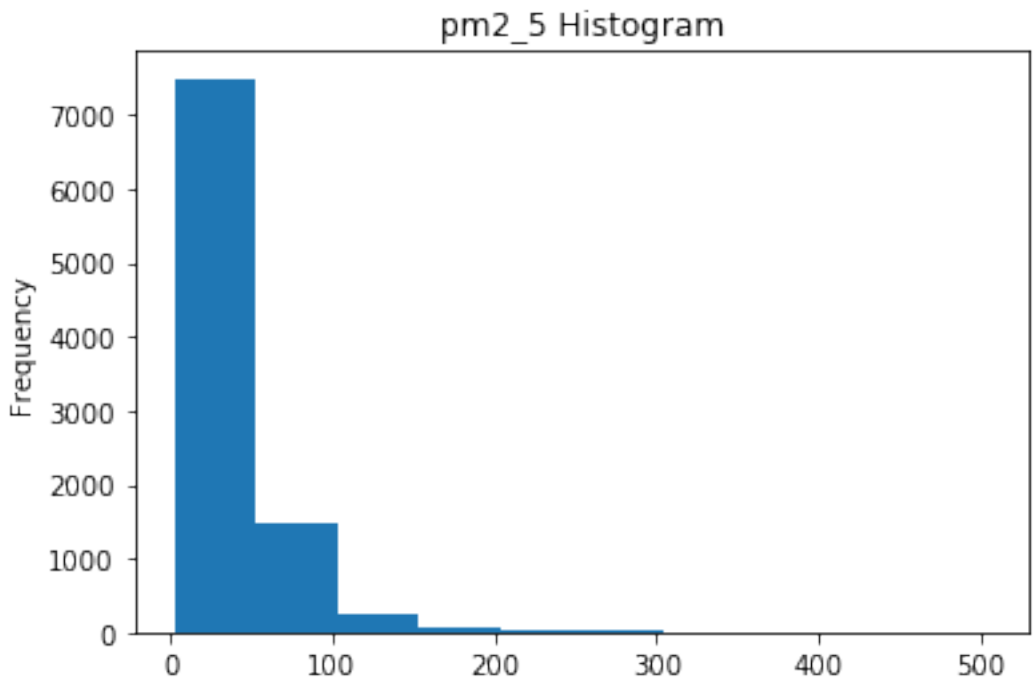
[21]:
```
columns = ['so2','no2','rspm','spm','pm2_5']
for column in columns:
    df[column].plot(kind='hist')
    plt.title(f'{column} Histogram')
    plt.show()
```


so2 Histogram
Frequency
400000
350000
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800

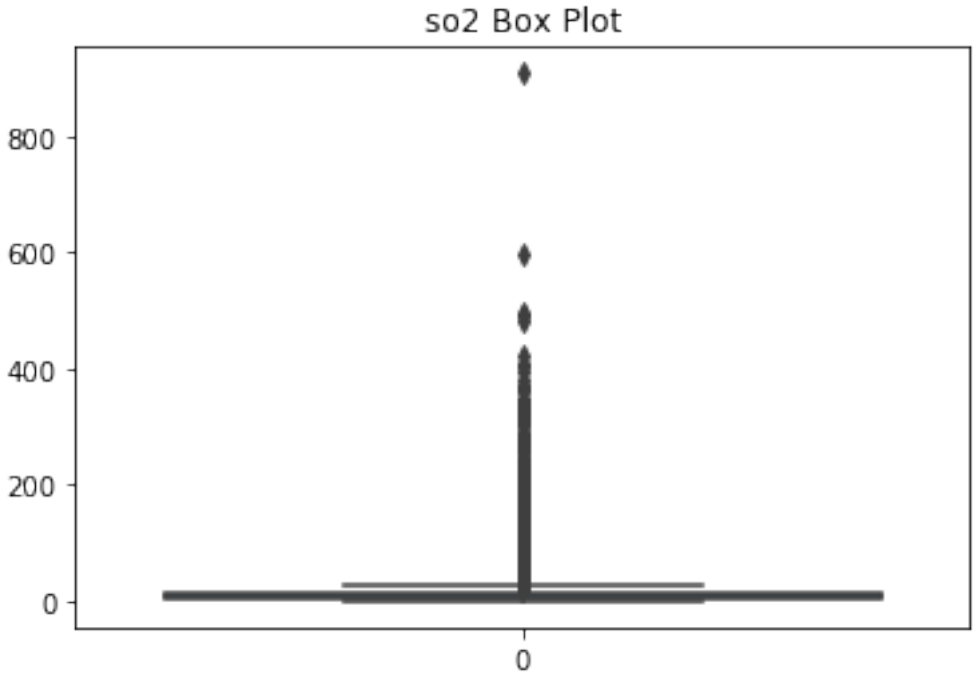

no2 Histogram
Frequency
400000
350000
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800

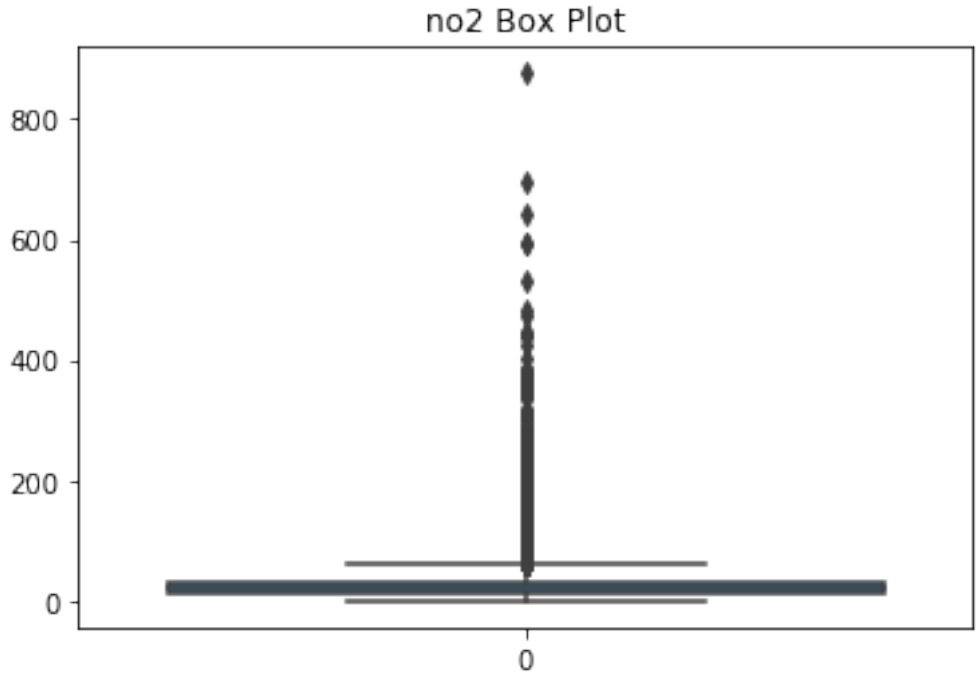

rspm Histogram
Frequency
400000
350000
300000
250000
200000
150000
100000
50000
0
0
1000
2000
3000
4000
5000
6000

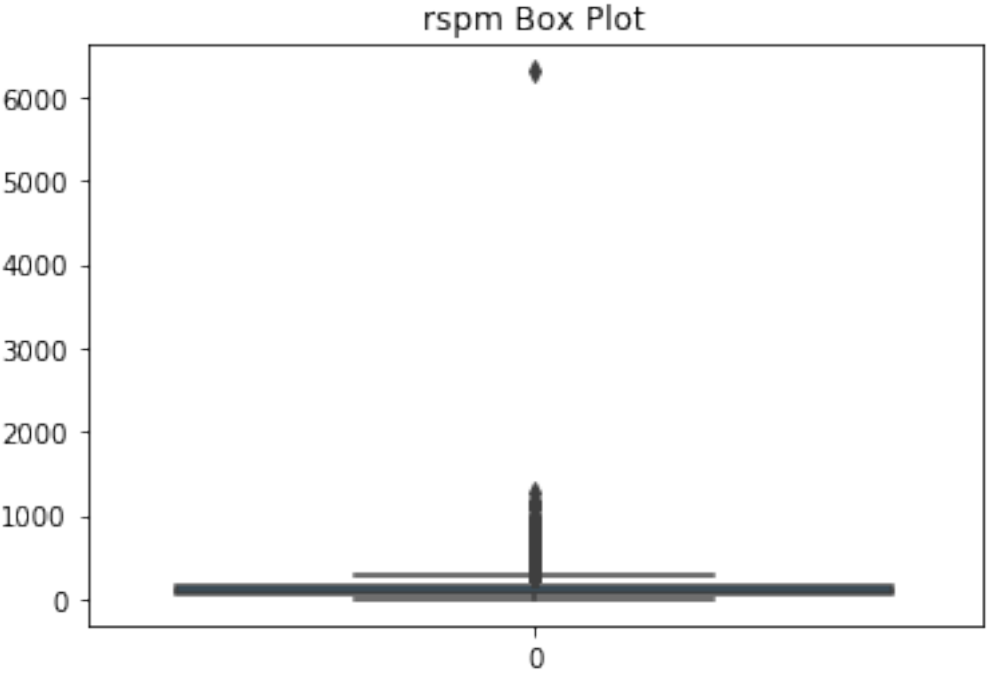

spm Histogram
Frequency
160000
140000
120000
100000
80000
60000
40000
20000
0
0
500
1000
1500
2000
2500
3000
3500

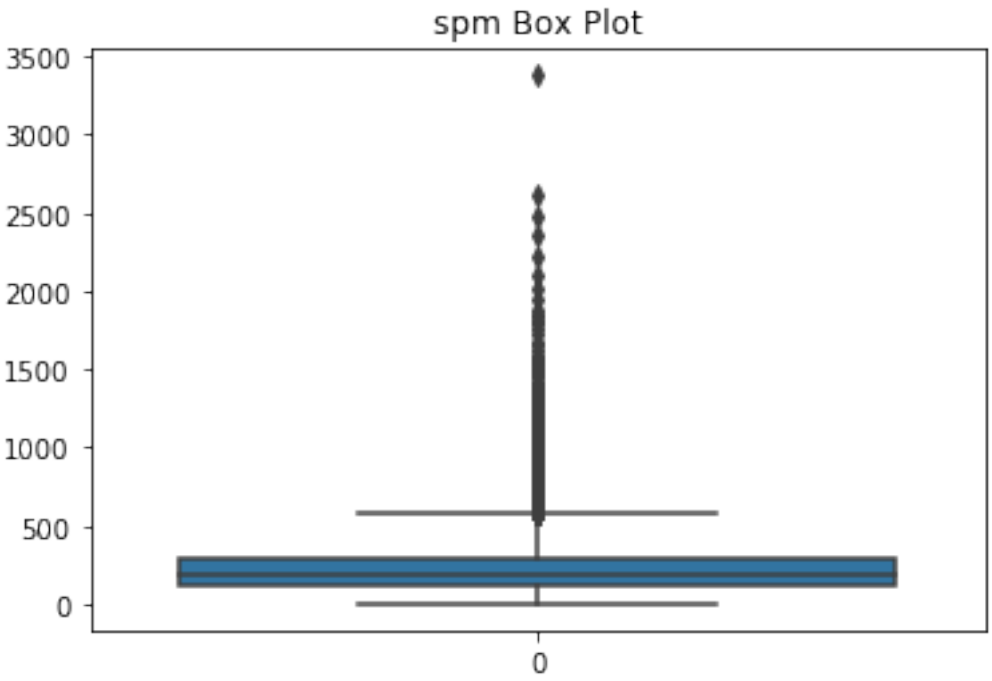


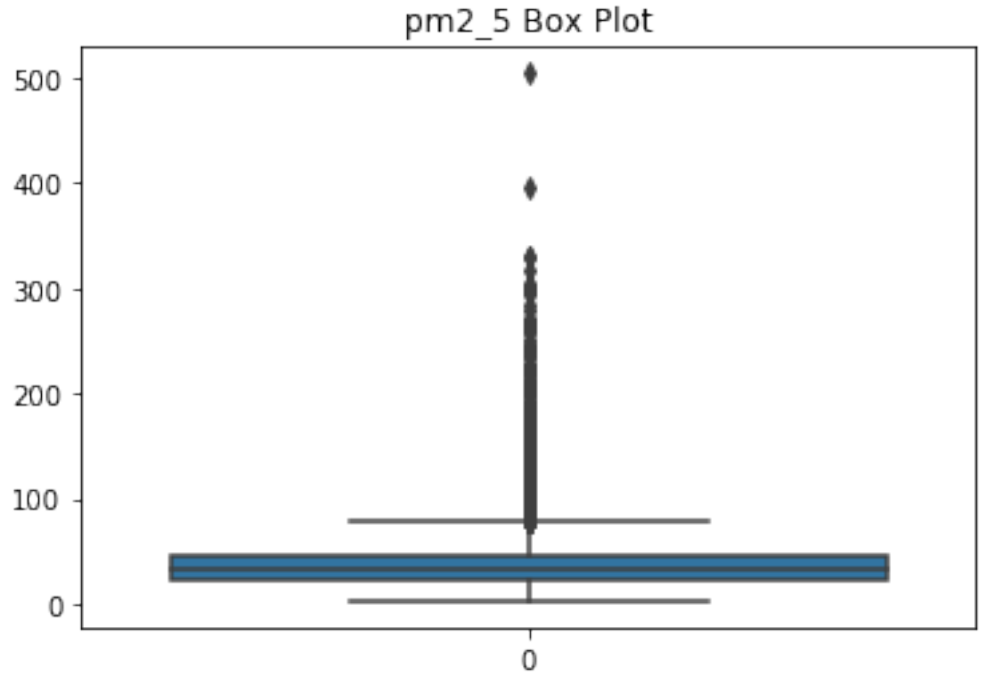
[22]:
```
for column in columns:
    sns.boxplot(df[column])
    plt.title(f'{column} Box Plot')
    plt.show()
```




no2 Box Plot
800
600
400
200
0
0


rspm Box Plot
6000
5000
4000
3000
2000
1000
0
0





Looks like none of the column follow normal distribution we will repalce the `Null or Missing Values` with **median**

[24]:
```
for column in columns:
    df[column].fillna(df[column].median(),inplace=True)
df.isna().sum()
```

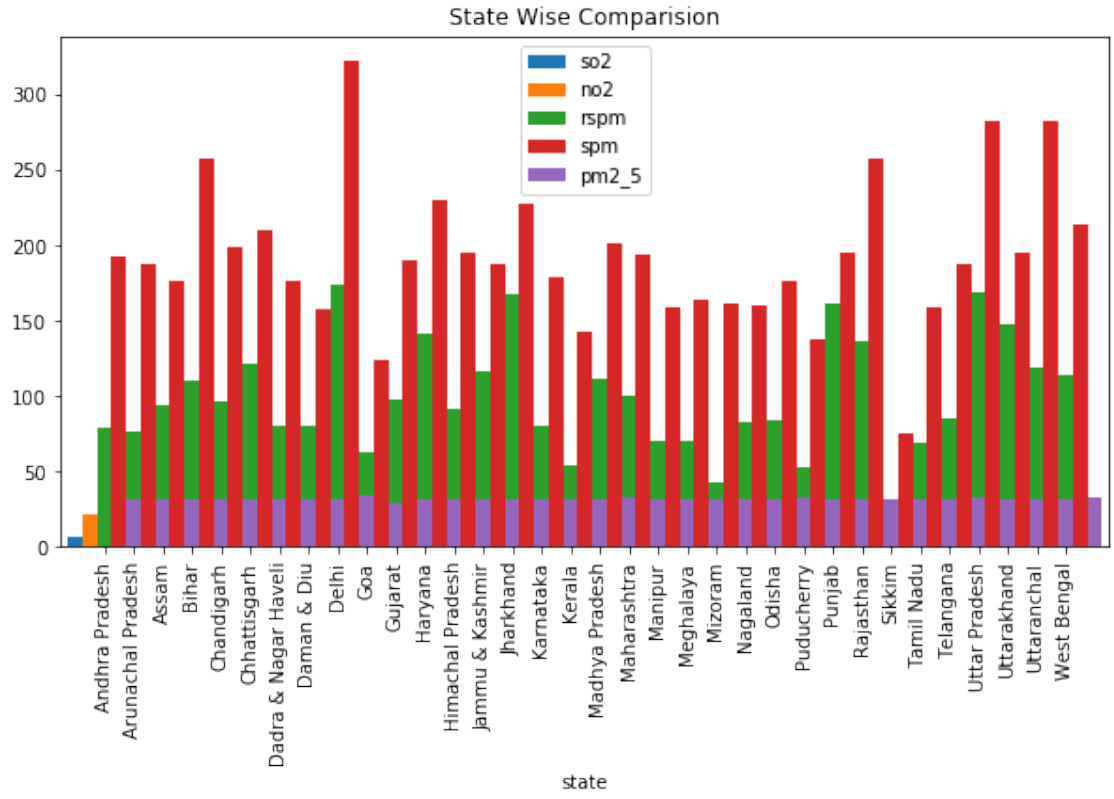
[24]:
```
state       0
location    0
type        0
so2         0
no2         0
rspm        0
spm         0
pm2_5       0
date        0
year        0
dtype: int64
```

1.11 10. Plot barchart state wise of so2, no2, rspm, spm, pm2_5

[32]:
```
new_df = df.groupby(['state']).agg({'so2':'mean','no2':'mean','rspm':
 ↪'mean','spm':'mean','pm2_5':'mean'}).reset_index()
```

[73]:
```
new_df.plot(x='state',kind='bar',stacked=False,title='State Wise␣
 ↪Comparision',figsize=(10,5),width=2.5)
```

[73]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcf978c10a0>
```



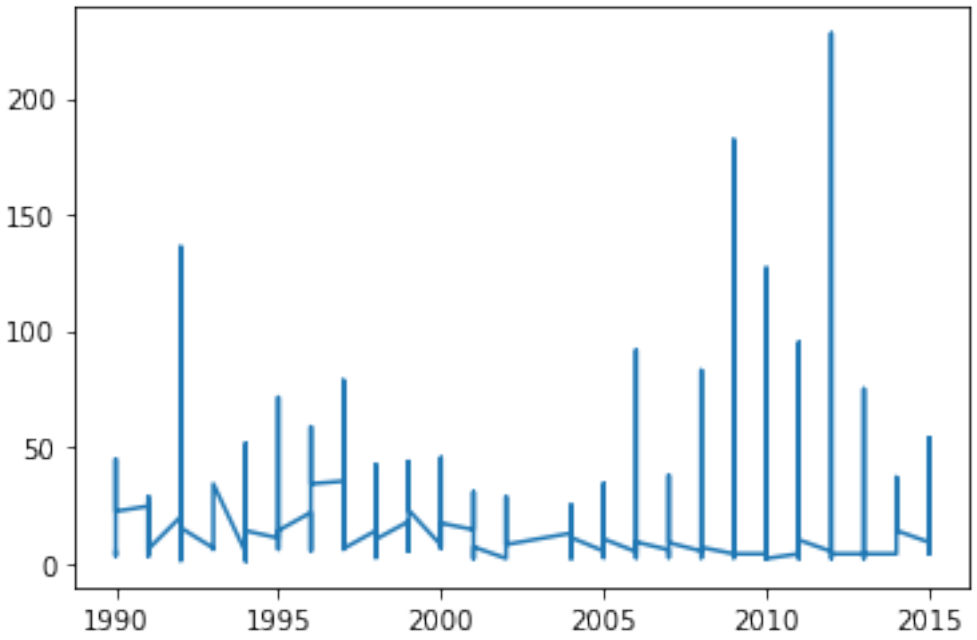
1.12 11. What is the yearly trend in a particular state, say 'Andhra Pradesh'?

[77]:
```
df_s = df[df['state']=='Andhra Pradesh']
```

[84]:
```
plt.plot(df_s['year'],df_s['so2'])
```

```
/usr/lib/python3/dist-packages/matplotlib/cbook/__init__.py:1402: FutureWarning:
Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and
will be removed in a future version.  Convert to a numpy array before indexing
instead.
  ndim = x[:, None].ndim
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:276: FutureWarning:
Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and
will be removed in a future version.  Convert to a numpy array before indexing
instead.
  x = x[:, np.newaxis]
/usr/lib/python3/dist-packages/matplotlib/axes/_base.py:278: FutureWarning:
Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and
will be removed in a future version.  Convert to a numpy array before indexing
instead.
  y = y[:, np.newaxis]
```

[84]: [<matplotlib.lines.Line2D at 0x7fcf934b76d0>]



[ ]: