Project 01 Monterrey

November 4, 2021

1 Classwork 12: Working with the Monterrey Pollution Data

1.1 Team 6 Members

- Alejandro Bañuelos A01244596
- Aline Martínez Briones A00826215
- Marco Ortiz A00823250
- Maximiliano Contreras Serrano A00825377

Determine if vehicle traffic, as represented by time (month, day of month, week day and hour of day) is correlated with pollutants that significantly affect citizen's health. Determine if vehicle verification and/or traffic restriction by zone or plate number are solutions to MMA's pollution. Determine if pollution is related to weather using principal component analysis Create regression models that can predict pollution levels And create machine learning models that can predict pollution emergencies (pollution above 100 IMECAs, that is create a new class where 1 means PM2.5>40.4, and 0 otherwise. Or use the variable CONT which means the same)

1.2 Section 1: Ingestion and Exploratory Analysis

1. Download the data file Monterrey Pollution Data 2.csv
2. Ingest the data into a Python notebook or R script
3. Plot several weather and time variables against pollutants
4. Create a correlation table. What causes pollution?

1.3 Section 2: Principal Component Analysis and Regression

1. Carry out principal component analysis
2. Obtain squared cosines and correlation plot
3. Try to obtain conclusions
4. Create regression models to predict levels of pollution of PM_2.5 and PM2.5 using weather variables and time (as representative for traffic) as predictors (features)

1.4 Section 3: Machine Learning

1. Follow your teacher as he shows you techniques to improve your predictions
2. Download the data file Monterrey Pollution Data 3.csv
3. Build classification models that will allow to predict if pollution levels will exceed legal norms using weather variables as predictors
4. Follow your teacher as he shows you how to transform data to improve predictions
5. Create new classification models to predict if PM2.5 pollution will exceed legal norms.

You will need: Monterrey Pollution Data w Days.csv Max-Min-Avg-Monterrey-Pollution.ipynb

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import tensorflow as tf
import numpy as np
```

[ ]:
```
df = pd.read_csv("Monterrey Pollution Data 2.csv")
df.head()

df = df.drop(["PM2.5Diff"],axis=1)
```

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

[ ]:

	O3	PM10	PM2.5	DayWeek	Hour	\
count	3859.000000	3859.000000	3859.000000	3859.000000	3859.000000	
mean	22.658720	86.049495	27.675823	4.075926	12.037834	
std	16.510561	49.008874	16.849434	1.973040	6.869488	
min	2.000000	8.000000	5.000000	1.000000	0.000000	
25%	10.000000	55.000000	15.000000	2.000000	6.000000	
50%	18.000000	76.000000	25.000000	4.000000	13.000000	
75%	32.000000	104.000000	36.000000	6.000000	18.000000	
max	92.000000	494.000000	123.000000	7.000000	23.000000	

	PRS	RAINF	RH	SR	TOUT	\
count	3859.000000	3859.000000	3859.000000	3859.000000	3859.000000	
mean	719.629956	0.002651	71.048458	0.182582	22.018969	
std	3.695949	0.042447	18.334615	0.261602	7.545535	
min	708.400000	0.000000	12.000000	0.000000	3.030000	
25%	717.200000	0.000000	58.000000	0.000000	17.370000	
50%	719.000000	0.000000	74.000000	0.025000	23.080000	
75%	721.200000	0.000000	87.000000	0.307000	27.280000	
max	734.300000	2.400000	96.000000	0.934000	38.520000	

	WSR	WDV	CONT
count	3859.000000	3859.000000	3859.000000
mean	7.331459	127.522156	0.182172
std	3.828216	96.515310	0.386036
min	0.900000	0.000000	0.000000
25%	4.200000	74.000000	0.000000
50%	6.700000	103.000000	0.000000
75%	10.200000	121.500000	0.000000
max	21.200000	360.000000	1.000000

[ ]:
```
O3 = df ["O3"]
PM_25 = df["PM2.5"]
PM_10 = df["PM10"]

column_names = df.columns
column_names = column_names.drop(["O3","PM2.5","PM10"])

x  = df[column_names]
x.head()
```

[ ]:

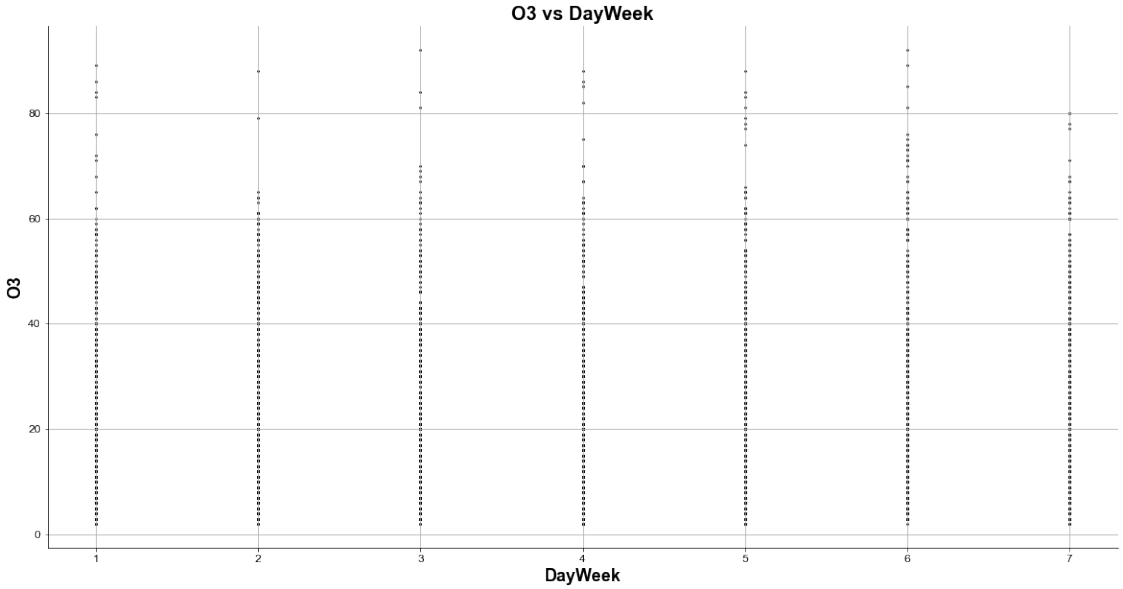
	DayWeek	Hour	PRS	RAINF	RH	SR	TOUT	WSR	WDV	CONT
0	5	1	725.4	0.01	96	0.000	3.91	7.2	344	1
1	5	2	725.0	0.02	96	0.000	3.81	6.3	341	0
2	5	3	724.4	0.00	96	0.000	3.85	5.2	346	0
3	5	5	724.0	0.01	96	0.000	3.79	5.6	336	0
4	5	8	724.3	0.00	96	0.023	3.93	4.8	354	0

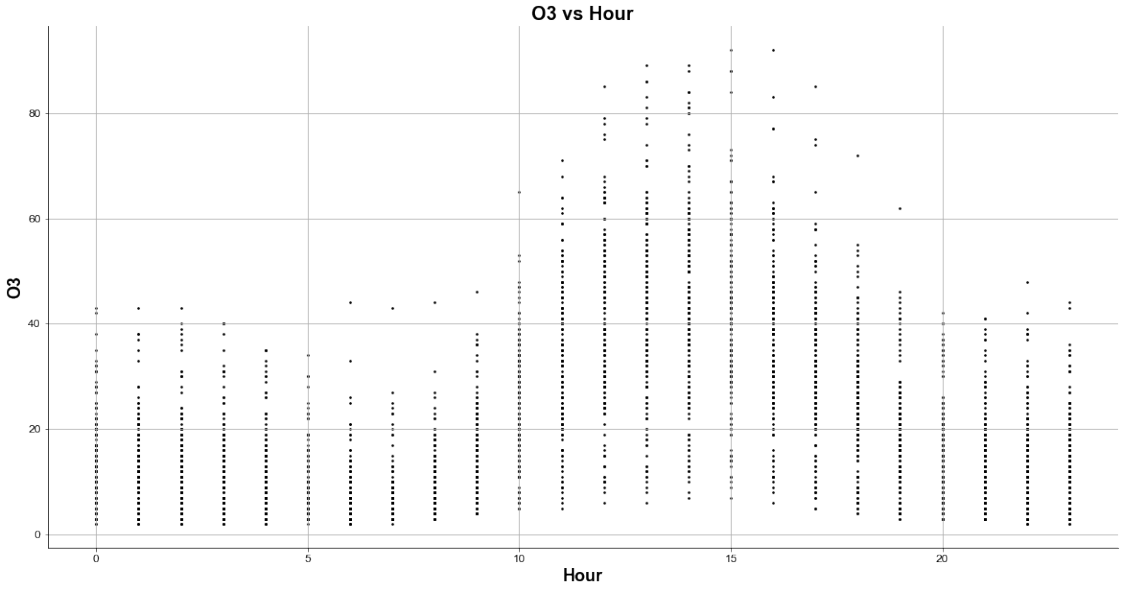
To create a better formating for the following graphs in the document we developed this code

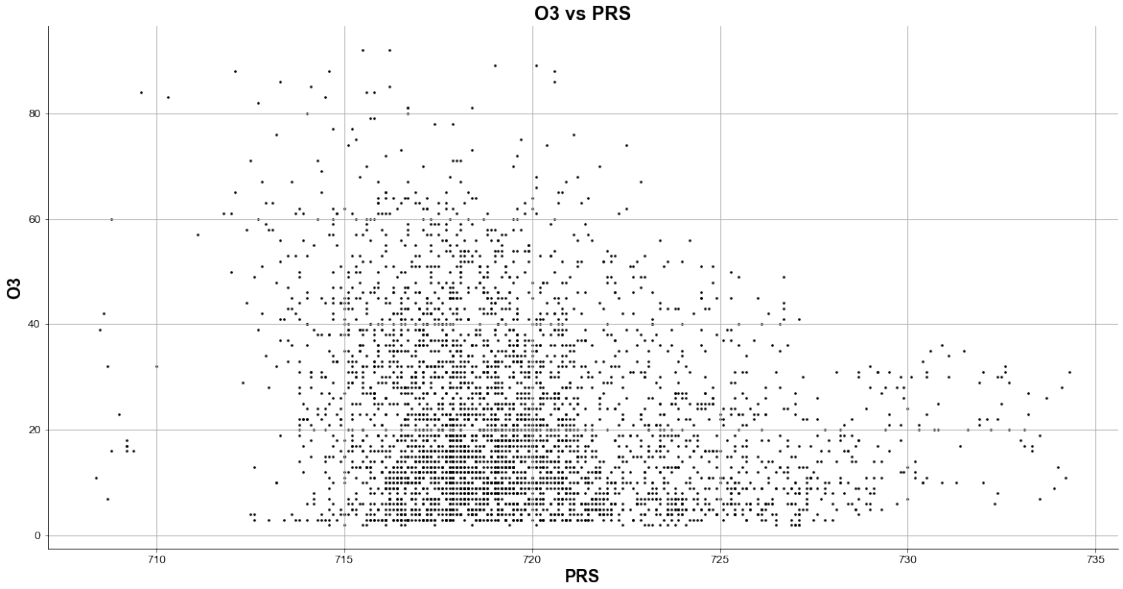
[ ]:
```
fontT = {'family' : 'Arial',
        'weight' : 'bold',
        'size'   : 20}

fontL = {'family' : 'Arial',
        'weight' : 'bold',
        'size'   : 18}

fontAx = {'family' : 'Arial',
        'weight' : 'normal',
        'size'   : 10}

fontTix = {'family' : 'Arial',
        'weight' : 'normal',
        'size'   : 6}
```

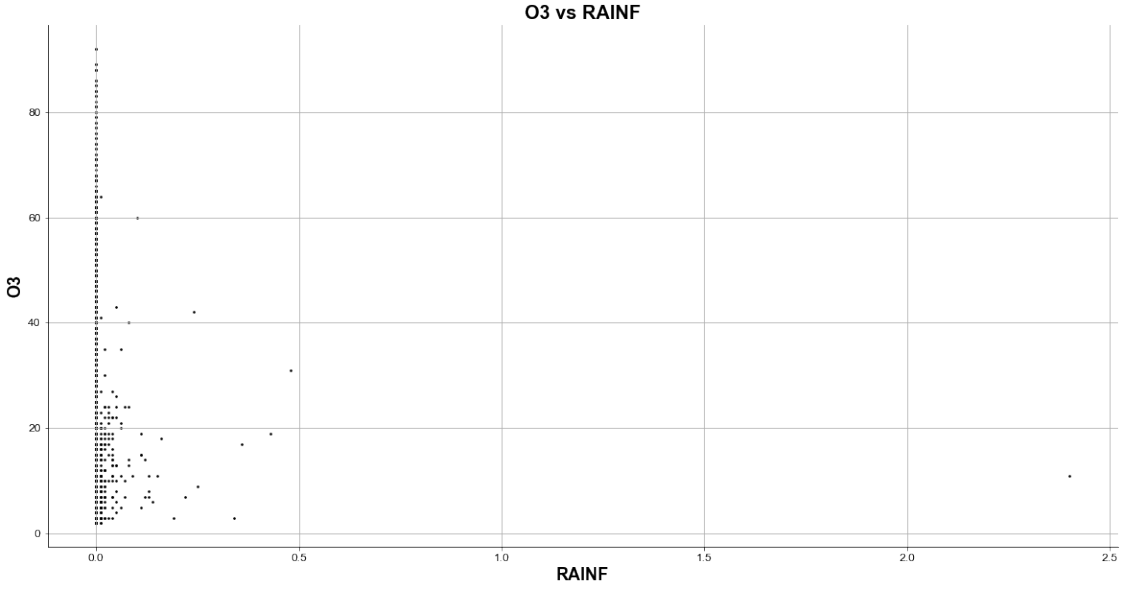
[ ]:
```
## Plots for O3#
for i in range(0,len(column_names)):
    nombre = column_names[i]
    y = O3
    x = df[nombre]
    plt.figure(figsize=(20,10))
    plt.scatter(x,y, marker=".", c = "k" , s = 10)
    plt.title("O3 vs " + nombre ,**fontT)
    plt.xlabel(nombre,**fontL)
    plt.ylabel("O3",**fontL)
    plt.xticks(fontsize = 12 , family = "Arial")
    plt.yticks(fontsize = 12 , family = "Arial")
    plt.gca().spines['top'].set_visible(False)
```

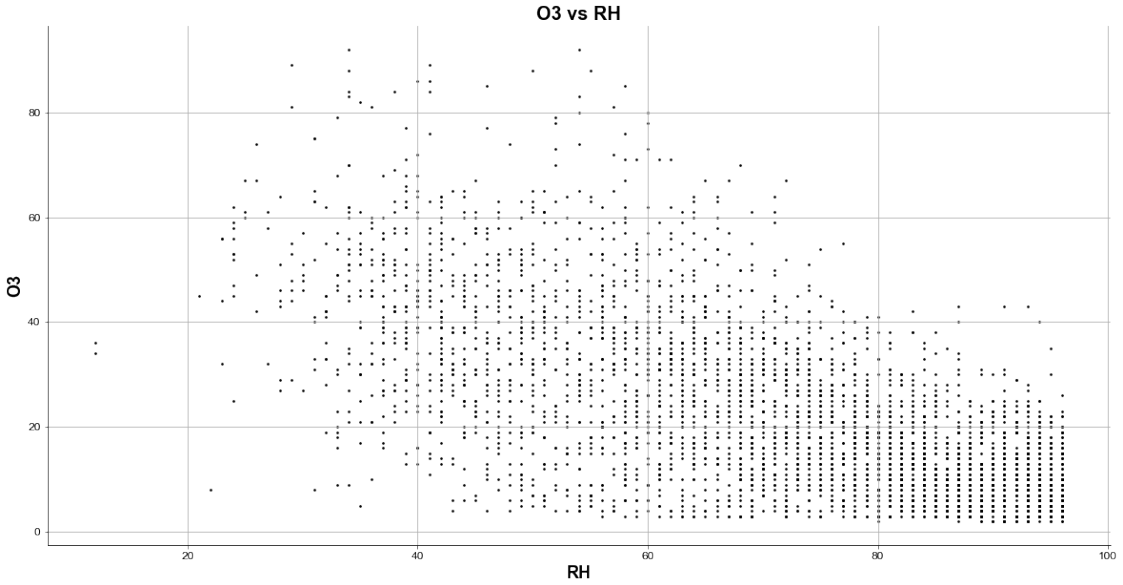
```
    plt.gca().spines['right'].set_visible(False)
    plt.grid("On")
    plt.show()
```

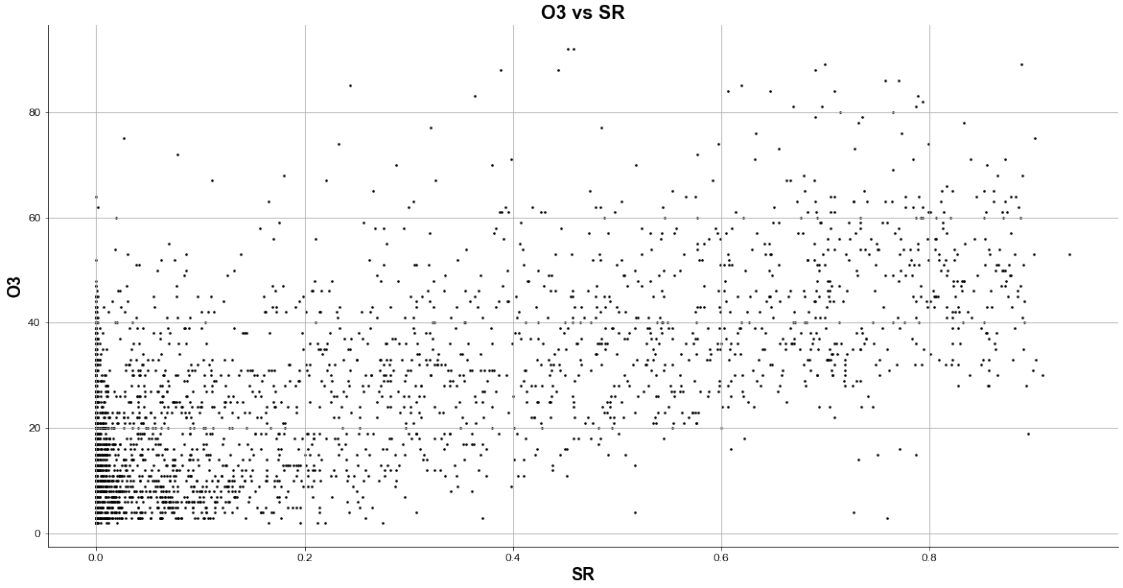


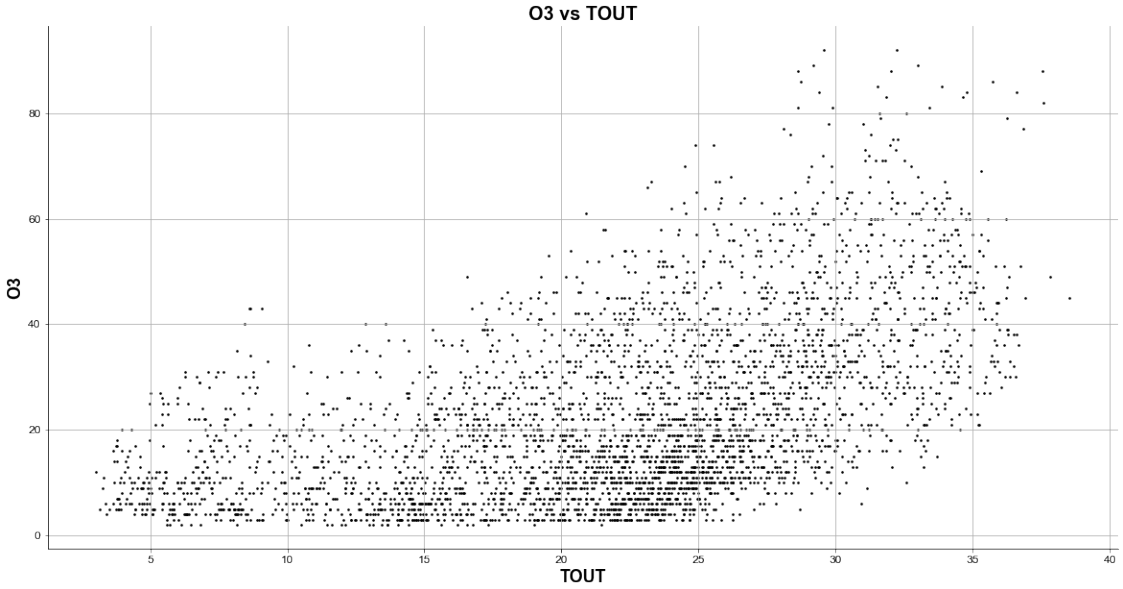


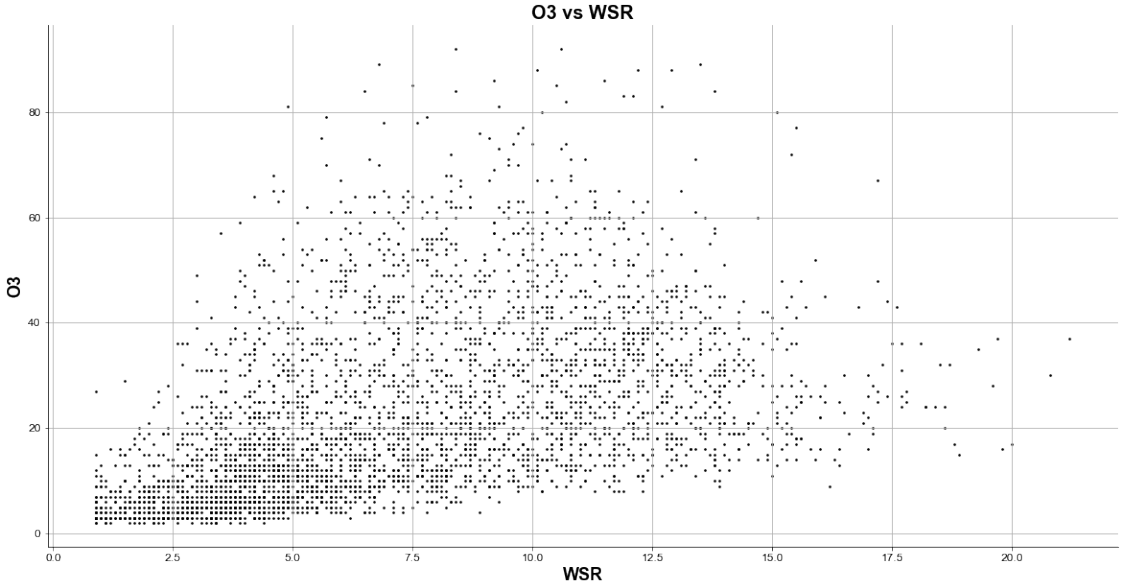

O3 vs PRS
O3
PRS
0
20
40
60
80
710
715
720
725
730
735

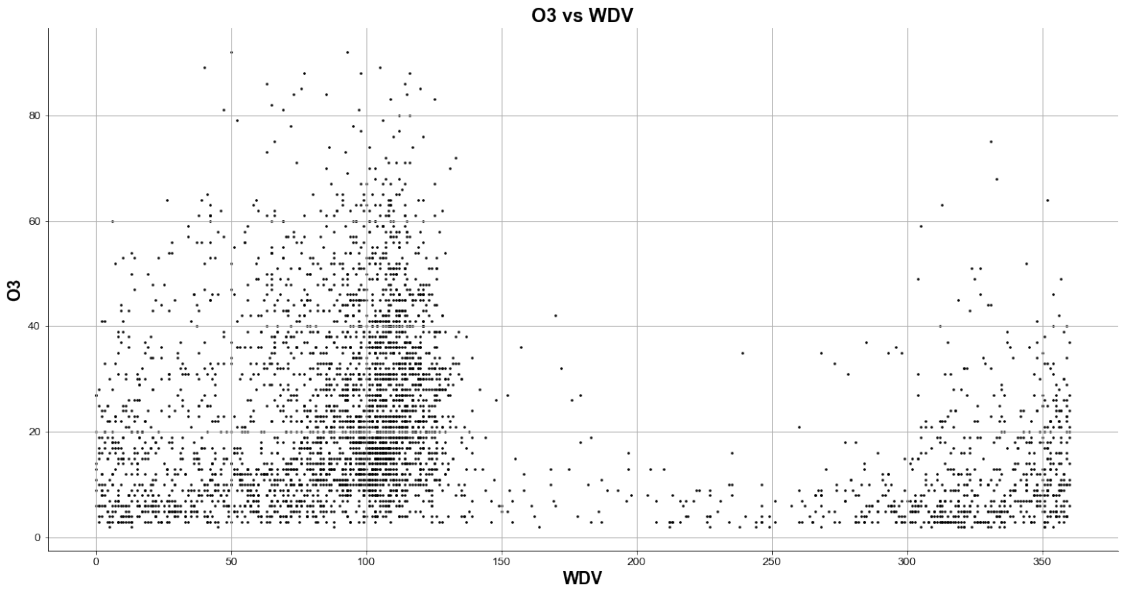

O3 vs RAINF
O3
RAINF
0
20
40
60
80
0.0
0.5
1.0
1.5
2.0
2.5

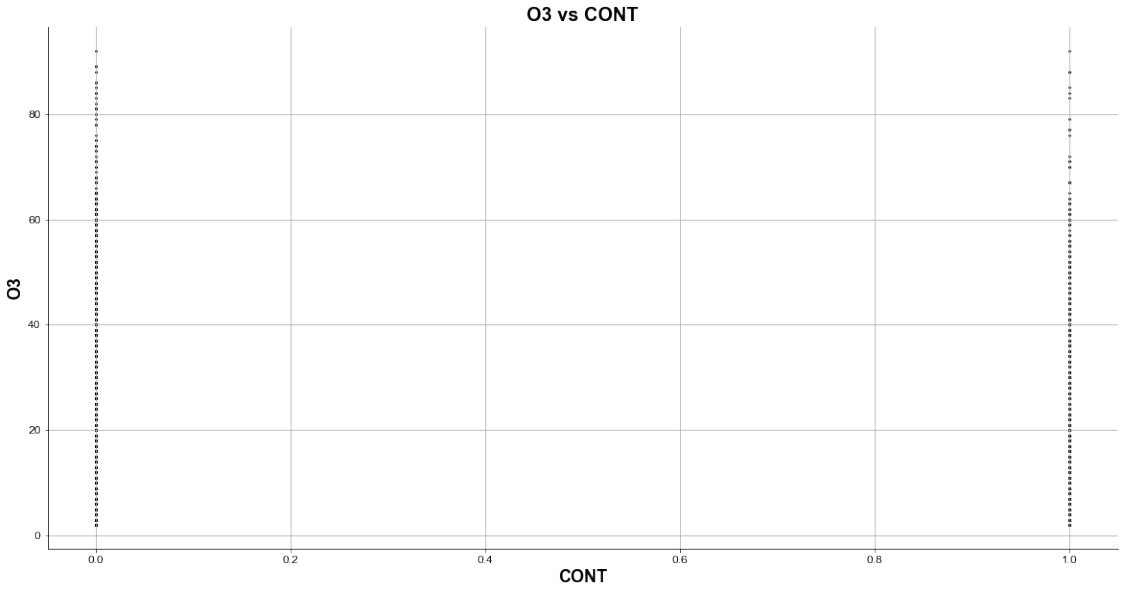

O3 vs RH
O3
RH
0
20
40
60
80
20
40
60
80
100


O3 vs SR
O3
SR
0
20
40
60
80
0.0
0.2
0.4
0.6
0.8


O3 vs TOUT
O3
TOUT
0
20
40
60
80
5
10
15
20
25
30
35
40

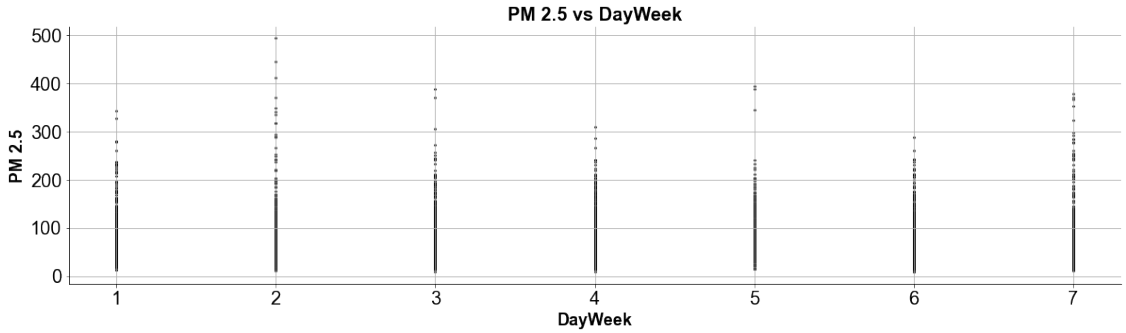

O3 vs WSR
O3
WSR
0
20
40
60
80
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

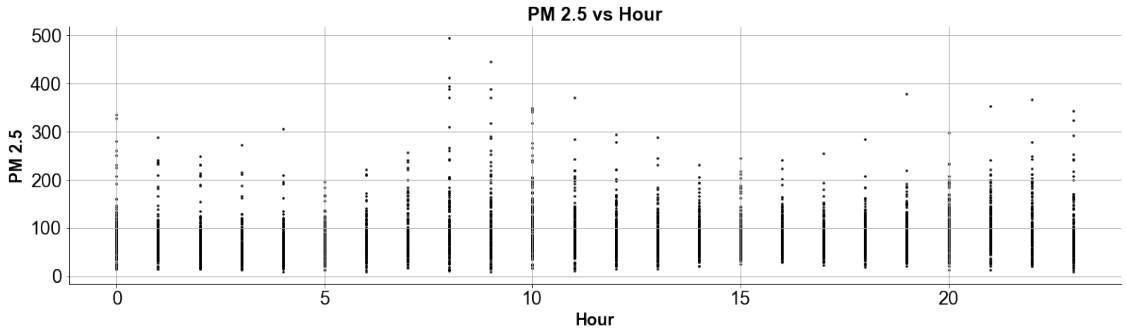


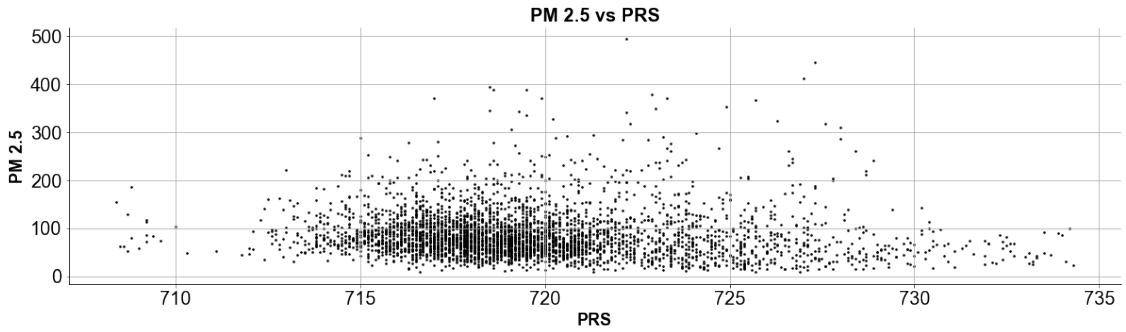


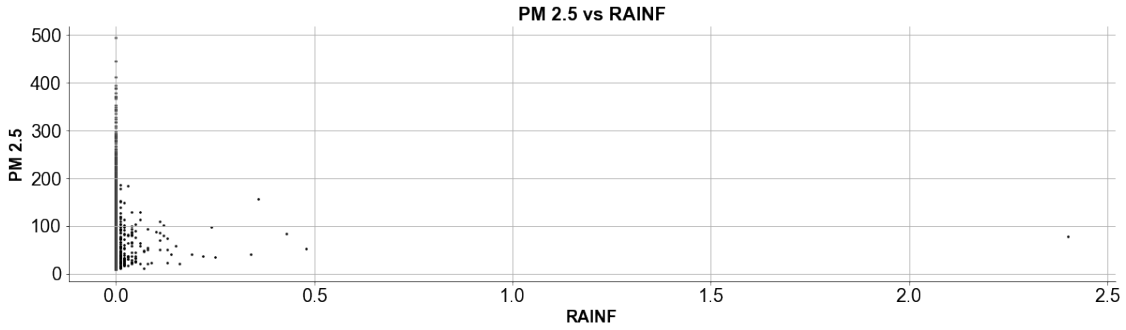
[ ]:
```
## Plots for PM 2.5#
for i in range(0,len(column_names)):
    nombre = column_names[i]
    y = PM_10
    x = df[nombre]
    plt.rcParams["figure.figsize"] = (20,5)
    plt.scatter(x,y, marker=".", c = "k" , s = 10)
    plt.title("PM 2.5 vs " + nombre ,**fontT)
```

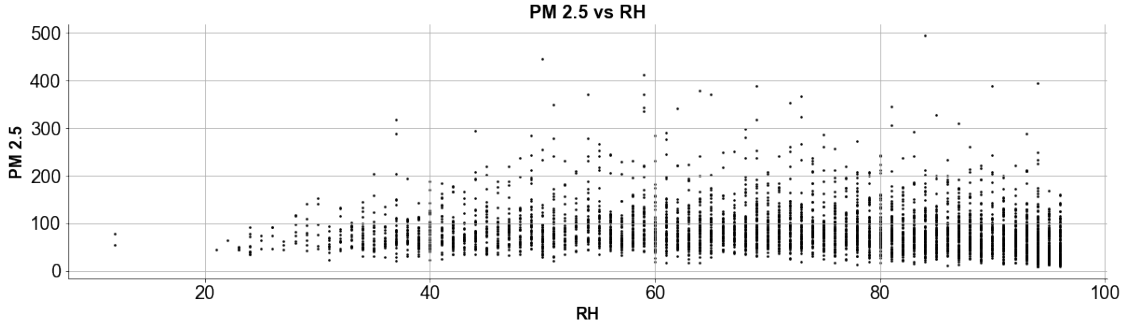
```
    plt.xlabel(nombre,**fontL)
    plt.ylabel("PM 2.5",**fontL)
    plt.xticks(fontsize = 20 , family = "Arial")
    plt.yticks(fontsize = 20 , family = "Arial")
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.grid("On")
    plt.show()
```

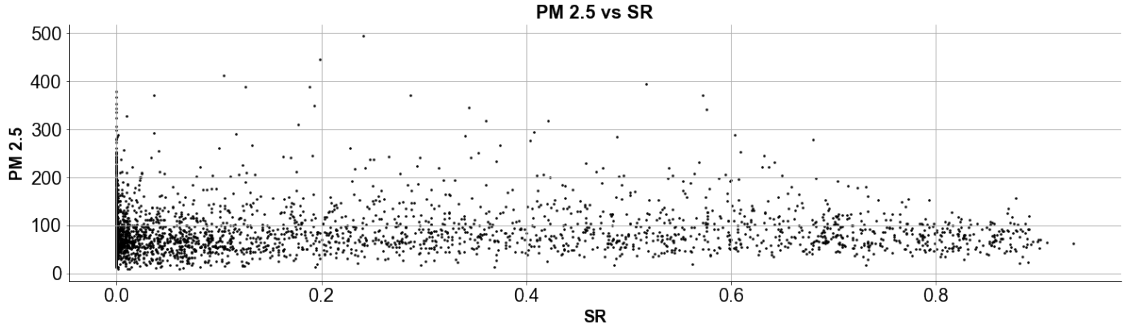


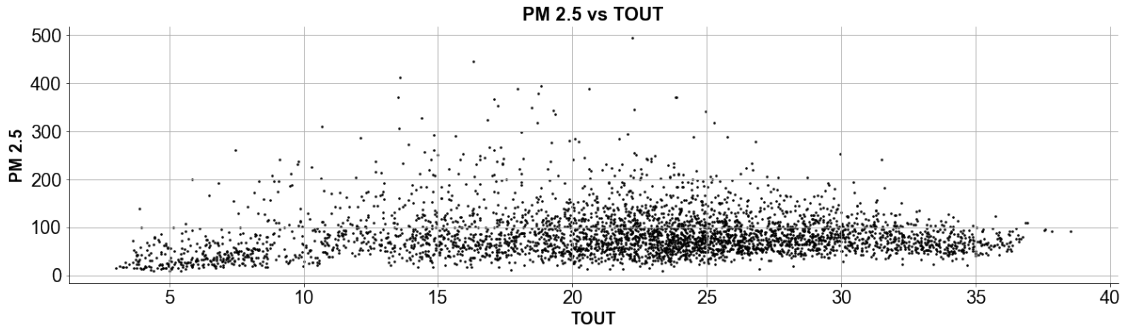


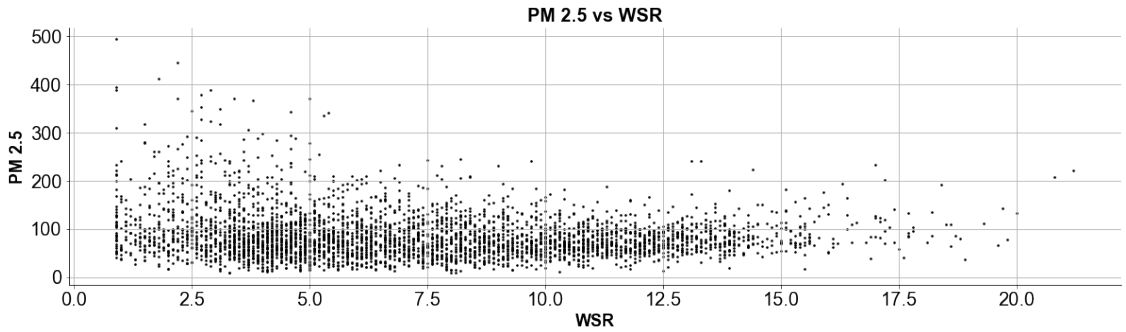


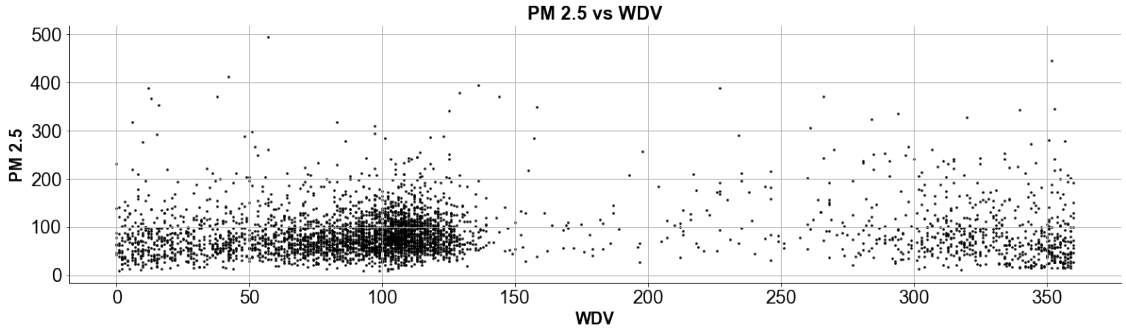

PM 2.5 vs RAINF
PM 2.5
RAINF
0
100
200
300
400
500
0.0
0.5
1.0
1.5
2.0
2.5

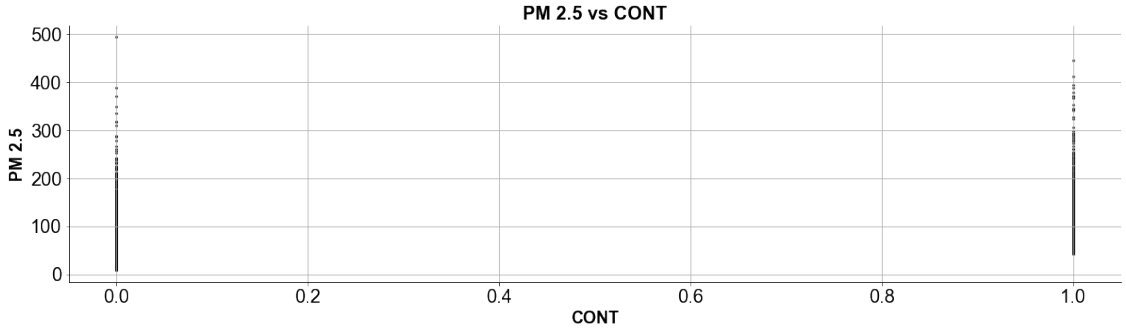

PM 2.5 vs RH
PM 2.5
RH
0
100
200
300
400
500
20
40
60
80
100


PM 2.5 vs SR
PM 2.5
SR
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8


PM 2.5 vs TOUT
PM 2.5
TOUT
0
100
200
300
400
500
5
10
15
20
25
30
35
40

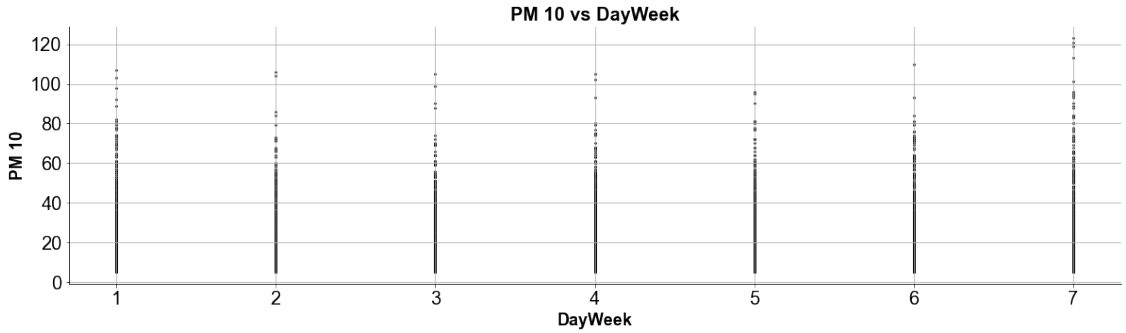


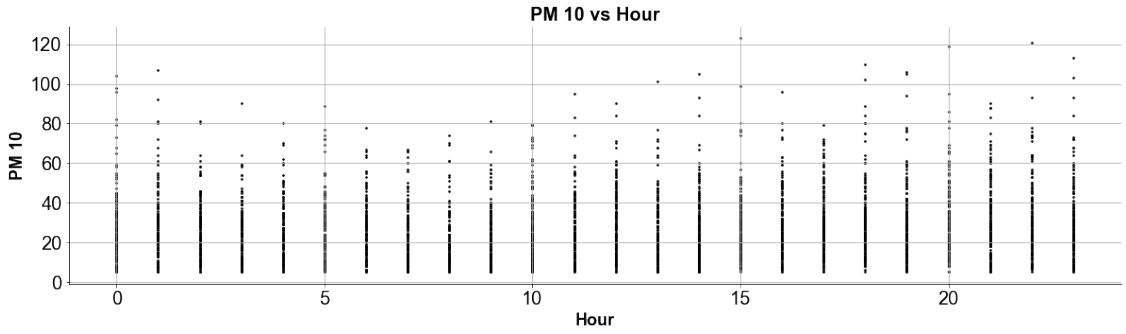


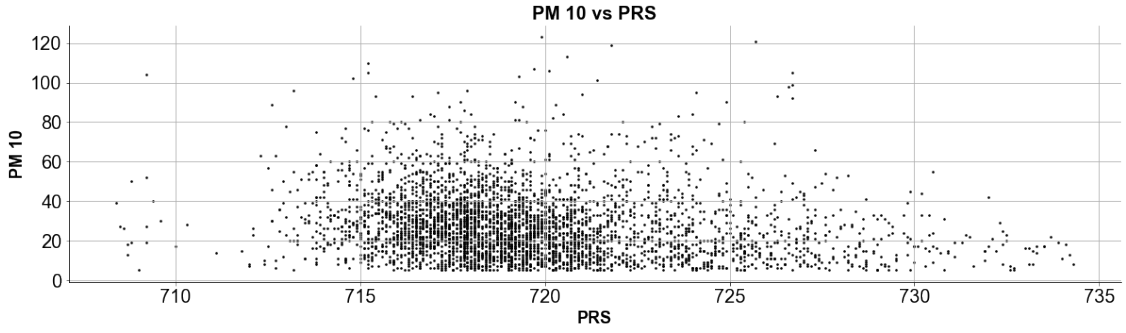


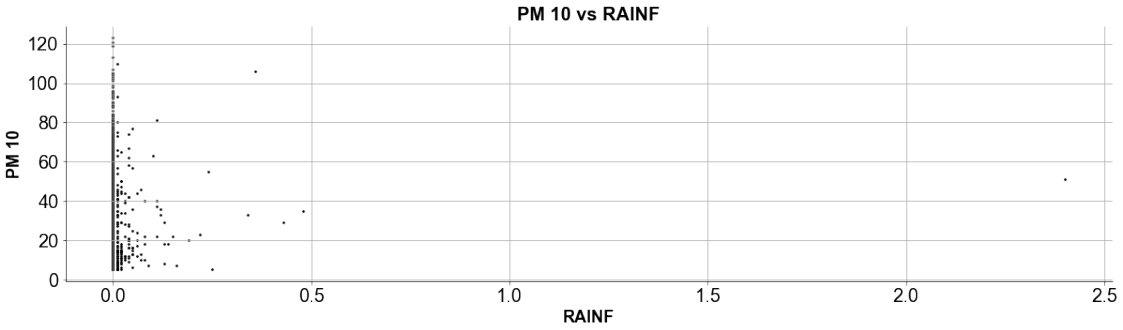
[ ]:
```
## Plots for PM 10#
for i in range(0,len(column_names)):
    nombre = column_names[i]
    y = PM_25
    x = df[nombre]
    plt.rcParams["figure.figsize"] = (20,5)
    plt.scatter(x,y, marker=".", c = "k" , s = 10)
    plt.title("PM 10 vs " + nombre ,**fontT)
```

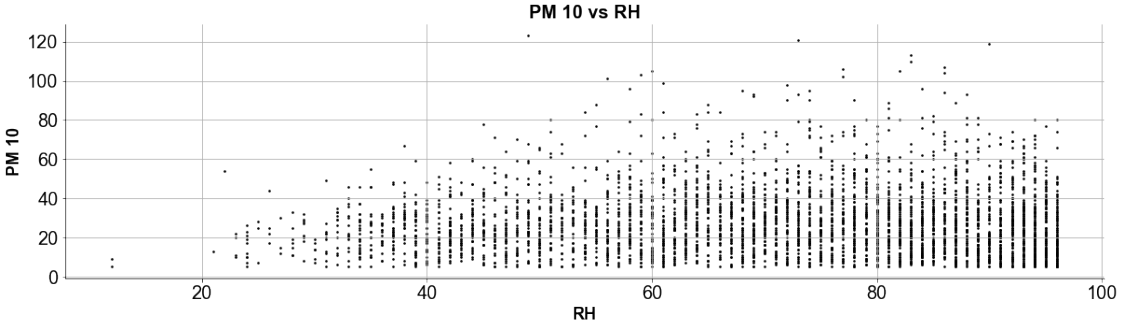
```
    plt.xlabel(nombre,**fontL)
    plt.ylabel("PM 10",**fontL)
    plt.xticks(fontsize = 20 , family = "Arial")
    plt.yticks(fontsize = 20 , family = "Arial")
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.grid("On")
    plt.show()
```

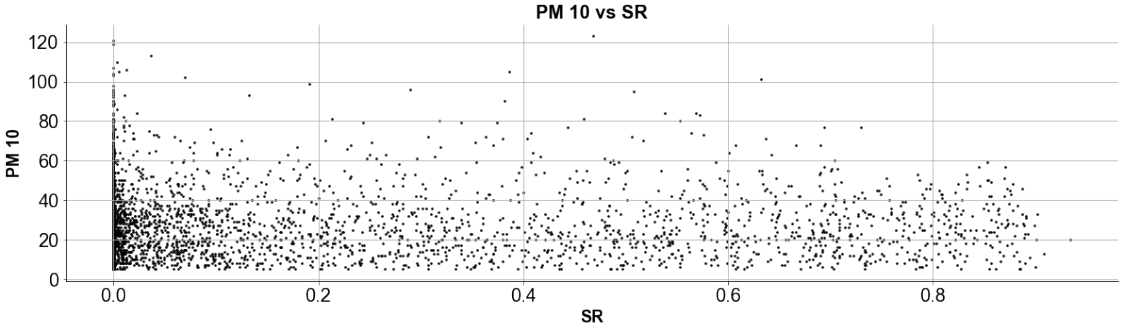


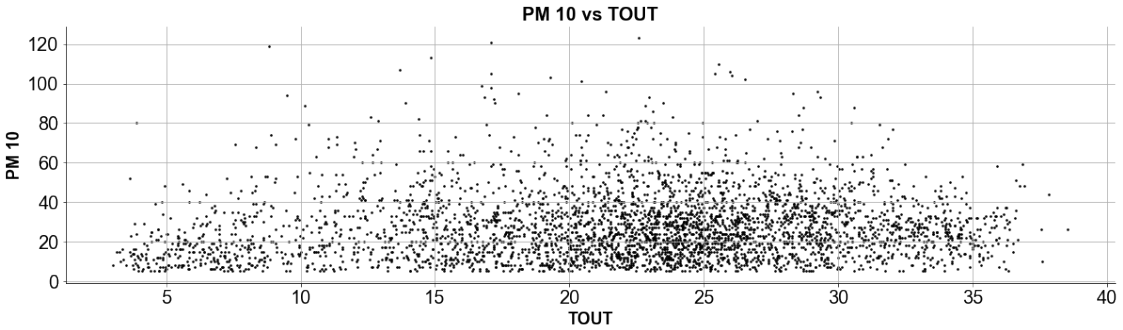


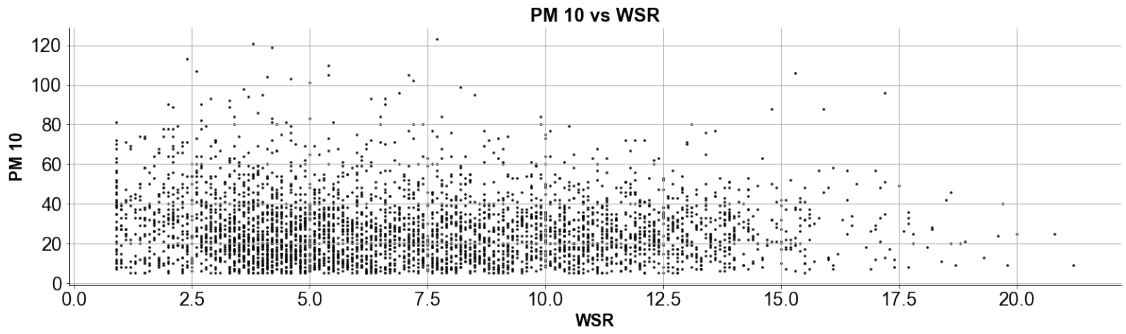


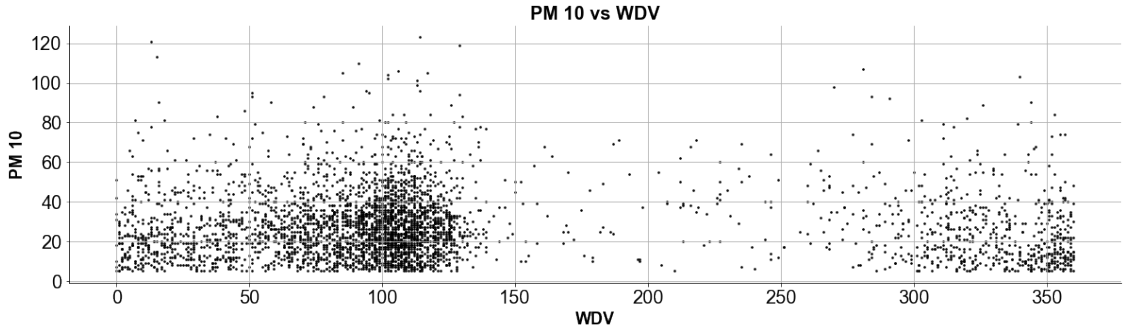

PM 10 vs RAINF
PM 10
RAINF
0
20
40
60
80
100
120
0.0
0.5
1.0
1.5
2.0
2.5

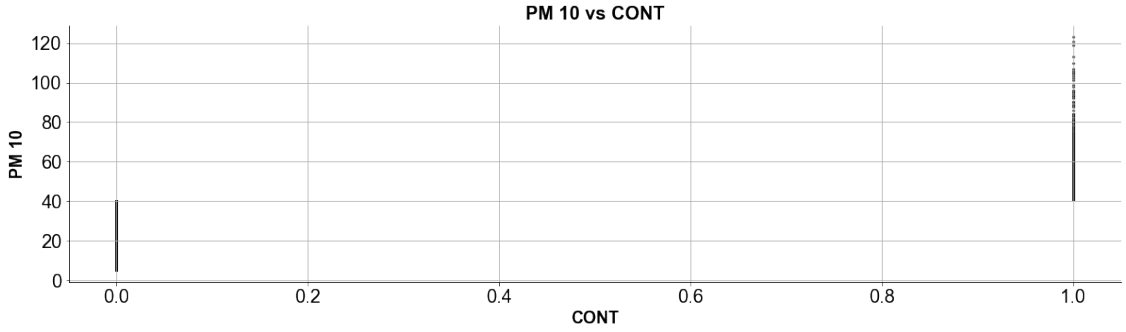

PM 10 vs RH
PM 10
RH
0
20
40
60
80
100
120
20
40
60
80
100


PM 10 vs SR
PM 10
SR
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8


PM 10 vs TOUT
PM 10
TOUT
0
20
40
60
80
100
120
5
10
15
20
25
30
35
40

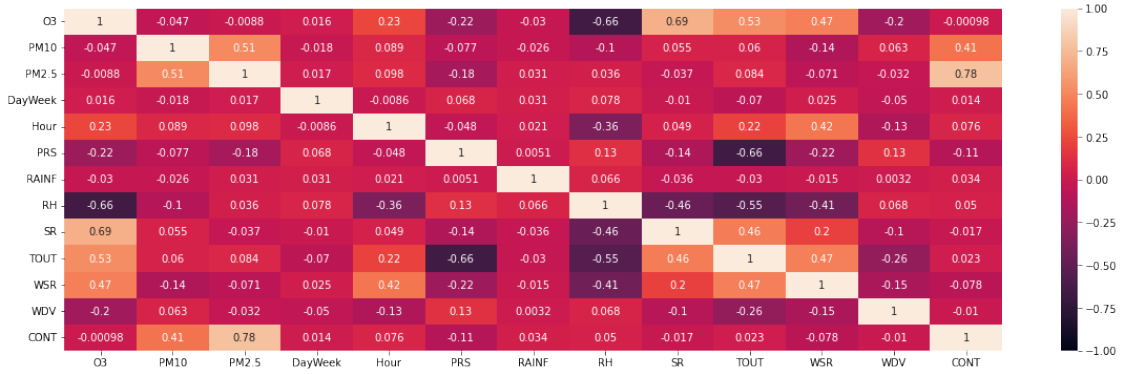






[ ]:
```
##Here is the correlation plot of the system##

corr_df = df

corr = corr_df.corr()
corr.style.background_gradient(cmap="coolwarm")

plt.figure(figsize=(20, 6))
```

```
heatmap = sns.heatmap(corr, vmin=-1, vmax=1, annot=True)
```



1.5 PCA O3

[ ]:
```
from sklearn.decomposition import PCA
from numpy import set_printoptions
from numpy import mean
from numpy import std
from numpy import zeros
from numpy import sqrt

set_printoptions(precision=6,suppress=True)
```

[ ]:
```
#Only O3 Dataframe
O3_df = df.drop(["PM10","PM2.5"],axis=1)
header_row=O3_df.columns.values

y=O3_df["O3"]
```

[ ]:
```
M = mean(O3_df.T, axis=1)
print(M)

#Std Deviation
st = std(O3_df.T, axis=1,ddof=1)

#Normalize data
Xdat = (O3_df - M)/st

print(st)
print(Xdat)
```

```
O3          22.658720
DayWeek      4.075926
Hour        12.037834
```

```
PRS        719.629956
RAINF        0.002651
RH          71.048458
SR           0.182582
TOUT        22.018969
WSR          7.331459
WDV        127.522156
CONT         0.182172
dtype: float64
O3         16.510561
DayWeek     1.973040
Hour        6.869488
PRS         3.695949
RAINF       0.042447
RH         18.334615
SR          0.261602
TOUT        7.545535
WSR         3.828216
WDV        96.515310
CONT        0.386036
dtype: float64
            O3   DayWeek      Hour       PRS     RAINF        RH        SR  \
0    -0.766704  0.468350 -1.606791  1.561181  0.173134  1.360898 -0.697939
1    -0.948406  0.468350 -1.461220  1.452954  0.408722  1.360898 -0.697939
2    -0.948406  0.468350 -1.315649  1.290614 -0.062453  1.360898 -0.697939
3    -0.948406  0.468350 -1.024506  1.182388  0.173134  1.360898 -0.697939
4    -1.069541  0.468350 -0.587793  1.263558 -0.062453  1.360898 -0.610019
…         …         …         …         …         …         …
3854 -0.706137 -1.052146  1.304634  0.749481 -0.062453  1.033648 -0.697939
3855 -0.766704 -1.052146  1.450205  0.695368 -0.062453  1.033648 -0.697939
3856 -0.585003 -1.052146  1.595776  0.695368 -0.062453  0.979107 -0.697939
3857 -0.585003 -0.545314 -1.752363  0.587141 -0.062453  0.979107 -0.697939
3858 -0.766704 -0.545314 -1.606791  0.505971 -0.062453  0.979107 -0.697939

          TOUT       WSR       WDV      CONT
0    -2.399958 -0.034339  2.242938  2.118530
1    -2.413211 -0.269436  2.211855 -0.471903
2    -2.407910 -0.556776  2.263660 -0.471903
3    -2.415862 -0.452289  2.160049 -0.471903
4    -2.397308 -0.661263  2.346548 -0.471903
…         …         …         …         …
3854 -0.630700 -0.034339 -0.399130 -0.471903
3855 -0.638652 -0.347801 -0.181548 -0.471903
3856 -0.642628 -0.478410 -0.181548 -0.471903
3857 -0.638652 -1.183700 -0.026132 -0.471903
3858 -0.625399 -1.653893 -0.326603 -0.471903

[3859 rows x 11 columns]
```

[ ]:
```
# create the PCA instance
pca = PCA()

# fit on data
pca = pca.fit(Xdat)

#Q eigenvectors
Q=pca.components_.T

#Explained deviations
s=sqrt(pca.explained_variance_)
```

[ ]:
```
# transform data obtain sample scores
F = pca.transform(Xdat)
print("Sample scores")

## Sample Scores ##
sample_scores = pd.DataFrame(F)
print(sample_scores)
```

```
Sample scores
            0         1         2         3         4         5         6   \
0    -3.447083 -0.590549 -0.123222  0.430742  2.143638 -1.290280 -0.931279
1    -3.560162 -1.505163 -0.511376  0.199182  0.712194  0.411215 -1.218664
2    -3.578845 -1.451739 -0.684835  0.035682  0.534186  0.035283 -1.280729
3    -3.437927 -1.389494 -0.380973  0.026385  0.597085  0.195431 -1.235541
4    -3.470018 -1.547614 -0.292443 -0.186225  0.638949 -0.055318 -1.304395
…          …         …         …         …         …         …         …
3854 -1.096254 -0.317503  1.100140 -1.246479 -0.441867  0.253446  0.968228
3855 -1.227664 -0.312129  1.036848 -1.322029 -0.294999  0.217817  0.859950
3856 -1.136594 -0.375324  1.055094 -1.317059 -0.231779  0.178805  0.909479
3857 -2.188889  0.218335 -1.272297  0.096377 -0.398644  0.432740  0.561803
3858 -2.328972  0.415191 -1.263652  0.175648 -0.488844  0.393106  0.844059

            7         8         9         10
0     1.943569  0.209988  0.182946  0.247274
1     1.134130 -0.161684  0.299474  0.278342
2     0.830481 -0.282142  0.403984  0.260889
3     0.762736 -0.407239  0.439700  0.327095
4     0.405050 -0.657201  0.332418  0.266779
…          …         …         …         …
3854 -0.109994 -0.911110 -0.047581 -0.319075
3855 -0.426539 -1.004148 -0.017934 -0.365335
3856 -0.578667 -1.038772  0.131200 -0.449340
3857  0.278782  0.433659  0.136225 -0.352271
3858 -0.184054  0.390829  0.105395 -0.317289

[3859 rows x 11 columns]
```

[ ]:
```
# Obtain Factor Loadings
L = zeros(shape=(Q.shape))
for col in range(0,len(s)-1):
    L[:,col] = Q[:,col] * s[col]
print("Factor Loadings")

columms_factorLoading = []

for i in range(0,11):
    nombre =  "F" + str(i)
    columms_factorLoading.append(nombre)


## Factor Loadings Dataframe ##
factor_loadings = pd.DataFrame(L, columns= columms_factorLoading)
print(factor_loadings)
```

```
Factor Loadings
          F0        F1        F2        F3        F4        F5        F6  \
0   0.833634 -0.230123 -0.163366  0.191511  0.132423 -0.043402  0.082081
1  -0.049659 -0.129622  0.291698  0.775800 -0.196031 -0.199224 -0.450027
2   0.432768 -0.178685  0.697528 -0.299901  0.071013 -0.118294  0.029355
3  -0.478246 -0.740930  0.123608  0.077532  0.118667 -0.103267  0.311122
4  -0.054981  0.124236  0.306043  0.297777  0.505460  0.732981  0.072038
5  -0.772080  0.314218  0.033979  0.120842 -0.147133  0.039108  0.005047
6   0.661140 -0.218016 -0.443858  0.289645  0.209734 -0.037838  0.129591
7   0.831701  0.363298 -0.101878 -0.032921 -0.088372  0.084712 -0.126625
8   0.659360 -0.123634  0.388265 -0.156353 -0.157256  0.081899 -0.203281
9  -0.304658 -0.266161 -0.239797 -0.309654  0.465099  0.001816 -0.678739
10  0.000510  0.453457  0.203244  0.103392  0.598275 -0.584906  0.119646

          F7        F8        F9  F10
0   0.110577 -0.021171  0.355351  0.0
1  -0.106839  0.037925 -0.015866  0.0
2  -0.316680 -0.288551  0.017620  0.0
3   0.110775  0.108130 -0.182417  0.0
4  -0.004714  0.034529 -0.004703  0.0
5   0.233540 -0.434658  0.015331  0.0
6  -0.004665 -0.351317 -0.185443  0.0
7  -0.095349  0.090601 -0.252992  0.0
8   0.539752  0.001194 -0.072558  0.0
9   0.003235 -0.060783 -0.001865  0.0
10  0.144461  0.077956 -0.031999  0.0
```

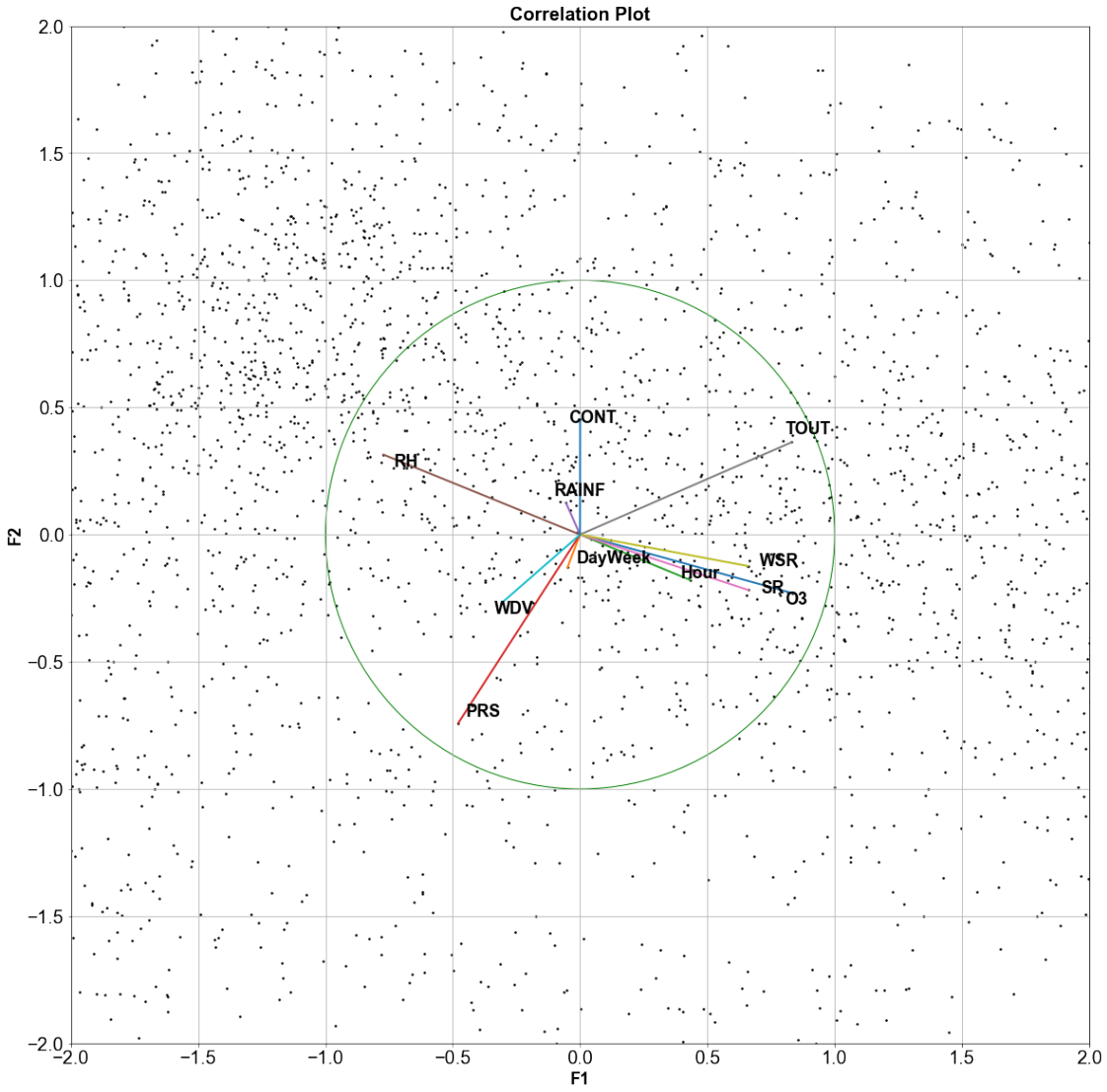
[ ]:
```
#Obtain squared cosines
COS2=L**2
print("Square Cosines")
```

```
## COS2  Dataframe ##
COS2_DF = pd.DataFrame(COS2)
print(COS2_DF)
```

```
Square Cosines
              0         1         2         3         4         5         6   \
0   6.949455e-01  0.052956  0.026688  0.036677  0.017536  0.001884  0.006737
1   2.466019e-03  0.016802  0.085088  0.601866  0.038428  0.039690  0.202524
2   1.872880e-01  0.031928  0.486545  0.089940  0.005043  0.013993  0.000862
3   2.287190e-01  0.548977  0.015279  0.006011  0.014082  0.010664  0.096797
4   3.022898e-03  0.015435  0.093662  0.088671  0.255490  0.537262  0.005189
5   5.961082e-01  0.098733  0.001155  0.014603  0.021648  0.001529  0.000025
6   4.371062e-01  0.047531  0.197010  0.083894  0.043988  0.001432  0.016794
7   6.917265e-01  0.131985  0.010379  0.001084  0.007810  0.007176  0.016034
8   4.347556e-01  0.015285  0.150750  0.024446  0.024729  0.006707  0.041323
9   9.281663e-02  0.070841  0.057502  0.095885  0.216317  0.000003  0.460687
10  2.595911e-07  0.205623  0.041308  0.010690  0.357933  0.342115  0.014315

          7         8         9    10
0   0.012227  0.000448  0.126274  0.0
1   0.011415  0.001438  0.000252  0.0
2   0.100286  0.083262  0.000310  0.0
3   0.012271  0.011692  0.033276  0.0
4   0.000022  0.001192  0.000022  0.0
5   0.054541  0.188927  0.000235  0.0
6   0.000022  0.123424  0.034389  0.0
7   0.009091  0.008209  0.064005  0.0
8   0.291332  0.000001  0.005265  0.0
9   0.000010  0.003695  0.000003  0.0
10  0.020869  0.006077  0.001024  0.0
```

[ ]:
```
import random

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.grid()

circ=plt.Circle((0,0), radius=1, color='g', fill=False)
ax.add_patch(circ)
ax.set_aspect('equal')

plt.scatter(L[:,0],L[:,1],marker=".", c = "k" , s = 10)
plt.scatter(F[:,0],F[:,1],marker=".", c = "k" , s = 10)


for row in range(0,len(s)):
    plt.plot([0, L[row,0]], [0,L[row,1]], linewidth=2,label='X1')
```

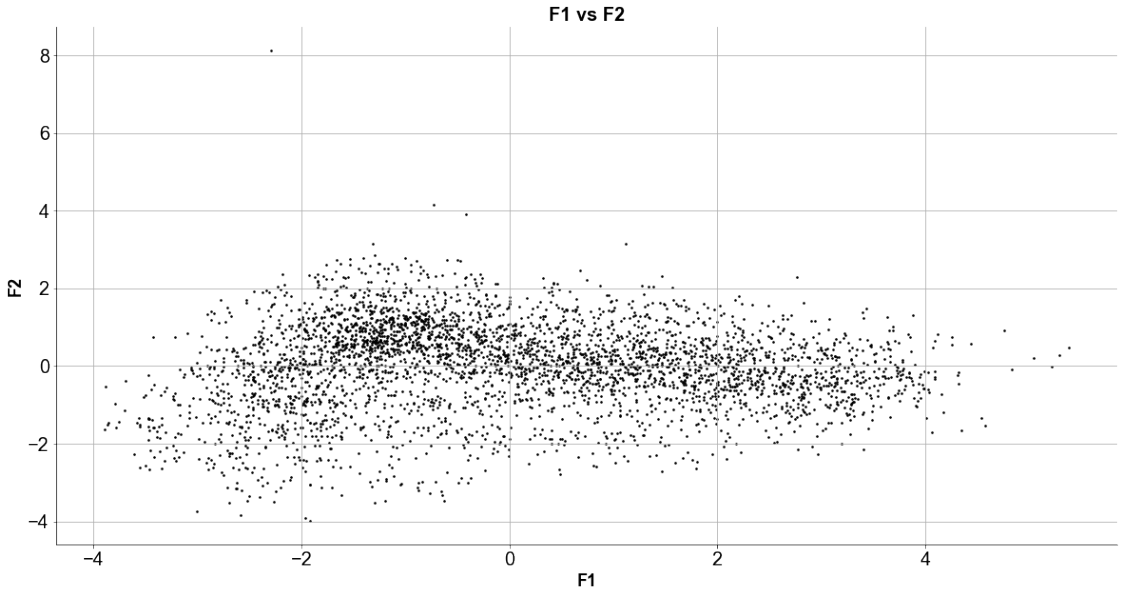
```
for row in range(0,len(s)):
    #ax.annotate('X'+str(row+1), (L[row,0]+0.01,L[row,1]+0.01))
    ax.annotate(header_row[row], (L[row,0]+random.uniform(-0.05,0.
 ↪05),L[row,1]+random.uniform(-0.05,0.05)),**fontL)

plt.ylabel('F2',**fontL)
plt.xlabel('F1',**fontL)

plt.xlim(-2,2)
plt.ylim(-2,2)

plt.xticks(fontsize = 20 , family = "Arial")
plt.yticks(fontsize = 20 , family = "Arial")

plt.title('Correlation Plot',**fontT)
fig.set_size_inches(30, 20)
fig.savefig('PCA_Correlation_Plot.jpg', dpi=300)
plt.show()
```



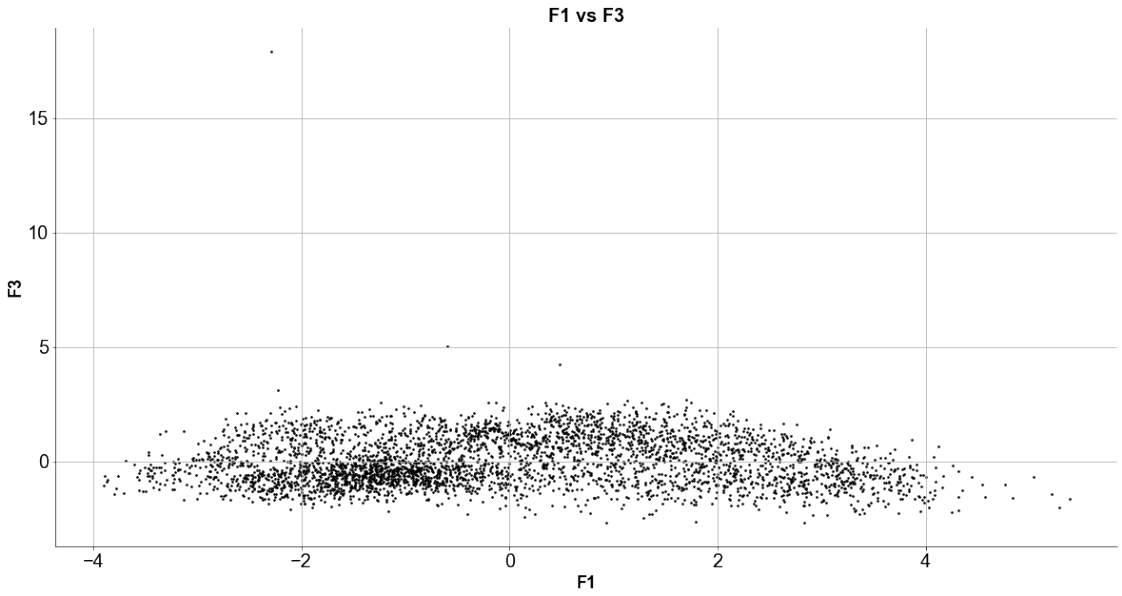
[ ]:
```
#Recuperate F just in case
F = pca.transform(Xdat)
F_df=pd.DataFrame(F[:,[0,1,2]],columns=["F1","F2","F3"])
```

[ ]:
```
## Plots for Factor Loadings  F1 VS F2 #
plt.figure(figsize=(20,10))

plt.scatter(x= F_df['F1'],y=F_df['F2'], marker=".", c = "k" , s = 10)
plt.title("F1 vs F2",**fontT)
plt.xlabel("F1",**fontL)
plt.ylabel("F2",**fontL)
plt.xticks(fontsize = 20 , family = "Arial")
plt.yticks(fontsize = 20 , family = "Arial")
```

```
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.show()
```



[ ]:
```
## Plots for Factor Loadings  F1 VS F2 #
plt.figure(figsize=(20,10))

plt.scatter(x= F_df['F1'],y=F_df['F3'], marker=".", c = "k" , s = 10)
plt.title("F1 vs F3",**fontT)
plt.xlabel("F1",**fontL)
plt.ylabel("F3",**fontL)
plt.xticks(fontsize = 20 , family = "Arial")
plt.yticks(fontsize = 20 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.show()
```



1.6 PCA 2.5

[ ]:
```
set_printoptions(precision=6,suppress=True)
```

[ ]:
```
#Only O3 Dataframe
PM25_df = df.drop(["PM10","O3"],axis=1)
header_row=PM25_df.columns.values

y= PM25_df.values
```

[ ]:
```
M = mean(PM25_df.T, axis=1)
print(M)

#Std Deviation
st = std(PM25_df.T, axis=1,ddof=1)

#Normalize data
Xdat = (PM25_df - M)/st

print(st)
print(Xdat)
```

```
PM2.5       27.675823
DayWeek      4.075926
Hour        12.037834
PRS        719.629956
RAINF        0.002651
RH          71.048458
```

```
SR           0.182582
TOUT        22.018969
WSR          7.331459
WDV        127.522156
CONT         0.182172
dtype: float64
PM2.5      16.849434
DayWeek     1.973040
Hour        6.869488
PRS         3.695949
RAINF       0.042447
RH         18.334615
SR          0.261602
TOUT        7.545535
WSR         3.828216
WDV        96.515310
CONT        0.386036
dtype: float64
         PM2.5   DayWeek      Hour       PRS     RAINF        RH        SR  \
0     3.105397  0.468350 -1.606791  1.561181  0.173134  1.360898 -0.697939
1     0.078589  0.468350 -1.461220  1.452954  0.408722  1.360898 -0.697939
2    -1.108395  0.468350 -1.315649  1.290614 -0.062453  1.360898 -0.697939
3    -0.811649  0.468350 -1.024506  1.182388  0.173134  1.360898 -0.697939
4    -1.167744  0.468350 -0.587793  1.263558 -0.062453  1.360898 -0.610019
…         …         …         …         …         …         …
3854 -1.108395 -1.052146  1.304634  0.749481 -0.062453  1.033648 -0.697939
3855 -0.574252 -1.052146  1.450205  0.695368 -0.062453  1.033648 -0.697939
3856 -0.692950 -1.052146  1.595776  0.695368 -0.062453  0.979107 -0.697939
3857 -0.396205 -0.545314 -1.752363  0.587141 -0.062453  0.979107 -0.697939
3858 -0.514903 -0.545314 -1.606791  0.505971 -0.062453  0.979107 -0.697939

          TOUT       WSR       WDV      CONT
0    -2.399958 -0.034339  2.242938  2.118530
1    -2.413211 -0.269436  2.211855 -0.471903
2    -2.407910 -0.556776  2.263660 -0.471903
3    -2.415862 -0.452289  2.160049 -0.471903
4    -2.397308 -0.661263  2.346548 -0.471903
…         …         …         …         …
3854 -0.630700 -0.034339 -0.399130 -0.471903
3855 -0.638652 -0.347801 -0.181548 -0.471903
3856 -0.642628 -0.478410 -0.181548 -0.471903
3857 -0.638652 -1.183700 -0.026132 -0.471903
3858 -0.625399 -1.653893 -0.326603 -0.471903

[3859 rows x 11 columns]
```

[ ]:
```
# create the PCA instance
pca = PCA()

# fit on data
pca = pca.fit(Xdat)

#Q eigenvectors
Q=pca.components_.T

#Explained deviations
s=sqrt(pca.explained_variance_)
```

[ ]:
```
# transform data obtain sample scores
F = pca.transform(Xdat)
print("Sample scores")

## Sample Scores ##
sample_scores = pd.DataFrame(F)
print(sample_scores)
```

```
Sample scores
            0         1         2         3         4         5         6   \
0    -3.332104  3.364667  0.788140 -1.419338 -0.512252  1.279109 -0.863443
1    -3.619532 -0.365693  0.406407 -0.699038  0.080974  1.092205 -1.073600
2    -3.700960 -1.136480  0.182067 -0.788381 -0.143029  0.800074 -1.204013
3    -3.511648 -0.900275  0.409144 -0.667464  0.060791  0.795374 -1.170142
4    -3.507265 -1.146002  0.575577 -0.918854 -0.048656  0.713199 -1.240448
…          …         …         …         …         …         …         …
3854 -0.870761 -0.978201  0.923340 -0.182787  0.875026 -1.397590  0.634430
3855 -0.952444 -0.566065  0.917680 -0.371798  0.881607 -1.301008  0.551943
3856 -0.948200 -0.632679  0.970539 -0.400835  0.895292 -1.307883  0.582909
3857 -2.200881 -0.422381 -1.291091  0.114737  0.101790 -0.186136  0.509124
3858 -2.262441 -0.413799 -1.398174  0.274009  0.063491 -0.279091  0.791066

            7         8         9         10
0     1.928408  0.181333  0.929406  0.460308
1     1.106379 -0.128359  0.540439  0.464637
2     0.716708 -0.314201 -0.332764  0.407086
3     0.650608 -0.416919 -0.137595  0.497708
4     0.280780 -0.683953 -0.389837  0.358207
…          …         …         …         …
3854 -0.239478 -0.929078 -0.398787 -0.371058
3855 -0.542952 -0.974821 -0.034864 -0.372315
3856 -0.732780 -1.004811 -0.129643 -0.383550
3857  0.257133  0.459435  0.106279 -0.238429
3858 -0.198690  0.420477 -0.010833 -0.248693

[3859 rows x 11 columns]
```

[ ]:
```
# Obtain Factor Loadings
L = zeros(shape=(Q.shape))
for col in range(0,len(s)-1):
    L[:,col] = Q[:,col] * s[col]
print("Factor Loadings")

columms_factorLoading = []

for i in range(0,11):
    nombre =  "F" + str(i)
    columms_factorLoading.append(nombre)


## Factor Loadings Dataframe ##
factor_loadings = pd.DataFrame(L, columns= columms_factorLoading)
print(factor_loadings)
```

```
Factor Loadings
          F0        F1        F2        F3        F4        F5        F6  \
0   0.098614  0.926575  0.060368 -0.094054 -0.061560  0.028244  0.024809
1  -0.074971  0.030014  0.314278  0.575360 -0.557906  0.338068 -0.357507
2   0.482353  0.044370  0.694308 -0.110425  0.142707 -0.233560 -0.025009
3  -0.563383 -0.242595  0.519107 -0.259623 -0.191178  0.170124  0.376025
4  -0.054471  0.092364  0.179540  0.392990  0.707622  0.533241  0.124385
5  -0.733892  0.183714 -0.161754  0.321727  0.075702 -0.218342 -0.087796
6   0.569974 -0.139360 -0.216871 -0.170308 -0.269924  0.526111  0.294819
7   0.880499  0.032747 -0.293039  0.108814  0.042523  0.011266 -0.099710
8   0.673572 -0.176396  0.355735  0.054880  0.077865 -0.143523 -0.229303
9  -0.327247 -0.040384 -0.010095 -0.576022  0.151915  0.358257 -0.634157
10  0.053451  0.914040  0.092564 -0.136195 -0.084604  0.074699  0.068934

          F7        F8        F9  F10
0   0.033075  0.043552  0.332060  0.0
1  -0.093435  0.039166 -0.006334  0.0
2  -0.342497 -0.278712 -0.009025  0.0
3   0.158736  0.099742  0.050671  0.0
4   0.005035  0.035116  0.000765  0.0
5   0.167103 -0.436425  0.025765  0.0
6   0.061686 -0.369003  0.019443  0.0
7  -0.054579  0.084724  0.019188  0.0
8   0.550589 -0.019955  0.015950  0.0
9  -0.010270 -0.059179  0.008795  0.0
10  0.111362  0.012974 -0.322031  0.0
```

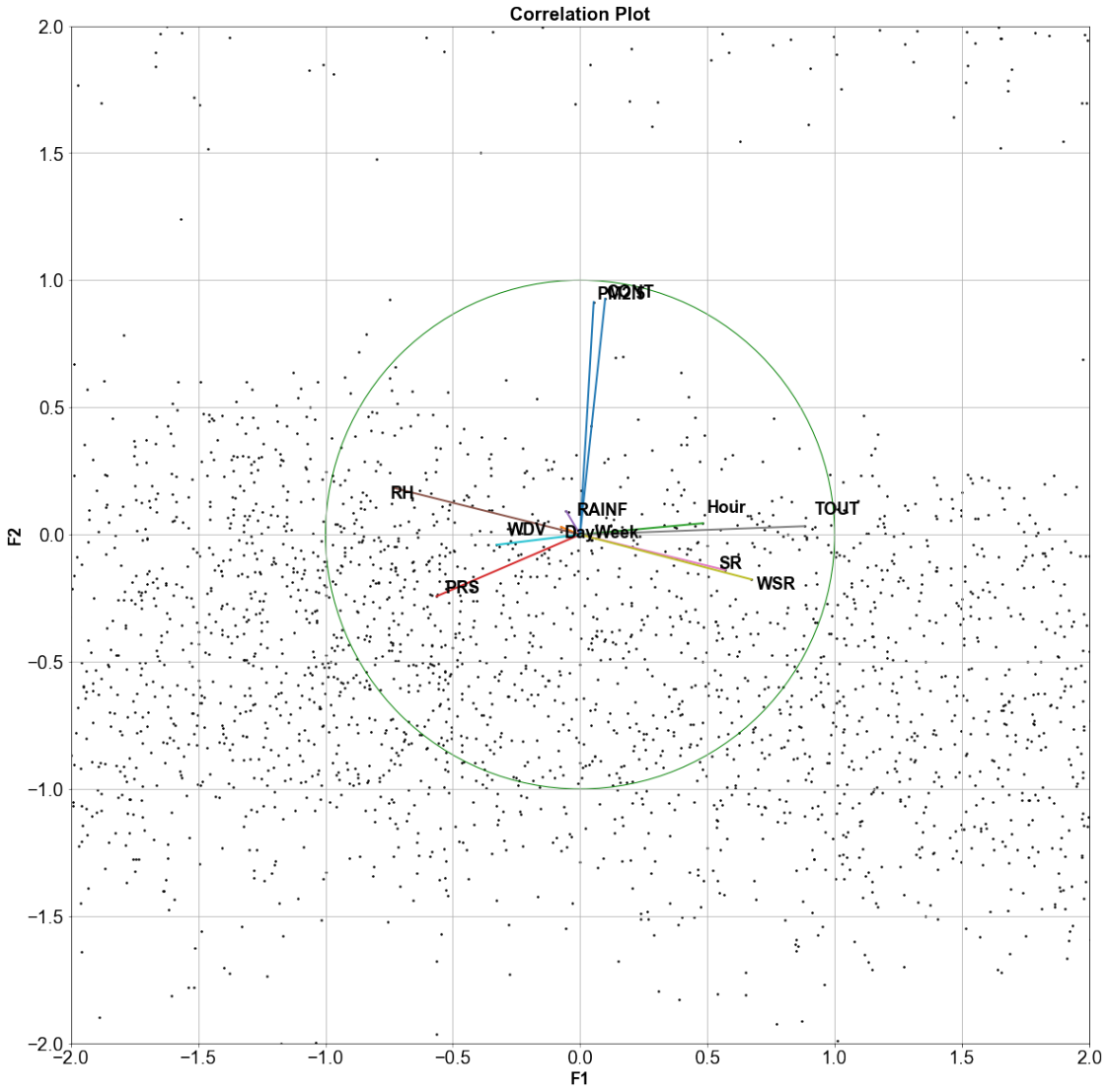
[ ]:
```
#Obtain squared cosines
COS2=L**2
print("Square Cosines")
```

```
## COS2  Dataframe ##
COS2_DF = pd.DataFrame(COS2)
print(COS2_DF)
```

```
Square Cosines
          0         1         2         3         4         5         6   \
0   0.009725  0.858540  0.003644  0.008846  0.003790  0.000798  0.000615
1   0.005621  0.000901  0.098771  0.331039  0.311259  0.114290  0.127812
2   0.232664  0.001969  0.482064  0.012194  0.020365  0.054550  0.000625
3   0.317401  0.058852  0.269473  0.067404  0.036549  0.028942  0.141394
4   0.002967  0.008531  0.032234  0.154442  0.500729  0.284346  0.015472
5   0.538598  0.033751  0.026164  0.103508  0.005731  0.047673  0.007708
6   0.324870  0.019421  0.047033  0.029005  0.072859  0.276793  0.086918
7   0.775278  0.001072  0.085872  0.011841  0.001808  0.000127  0.009942
8   0.453699  0.031115  0.126547  0.003012  0.006063  0.020599  0.052580
9   0.107090  0.001631  0.000102  0.331801  0.023078  0.128348  0.402155
10  0.002857  0.835469  0.008568  0.018549  0.007158  0.005580  0.004752

          7         8             9    10
0   0.001094  0.001897  1.102636e-01  0.0
1   0.008730  0.001534  4.011926e-05  0.0
2   0.117305  0.077680  8.145238e-05  0.0
3   0.025197  0.009948  2.567527e-03  0.0
4   0.000025  0.001233  5.856692e-07  0.0
5   0.027923  0.190467  6.638242e-04  0.0
6   0.003805  0.136163  3.780443e-04  0.0
7   0.002979  0.007178  3.681789e-04  0.0
8   0.303148  0.000398  2.544168e-04  0.0
9   0.000105  0.003502  7.735630e-05  0.0
10  0.012402  0.000168  1.037038e-01  0.0
```

[ ]:
```
import random

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.grid()

circ=plt.Circle((0,0), radius=1, color='g', fill=False)
ax.add_patch(circ)
ax.set_aspect('equal')

plt.scatter(L[:,0],L[:,1],marker=".", c = "k" , s = 10)
plt.scatter(F[:,0],F[:,1],marker=".", c = "k" , s = 10)


for row in range(0,len(s)):
    plt.plot([0, L[row,0]], [0,L[row,1]], linewidth=2,label='X1')
```

```
for row in range(0,len(s)):
    #ax.annotate('X'+str(row+1), (L[row,0]+0.01,L[row,1]+0.01))
    ax.annotate(header_row[row], (L[row,0]+random.uniform(-0.05,0.
 ↪05),L[row,1]+random.uniform(-0.05,0.05)),**fontL)

plt.ylabel('F2',**fontL)
plt.xlabel('F1',**fontL)

plt.xlim(-2,2)
plt.ylim(-2,2)

plt.xticks(fontsize = 20 , family = "Arial")
plt.yticks(fontsize = 20 , family = "Arial")

plt.title('Correlation Plot',**fontT)
fig.set_size_inches(30, 20)
fig.savefig('PCA_Correlation_Plot.jpg', dpi=300)
plt.show()
```



1.7 Regresion

1.8 Variable Regression

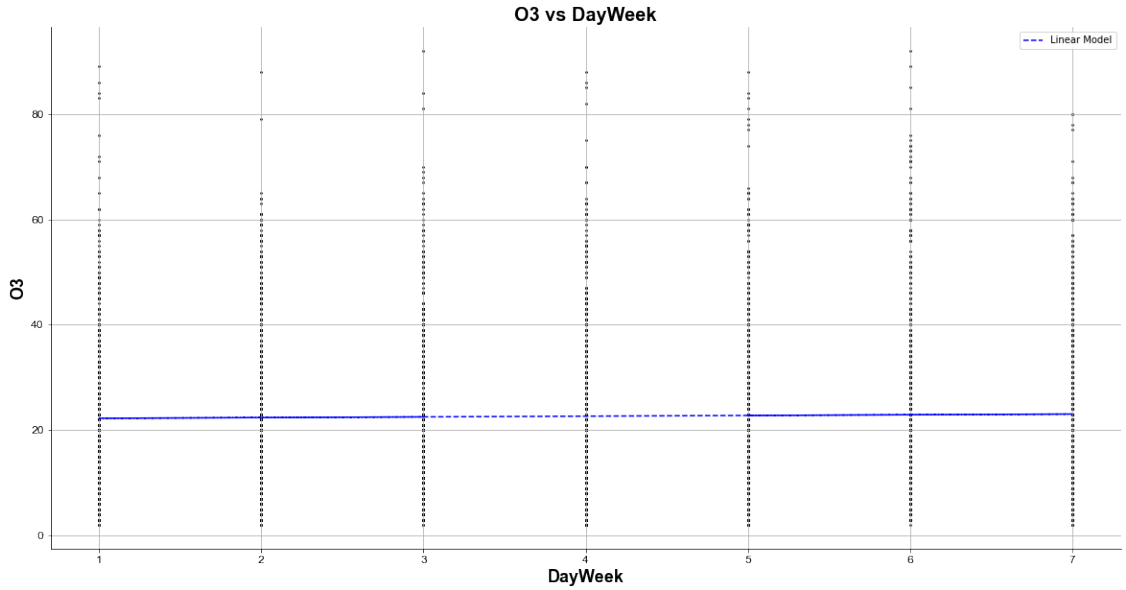
[ ]:
```
## We get back the data that we are analyzing ##
regression_columnNames = column_names.drop(['CONT'])

## Predicting Variable ##
x  = df[regression_columnNames]

## Y model of O3 ##
y = O3.values
```

[ ]:
```
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
from sklearn.metrics import r2_score

def regression_results(y_true, y_pred):
    explained_variance=metrics.explained_variance_score(y_true, y_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    median_absolute_error=metrics.median_absolute_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)

    print('explained_variance: ', round(explained_variance,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

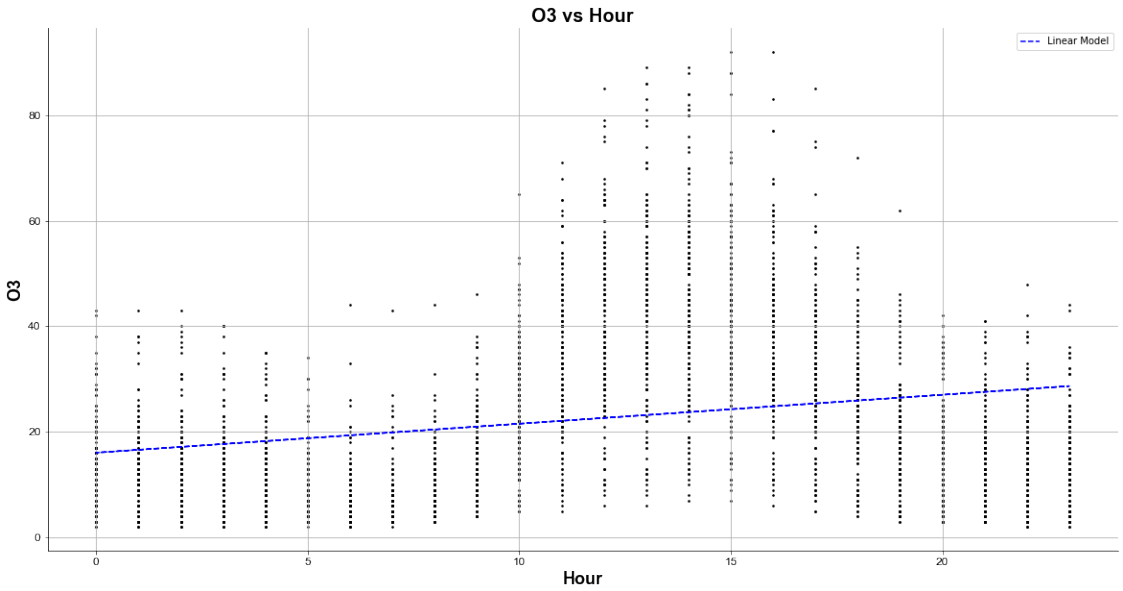
[ ]:
```
for i in range(0,len(regression_columnNames)):
    nombre = regression_columnNames[i]
    y = O3.values
    x = df[nombre].values

    # Reshape Values#
    x = x.reshape(-1,1)
    x.shape

    # Reshape Values#
    y = y.reshape(-1,1)
    y.shape

    #Linear model#
    lm = LinearRegression().fit(x,y)
    lmp = lm.predict(x)
    regression_results(y,lmp)

    ## Plots for O3 vs Hour#
    plt.figure(figsize=(20,10))
    plt.scatter(x,y, marker=".", c = "k" , s = 10)
    plt.plot(x, lmp, 'b--')
    plt.title("O3 vs " + nombre ,**fontT)
    plt.xlabel(nombre,**fontL)
    plt.ylabel("O3",**fontL)
    plt.xticks(fontsize = 12 , family = "Arial")
    plt.yticks(fontsize = 12 , family = "Arial")
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.grid("On")
```

```
    plt.legend(['Linear Model'])
    plt.show()

    regression_results(y,lmp)
```

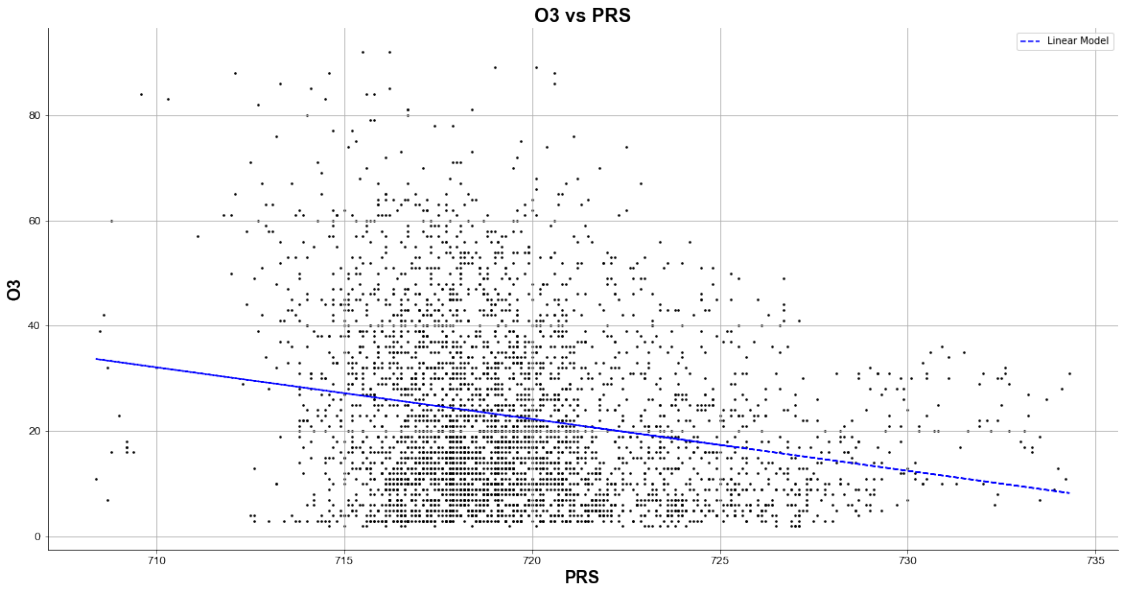
```
explained_variance:  0.0003
r2:  0.0003
MAE:  13.2458
MSE:  272.4583
RMSE:  16.5063
```



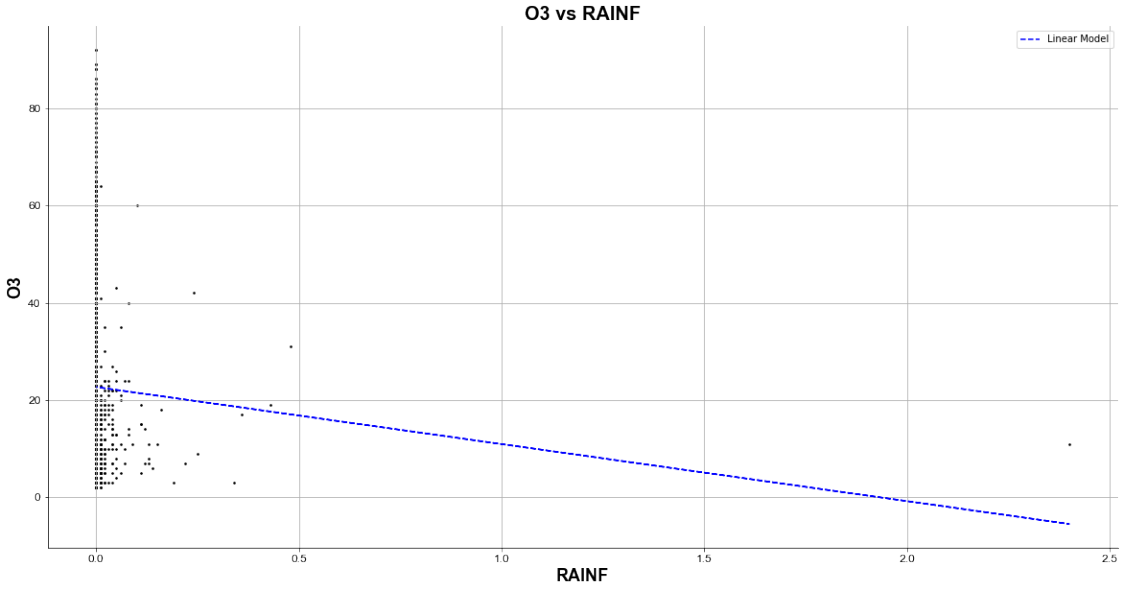

```
explained_variance:  0.0003
r2:  0.0003
MAE:  13.2458
MSE:  272.4583
RMSE:  16.5063
explained_variance:  0.0522
r2:  0.0522
MAE:  12.7226
MSE:  258.3135
RMSE:  16.0721
```



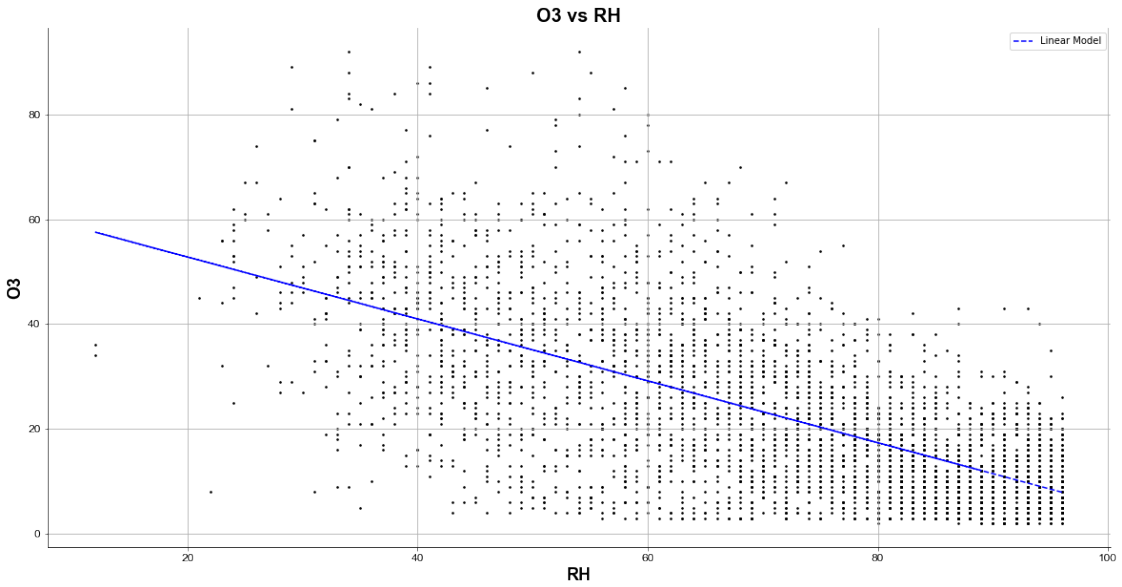

```
explained_variance:  0.0522
r2:  0.0522
MAE:  12.7226
MSE:  258.3135
RMSE:  16.0721
explained_variance:  0.0483
r2:  0.0483
MAE:  12.9606
MSE:  259.3697
RMSE:  16.105
```



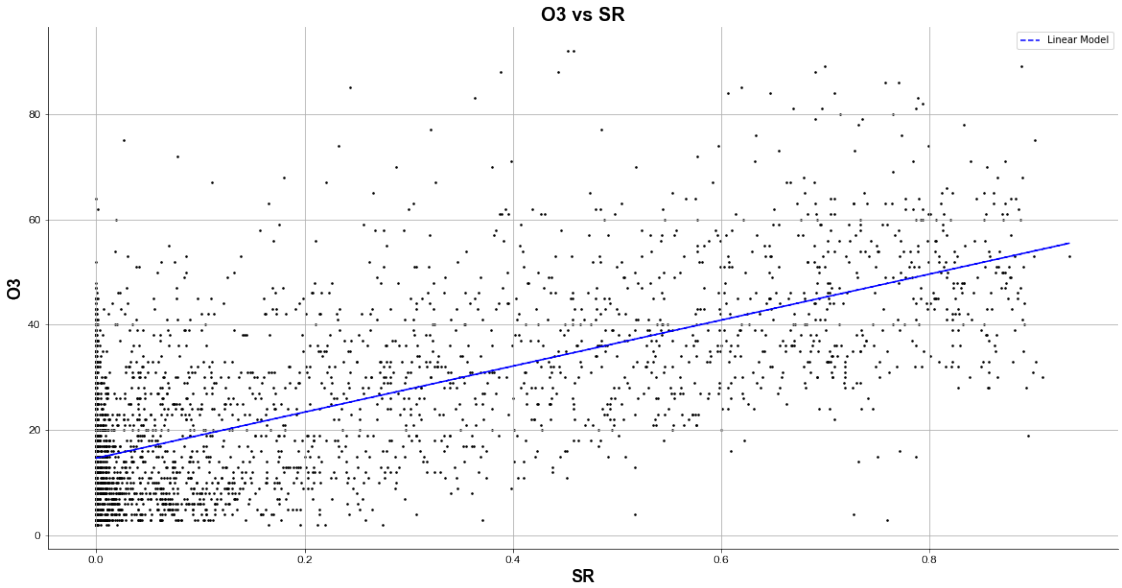
```
explained_variance:  0.0483
r2:  0.0483
MAE:  12.9606
MSE:  259.3697
RMSE:  16.105
explained_variance:  0.0009
r2:  0.0009
MAE:  13.2423
MSE:  272.2804
RMSE:  16.5009
```



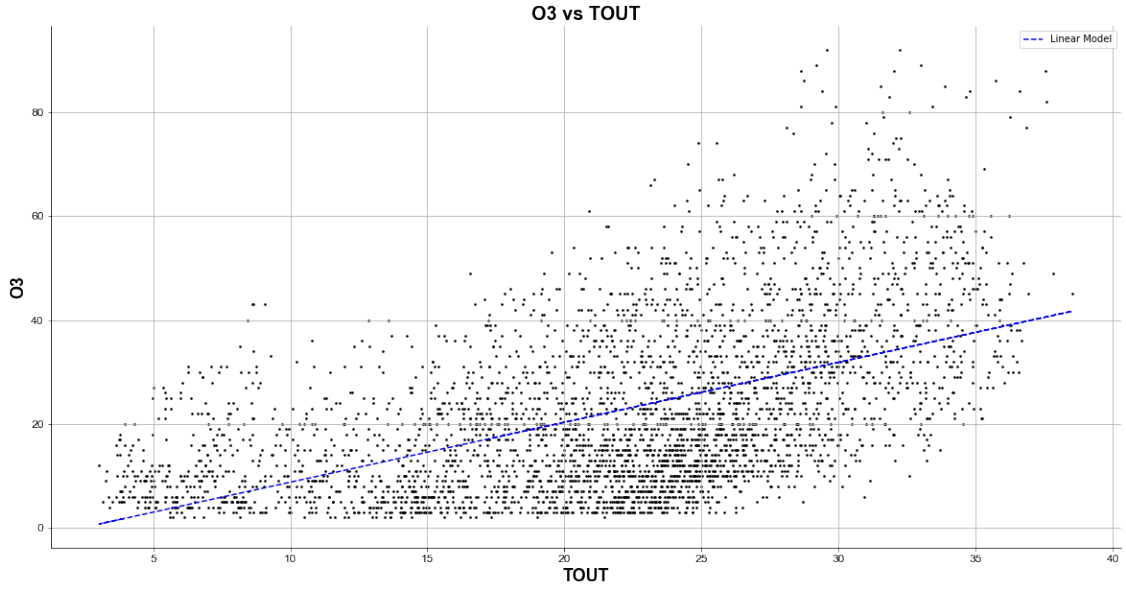

```
explained_variance:  0.0009
r2:  0.0009
MAE:  13.2423
MSE:  272.2804
RMSE:  16.5009
explained_variance:  0.4298
r2:  0.4298
MAE:  9.3624
MSE:  155.3966
RMSE:  12.4658
```



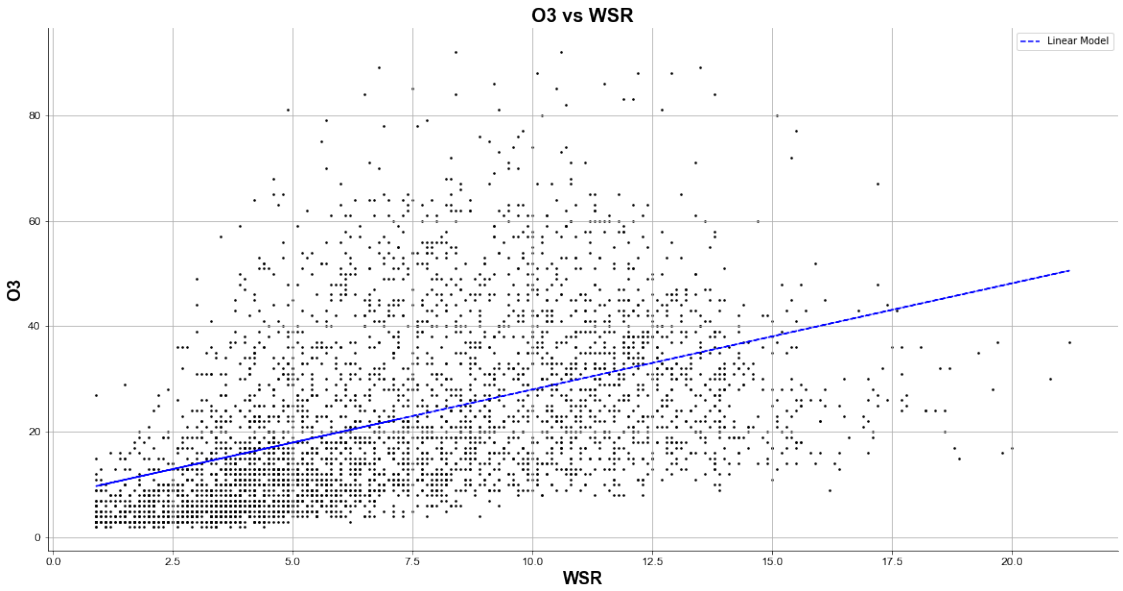

```
explained_variance:  0.4298
r2:  0.4298
MAE:  9.3624
MSE:  155.3966
RMSE:  12.4658
explained_variance:  0.4788
r2:  0.4788
MAE:  9.2337
MSE:  142.0518
RMSE:  11.9185
```



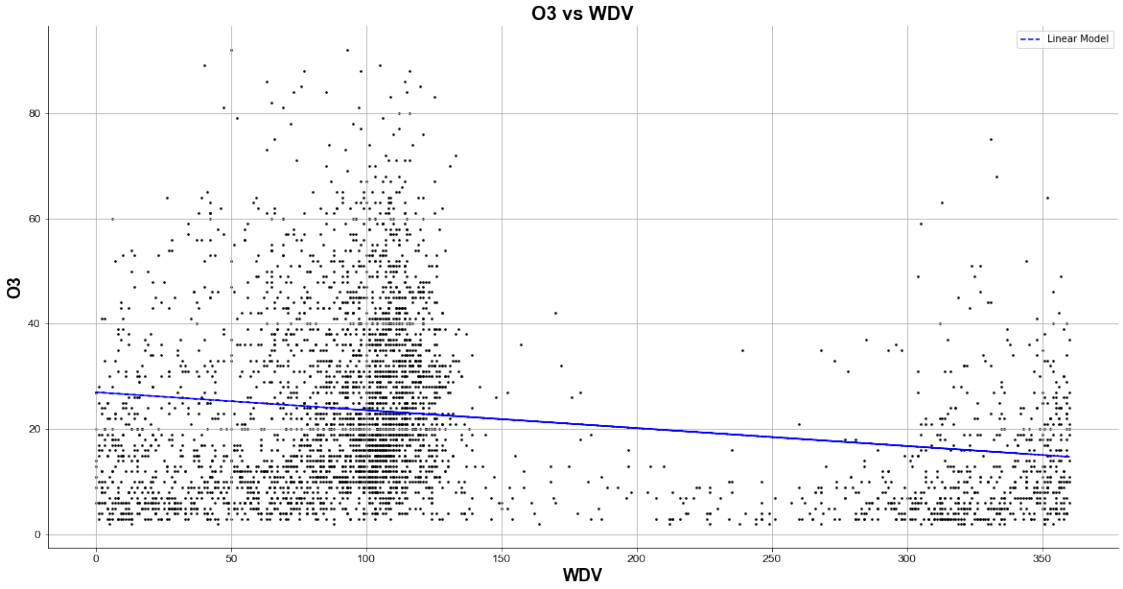
```
explained_variance:  0.4788
r2:  0.4788
MAE:  9.2337
MSE:  142.0518
RMSE:  11.9185
explained_variance:  0.2779
r2:  0.2779
MAE:  11.2832
MSE:  196.7947
RMSE:  14.0284
```




```
explained_variance:  0.2779
r2:  0.2779
MAE:  11.2832
MSE:  196.7947
RMSE:  14.0284
explained_variance:  0.2174
r2:  0.2174
MAE:  11.1855
MSE:  213.2824
RMSE:  14.6042
```




```
explained_variance:  0.2174
r2:  0.2174
MAE:  11.1855
MSE:  213.2824
RMSE:  14.6042
explained_variance:  0.0395
r2:  0.0395
MAE:  12.9517
MSE:  261.7575
RMSE:  16.1789
```



```
explained_variance:  0.0395
r2:  0.0395
MAE:  12.9517
MSE:  261.7575
RMSE:  16.1789
```

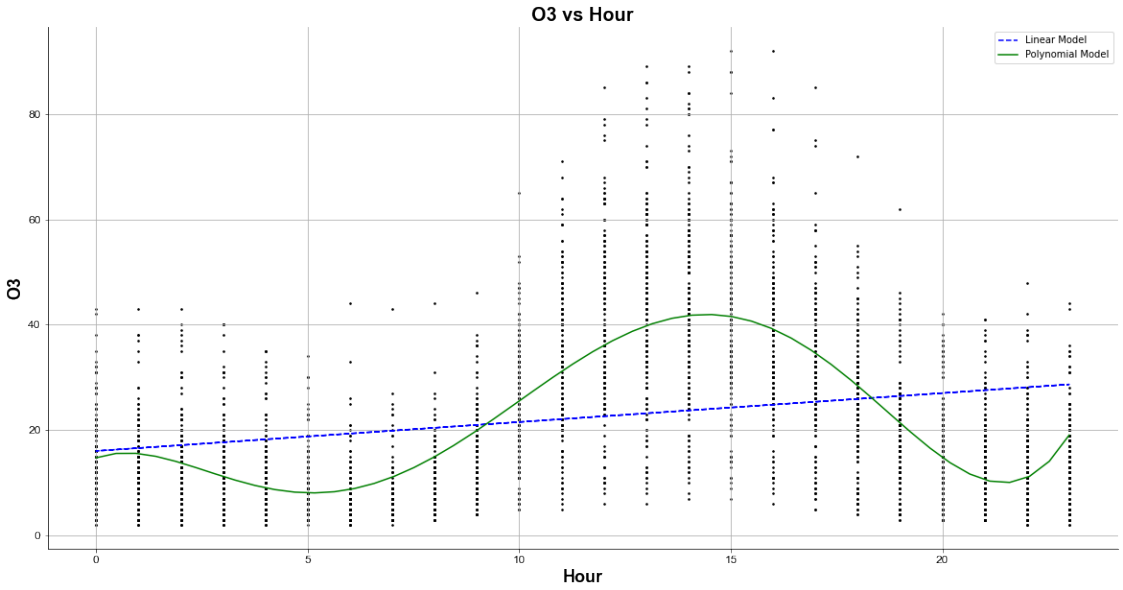
1.9 Polynomial Regression

1.10 O3 vs Hour

[ ]:
```
nombre = "Hour"

deg = 5
y = O3.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
```

```
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0522
r2:  0.0522
MAE:  12.7226
MSE:  258.3135
RMSE:  16.0721
0.5005209891262465
explained_variance:  0.5005
r2:  0.5005
MAE:  8.7186
MSE:  136.122
RMSE:  11.6671
```



1.11 O3 vs PRS

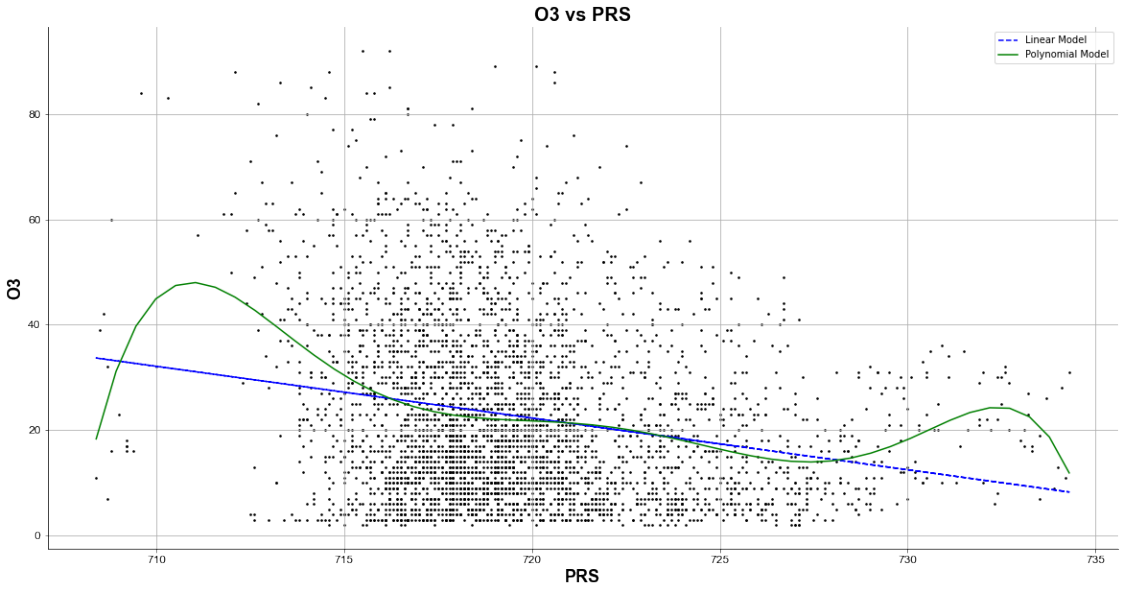
[ ]:
```
nombre = "PRS"

deg = 20
y = O3.values
x = df[nombre].values
```

```
# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0483
r2:  0.0483
MAE:  12.9606
MSE:  259.3697
RMSE:  16.105
0.07205411530220662
explained_variance:  0.0721
```

```
r2:  0.0721
MAE:  12.6928
MSE:  252.8912
RMSE:  15.9026
```

```
C:\Python39\lib\site-packages\IPython\core\interactiveshell.py:3441:
RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```

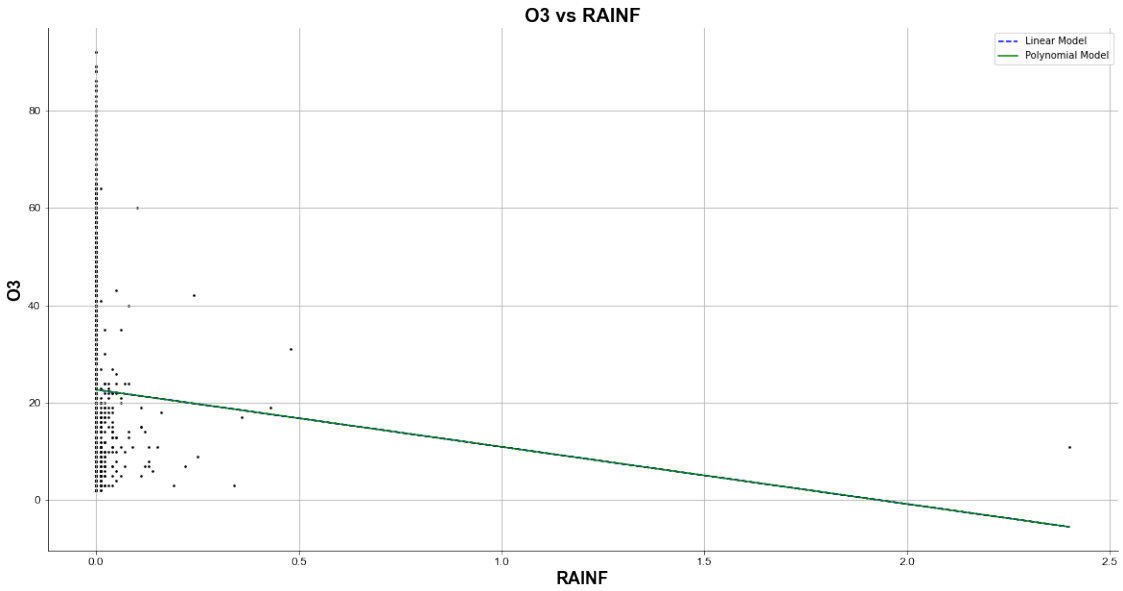


1.12 O3 vs RAINF

[ ]:
```
nombre = "RAINF"

deg = 1
y = O3.values
x = df[nombre].values



# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
```

```
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0009
r2:  0.0009
MAE:  13.2423
MSE:  272.2804
RMSE:  16.5009
0.0009085669235929794
explained_variance:  0.0009
r2:  0.0009
MAE:  13.2423
MSE:  272.2804
RMSE:  16.5009
```

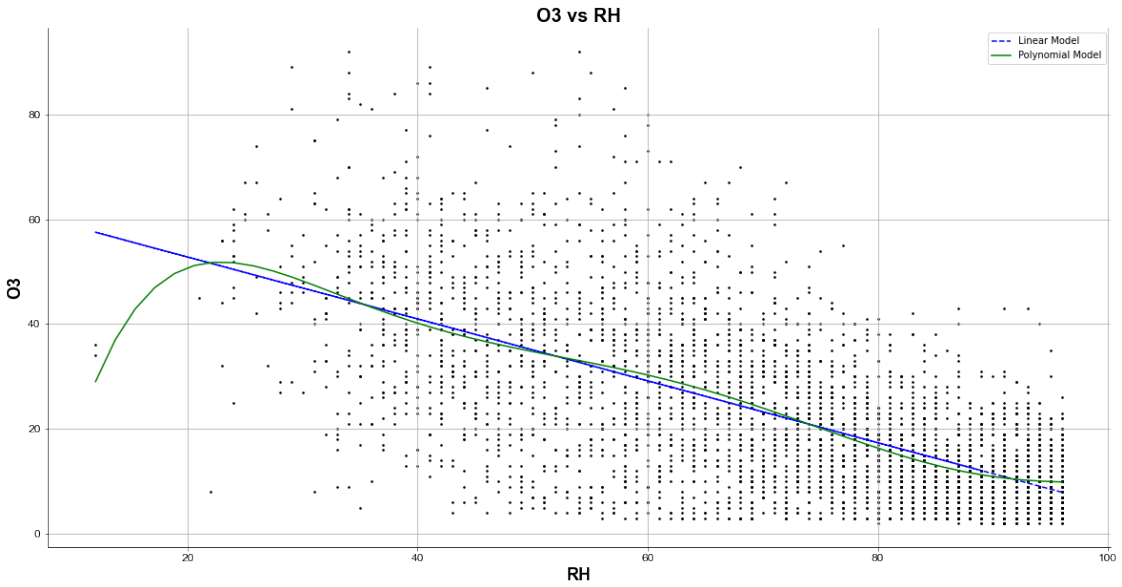


1.13 O3 vs RH

[ ]:
```
nombre = "RH"

deg = 6
y = O3.values
x = df[nombre].values



# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
```

```
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.4298
r2:  0.4298
MAE:  9.3624
MSE:  155.3966
RMSE:  12.4658
0.43466429562493647
explained_variance:  0.4347
r2:  0.4347
MAE:  9.2991
MSE:  154.0698
RMSE:  12.4125
```

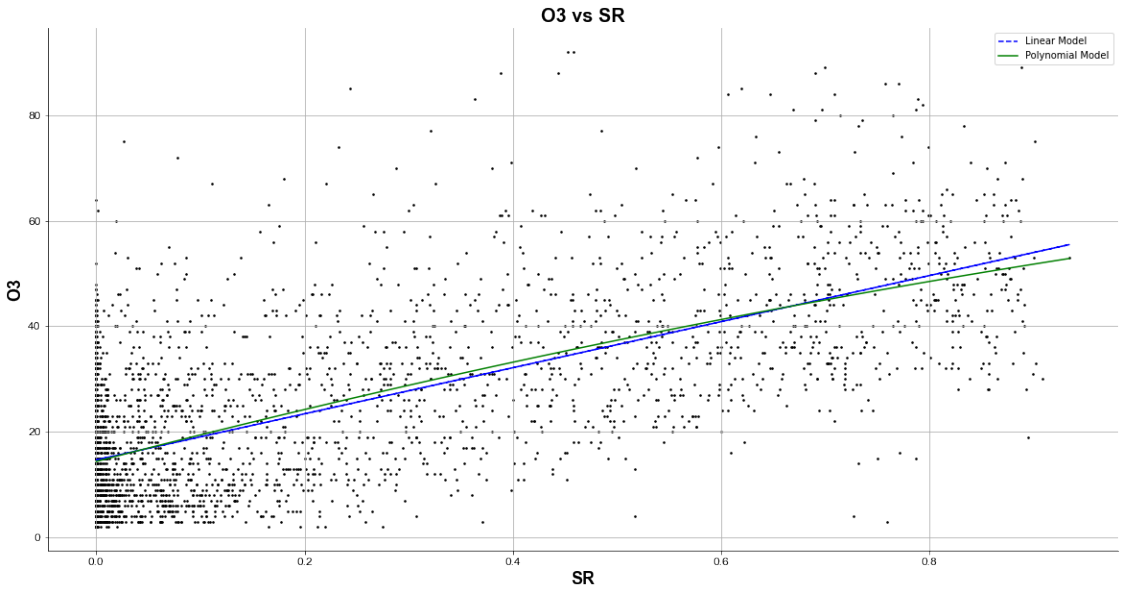


1.14 O3 vs SR

[ ]:
```
nombre = "SR"

deg = 2
y = O3.values
x = df[nombre].values



# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
```

```
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.4788
r2:  0.4788
MAE:  9.2337
MSE:  142.0518
RMSE:  11.9185
0.4800531633289926
explained_variance:  0.4801
r2:  0.4801
MAE:  9.2194
MSE:  141.7001
RMSE:  11.9038
```

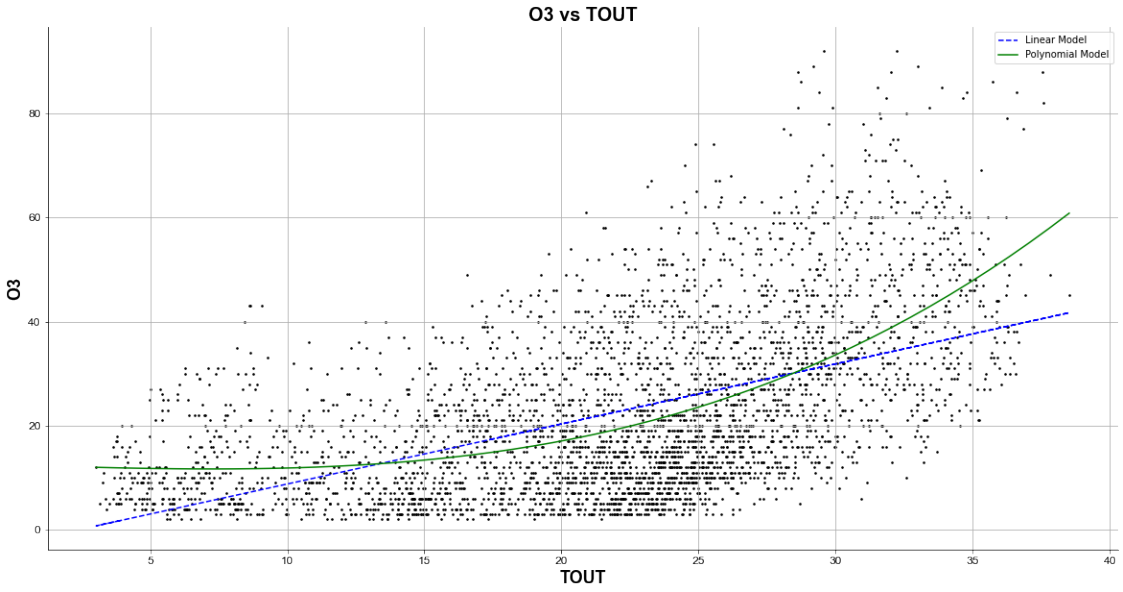


1.15 O3 vs TOUT

[ ]:
```
nombre = "TOUT"

deg = 3
y = O3.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = O3
x = df[nombre]
```

```
mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.2779
r2:  0.2779
MAE:  11.2832
MSE:  196.7947
RMSE:  14.0284
0.3370828596710743
explained_variance:  0.3371
r2:  0.3371
MAE:  10.676
MSE:  180.6635
RMSE:  13.4411
```

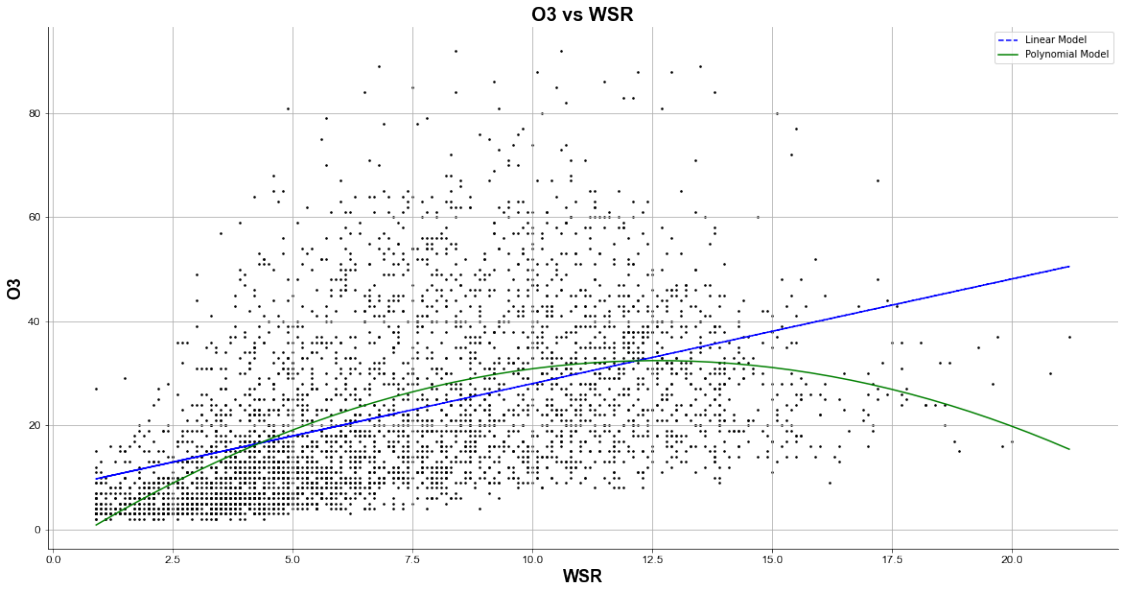


1.16 O3 vs WSR

[ ]:
```
nombre = "WSR"

deg = 2
y = O3.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])
```

```
regression_results(y,predicted)
```

```
explained_variance:  0.2174
r2:  0.2174
MAE:  11.1855
MSE:  213.2824
RMSE:  14.6042
0.2708880021393766
explained_variance:  0.2709
r2:  0.2709
MAE:  10.7847
MSE:  198.7034
RMSE:  14.0962
```

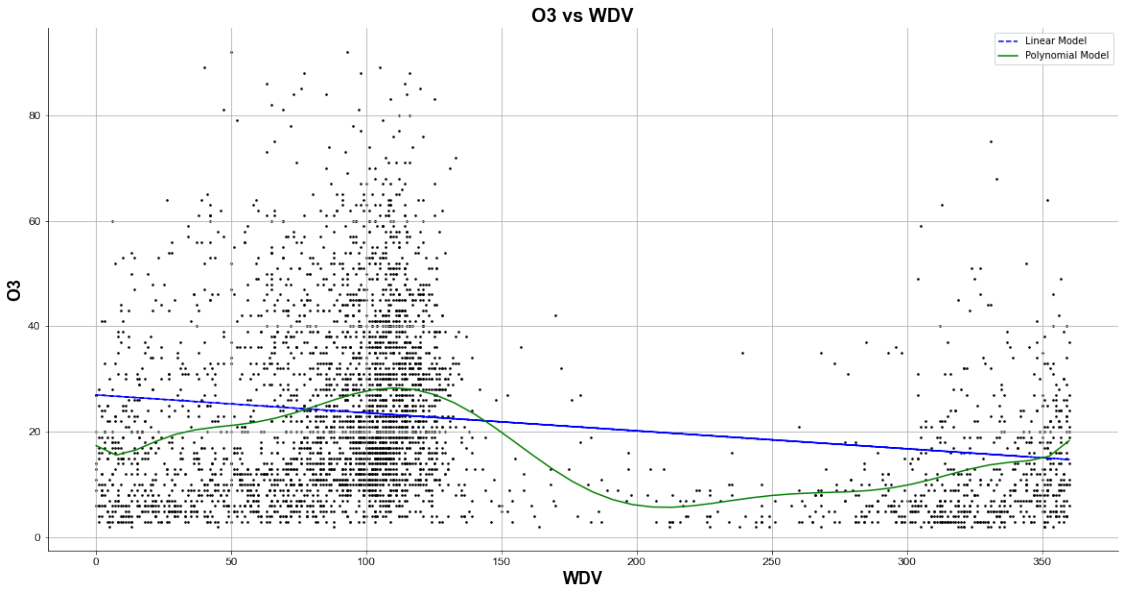


1.17 O3 vs WSR

[ ]:
```
nombre = "WDV"

deg = 10
y = O3.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
```

```
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for O3 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = O3
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("O3 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("O3",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0395
r2:  0.0395
MAE:  12.9517
MSE:  261.7575
RMSE:  16.1789
0.1342854005706544
explained_variance:  0.1343
r2:  0.1343
MAE:  12.0688
MSE:  235.9315
RMSE:  15.3601
```

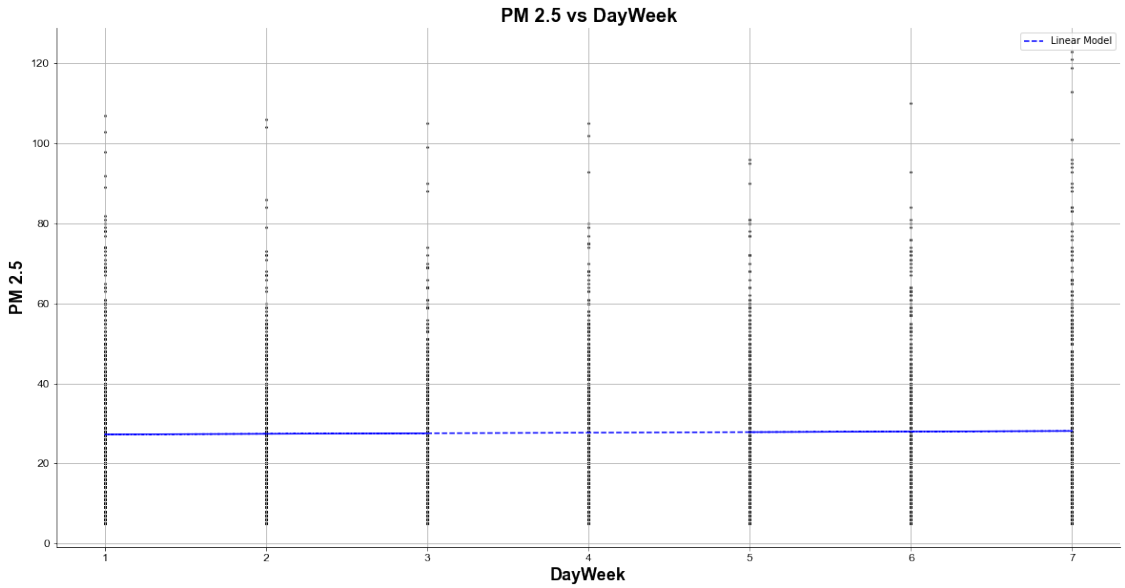


1.18 PM 2.5

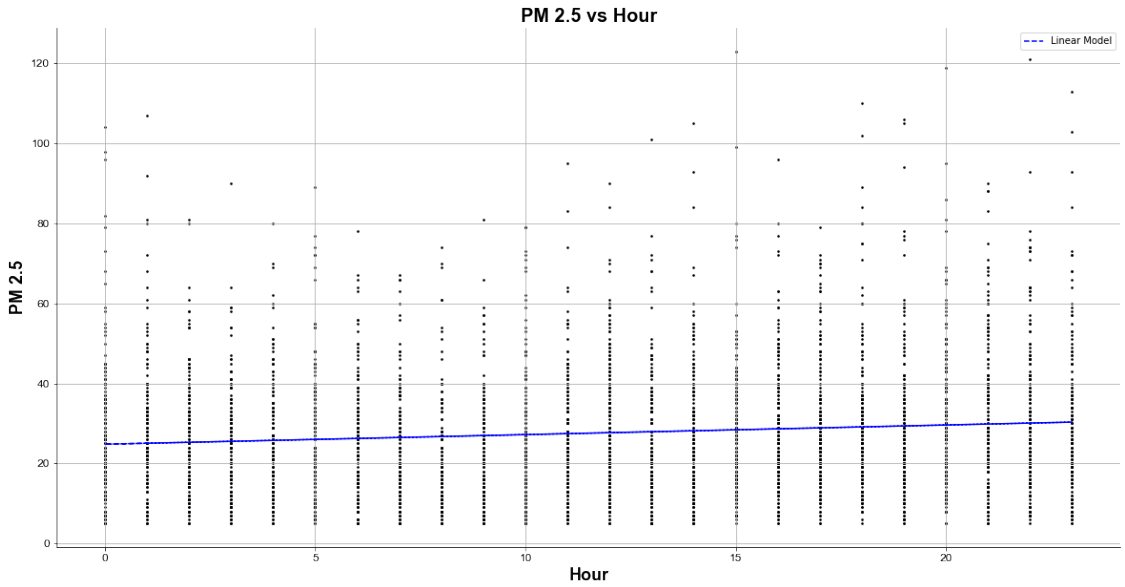
[ ]:
```
for i in range(0,len(regression_columnNames)):
    nombre = regression_columnNames[i]
    y = PM_25.values
    x = df[nombre].values

    # Reshape Values#
    x = x.reshape(-1,1)
    x.shape

    # Reshape Values#
    y = y.reshape(-1,1)
    y.shape

    #Linear model#
    lm = LinearRegression().fit(x,y)
    lmp = lm.predict(x)
    regression_results(y,lmp)

    ## Plots for O3 vs Hour"
    plt.figure(figsize=(20,10))
    plt.scatter(x,y, marker=".", c = "k" , s = 10)
    plt.plot(x, lmp, 'b--')
    plt.title("PM 2.5 vs " + nombre ,**fontT)
    plt.xlabel(nombre,**fontL)
    plt.ylabel("PM 2.5",**fontL)
```

```
    plt.xticks(fontsize = 12 , family = "Arial")
    plt.yticks(fontsize = 12 , family = "Arial")
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.grid("On")
    plt.legend(['Linear Model'])
    plt.show()

    regression_results(y,lmp)
```

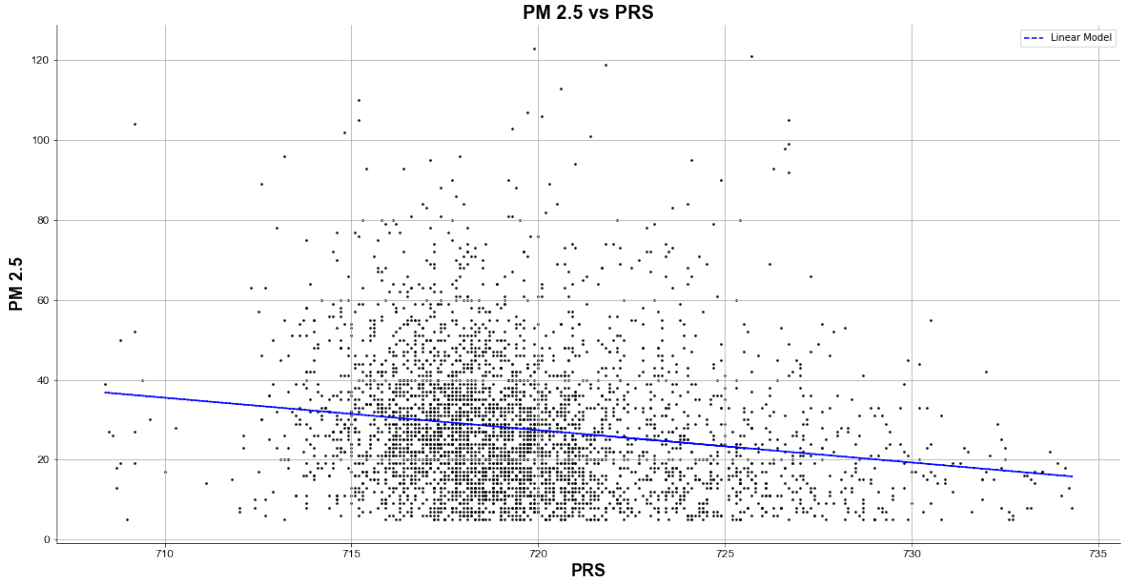
```
explained_variance:  0.0003
r2:  0.0003
MAE:  12.8635
MSE:  283.7502
RMSE:  16.8449
```



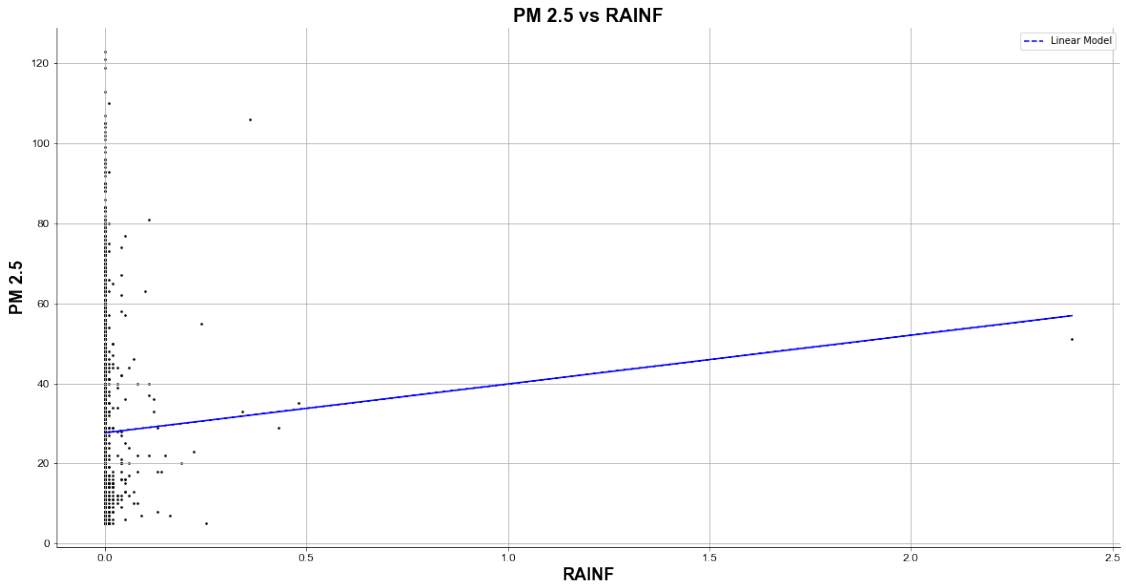

```
explained_variance:  0.0003
r2:  0.0003
MAE:  12.8635
MSE:  283.7502
RMSE:  16.8449
explained_variance:  0.0096
r2:  0.0096
MAE:  12.8232
MSE:  281.1104
RMSE:  16.7663
```



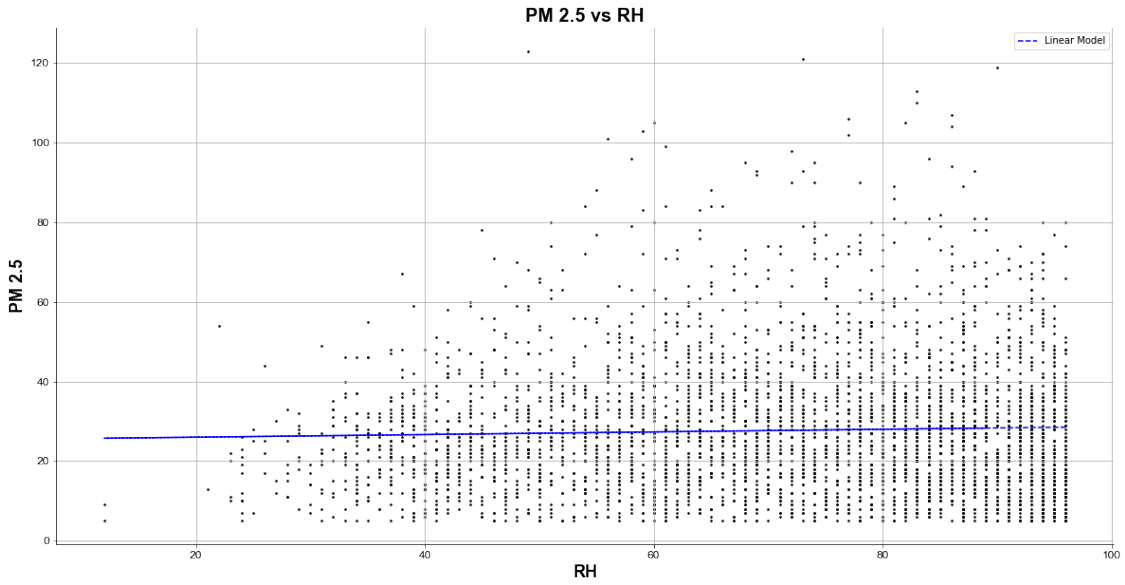

```
explained_variance:  0.0096
r2:  0.0096
MAE:  12.8232
MSE:  281.1104
RMSE:  16.7663
explained_variance:  0.0317
r2:  0.0317
MAE:  12.5394
MSE:  274.834
RMSE:  16.5781
```



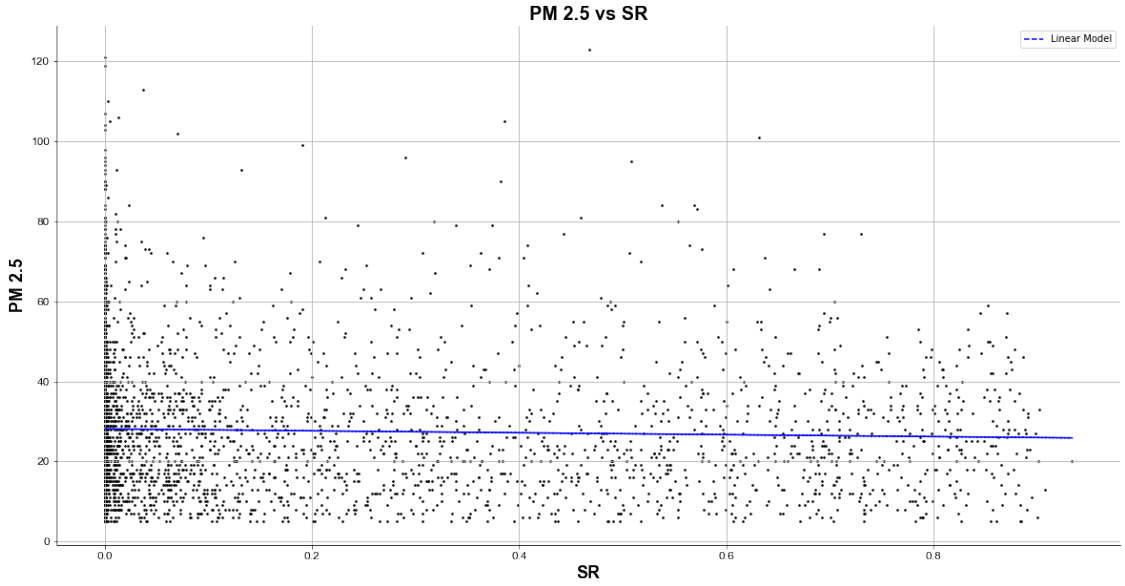
```
explained_variance:  0.0317
r2:  0.0317
MAE:  12.5394
MSE:  274.834
RMSE:  16.5781
explained_variance:  0.0009
r2:  0.0009
MAE:  12.8564
MSE:  283.5616
RMSE:  16.8393
```



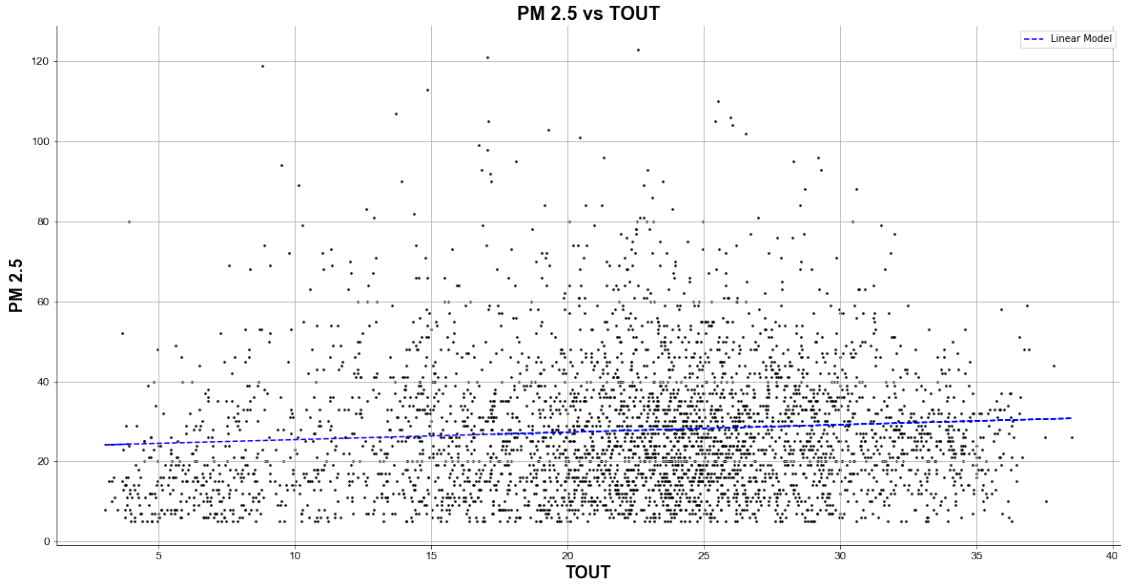

```
explained_variance:  0.0009
r2:  0.0009
MAE:  12.8564
MSE:  283.5616
RMSE:  16.8393
explained_variance:  0.0013
r2:  0.0013
MAE:  12.8484
MSE:  283.4589
RMSE:  16.8362
```



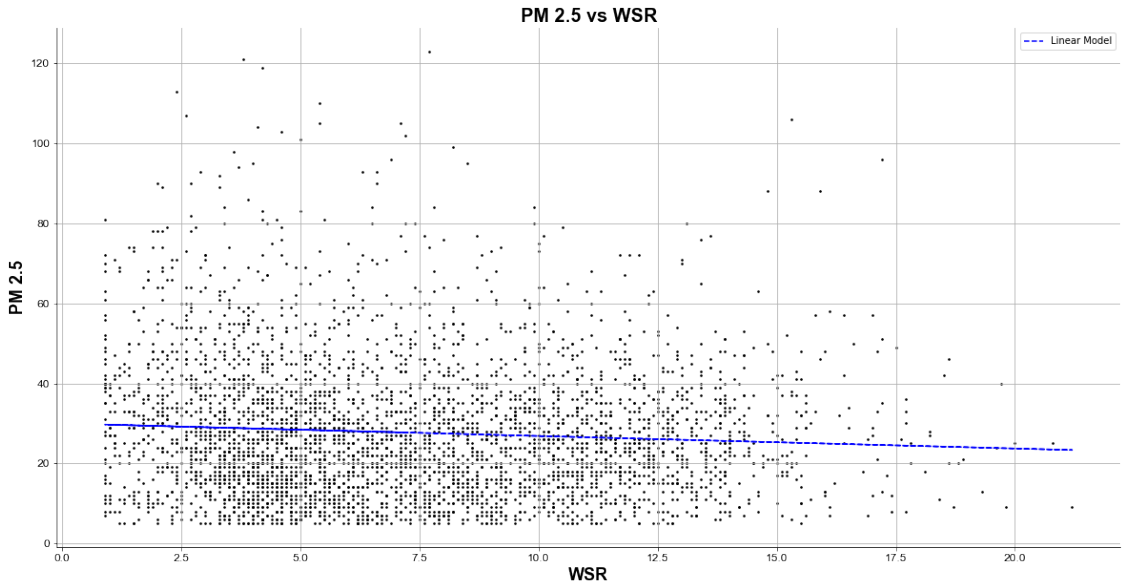

```
explained_variance:  0.0013
r2:  0.0013
MAE:  12.8484
MSE:  283.4589
RMSE:  16.8362
explained_variance:  0.0014
r2:  0.0014
MAE:  12.851
MSE:  283.4349
RMSE:  16.8355
```



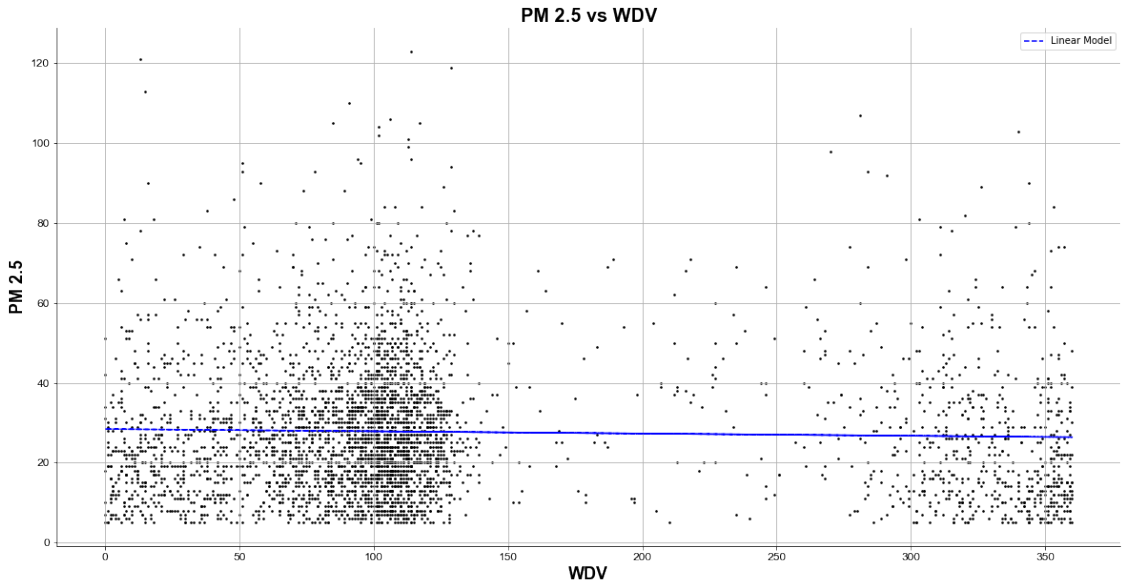
```
explained_variance:  0.0014
r2:  0.0014
MAE:  12.851
MSE:  283.4349
RMSE:  16.8355
explained_variance:  0.007
r2:  0.007
MAE:  12.7767
MSE:  281.8413
RMSE:  16.7881
```




```
explained_variance:  0.007
r2:  0.007
MAE:  12.7767
MSE:  281.8413
RMSE:  16.7881
explained_variance:  0.0051
r2:  0.0051
MAE:  12.8504
MSE:  282.3919
RMSE:  16.8045
```




```
explained_variance:  0.0051
r2:  0.0051
MAE:  12.8504
MSE:  282.3919
RMSE:  16.8045
explained_variance:  0.001
r2:  0.001
MAE:  12.8522
MSE:  283.5324
RMSE:  16.8384
```



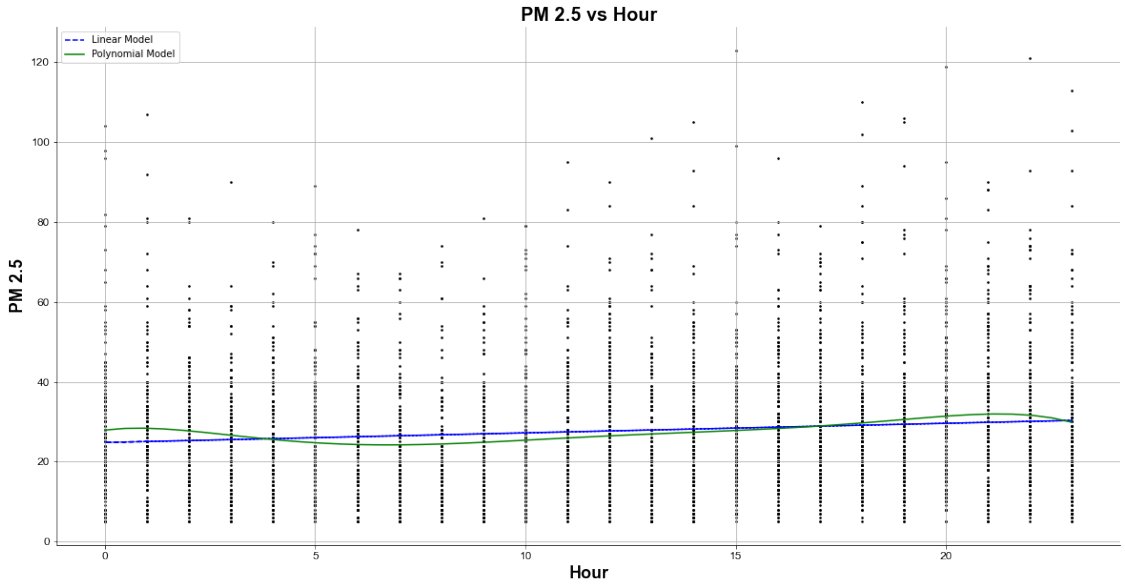
```
explained_variance:  0.001
r2:  0.001
MAE:  12.8522
MSE:  283.5324
RMSE:  16.8384
```

1.19 PM 2.5 vs Hour

[ ]:
```
nombre = "Hour"

deg = 6
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
```

```
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0096
r2:  0.0096
MAE:  12.8232
MSE:  281.1104
RMSE:  16.7663
0.019180298052115963
explained_variance:  0.0192
r2:  0.0192
MAE:  12.7589
MSE:  278.3859
RMSE:  16.6849
```



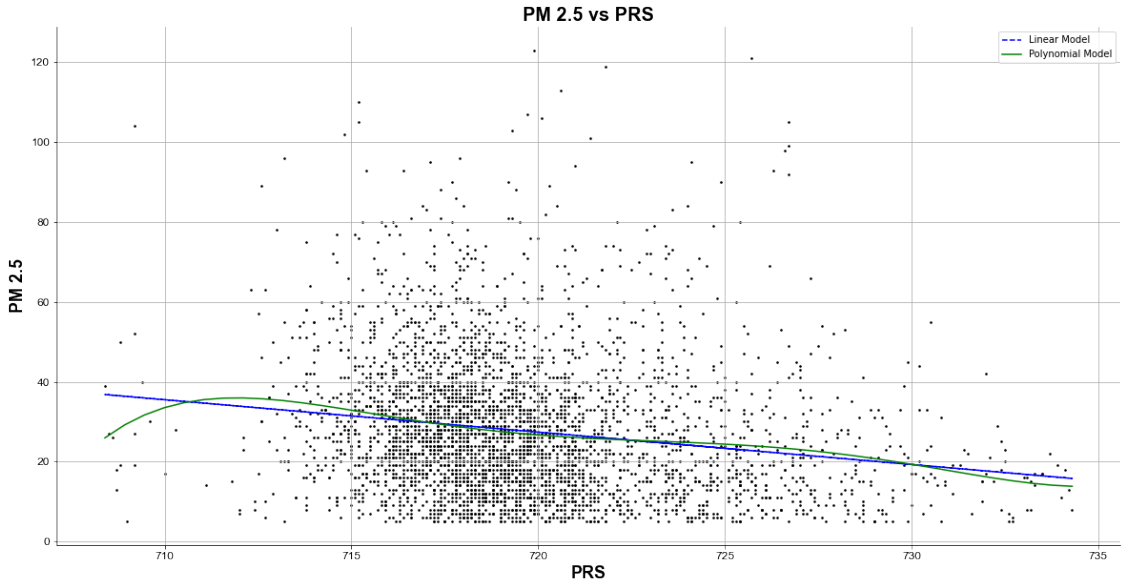
1.20 PM 2.5 vs PRS

[ ]:
```
nombre = "PRS"

deg = 6
y = PM_25.values
x = df[nombre].values

# Reshape Values#
```

```
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0317
r2:  0.0317
MAE:  12.5394
MSE:  274.834
RMSE:  16.5781
0.034641081600879975
explained_variance:  0.0346
r2:  0.0346
MAE:  12.53
MSE:  273.9977
RMSE:  16.5529
```

```
C:\Python39\lib\site-packages\IPython\core\interactiveshell.py:3441:
RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```

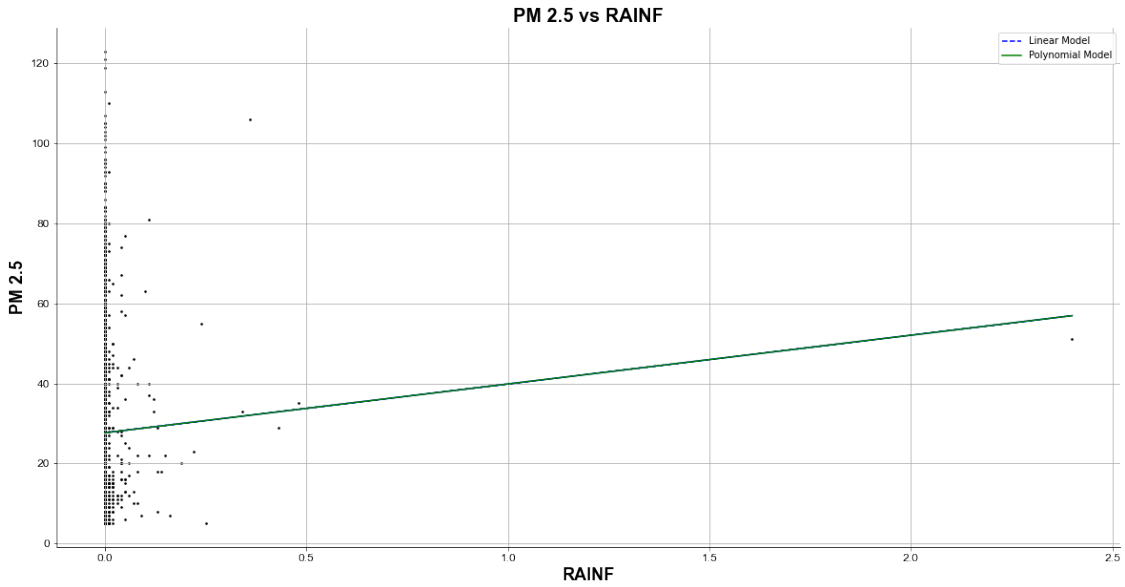


1.21 PM 2.5 vs RAINF

[ ]:
```
nombre = "RAINF"

deg = 1
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
```

```
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0009
r2:  0.0009
MAE:  12.8564
MSE:  283.5616
RMSE:  16.8393
0.0009453057974132539
explained_variance:  0.0009
r2:  0.0009
MAE:  12.8564
MSE:  283.5616
RMSE:  16.8393
```

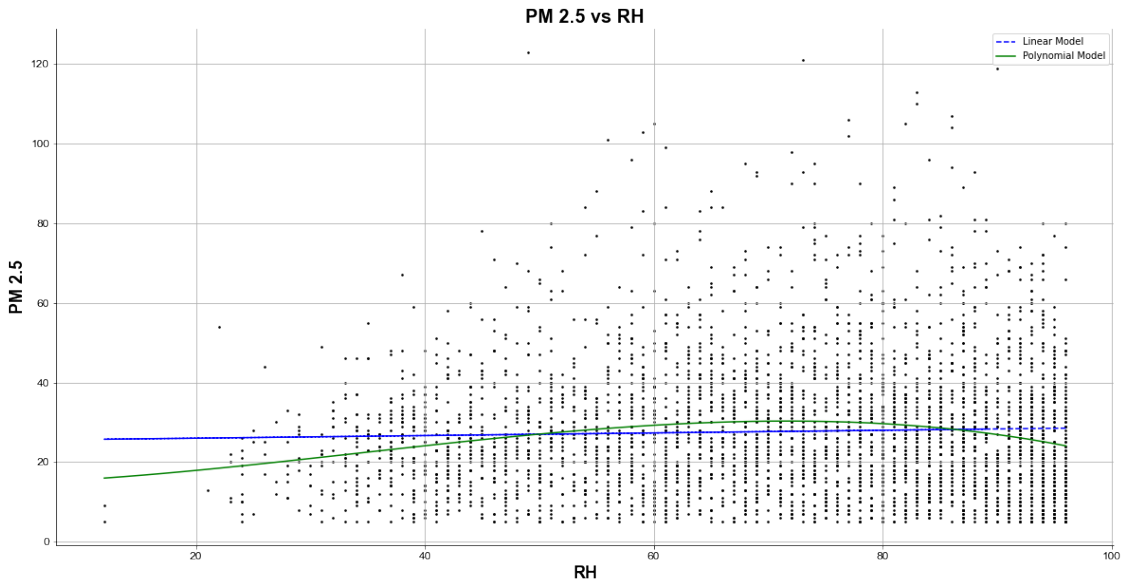


1.22 PM 2.5 vs RH

[ ]:
```
nombre = "RH"

deg = 3
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]
```

```
mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0013
r2:  0.0013
MAE:  12.8484
MSE:  283.4589
RMSE:  16.8362
0.020915777966912907
explained_variance:  0.0209
r2:  0.0209
MAE:  12.7254
MSE:  277.8933
RMSE:  16.6701
```

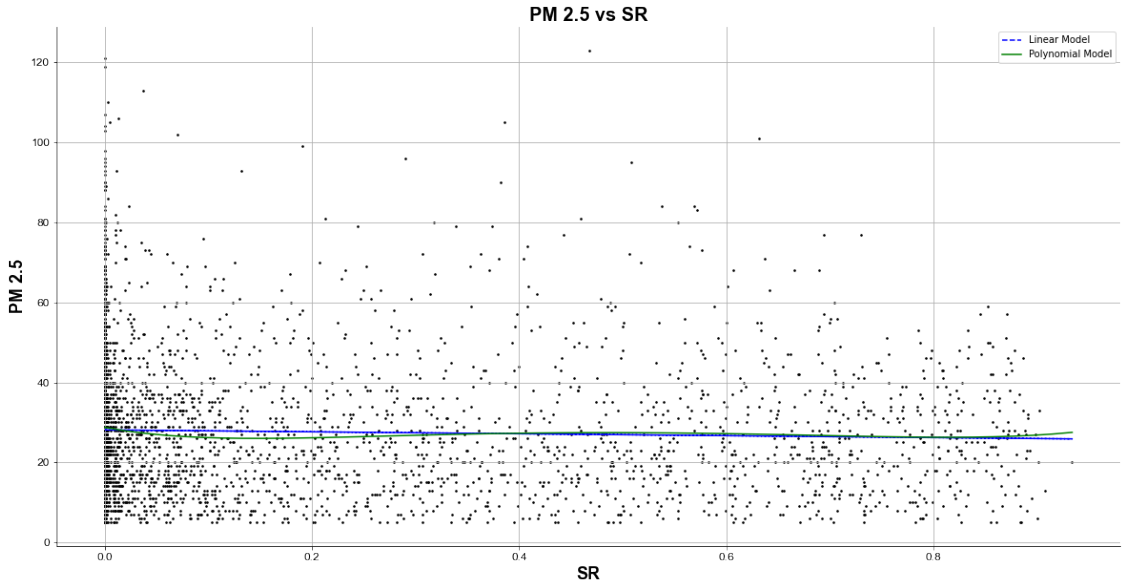


1.23 PM 2.5 vs SR

[ ]:
```
nombre = "SR"

deg = 6
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])
```

```
regression_results(y,predicted)
```

```
explained_variance:  0.0014
r2:  0.0014
MAE:  12.851
MSE:  283.4349
RMSE:  16.8355
0.0036601879214738 23
explained_variance:  0.0037
r2:  0.0037
MAE:  12.84
MSE:  282.791
RMSE:  16.8164
```

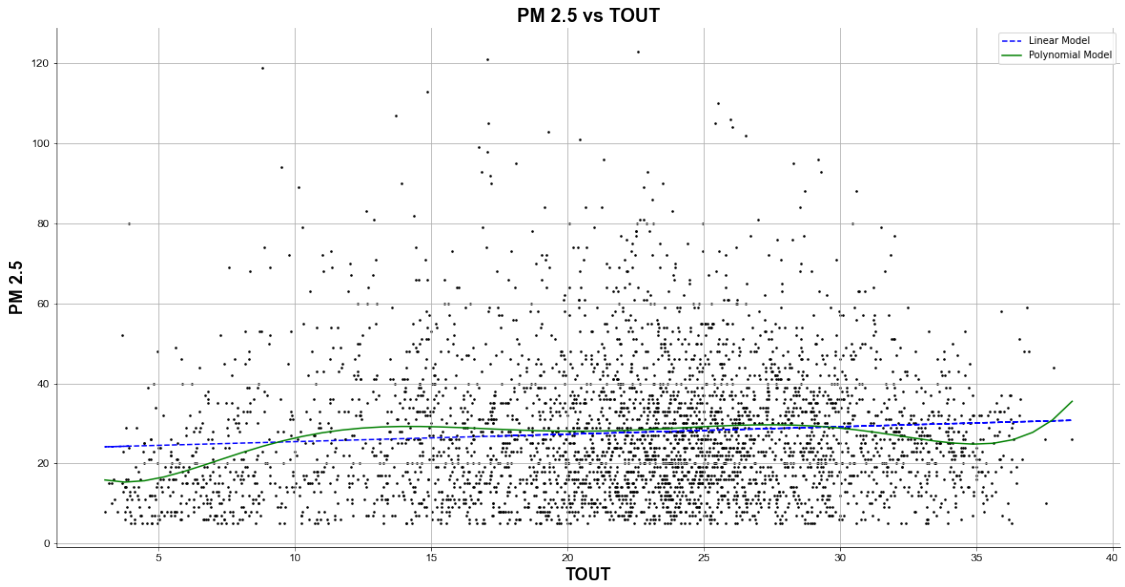


1.24 PM 2.5 vs TOU

[ ]:
```
nombre = "TOUT"

deg = 6
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
```

```
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.007
r2:  0.007
MAE:  12.7767
MSE:  281.8413
RMSE:  16.7881
0.028606063368382673
explained_variance:  0.0286
r2:  0.0286
MAE:  12.5804
MSE:  275.7106
RMSE:  16.6045
```

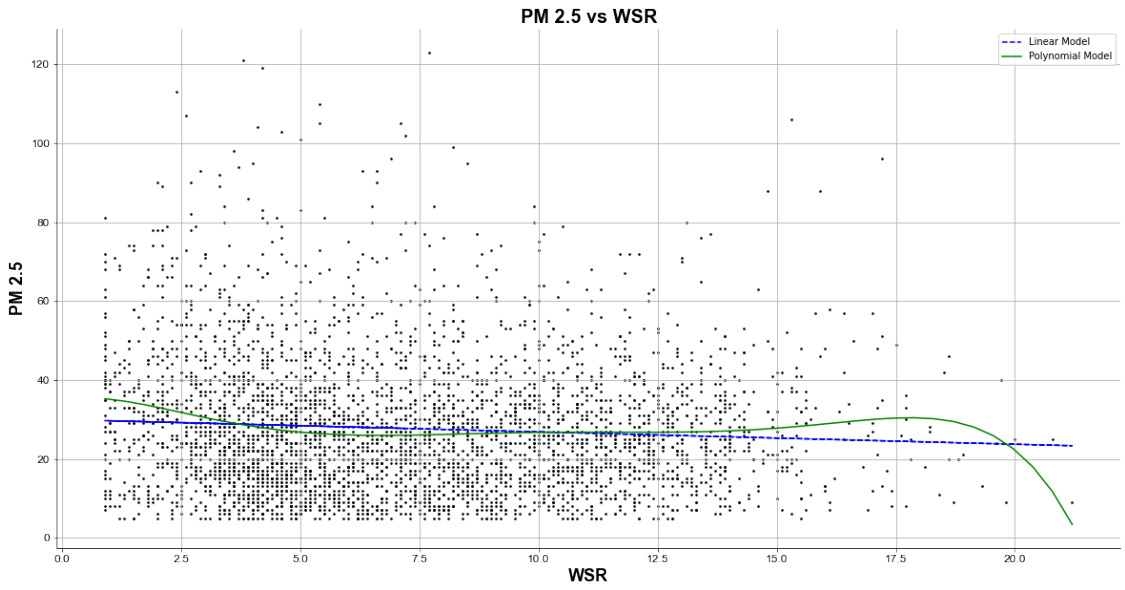


1.25 PM 2.5 vs WSR

[ ]:
```
nombre = "WSR"

deg = 6
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]
```

```
mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])

regression_results(y,predicted)
```

```
explained_variance:  0.0051
r2:  0.0051
MAE:  12.8504
MSE:  282.3919
RMSE:  16.8045
0.017639224785839458
explained_variance:  0.0176
r2:  0.0176
MAE:  12.735
MSE:  278.8233
RMSE:  16.698
```

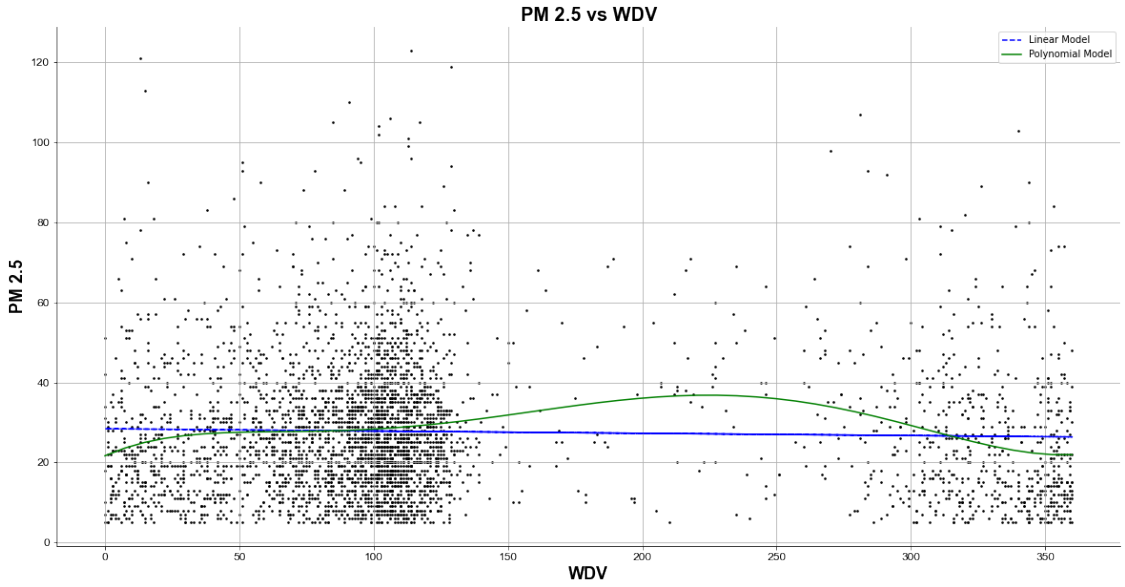


1.26 PM 2.5 vs WDV

[ ]:
```
nombre = "WDV"

deg = 6
y = PM_25.values
x = df[nombre].values

# Reshape Values#
x = x.reshape(-1,1)
x.shape

# Reshape Values#
y = y.reshape(-1,1)
y.shape

#Linear model#
lm = LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)

## Plots for PM_25 vs Hour#
plt.figure(figsize=(20,10))
plt.scatter(x,y, marker=".", c = "k" , s = 10)
plt.plot(x, lmp, 'b--')
y = PM_25
x = df[nombre]

mymodel = np.poly1d(np.polyfit(x, y, deg))
print(r2_score(y, mymodel(x)))
predicted = mymodel(x)
myline = np.linspace(min(x),max(x))
plt.plot(myline, mymodel(myline),'g-')

plt.title("PM 2.5 vs " + nombre ,**fontT)
plt.xlabel(nombre,**fontL)
plt.ylabel("PM 2.5",**fontL)
plt.xticks(fontsize = 12 , family = "Arial")
plt.yticks(fontsize = 12 , family = "Arial")
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.grid("On")
plt.legend(['Linear Model','Polynomial Model'])
```

```
regression_results(y,predicted)
```

```
explained_variance:  0.001
r2:  0.001
MAE:  12.8522
MSE:  283.5324
RMSE:  16.8384
0.02193911346519728
explained_variance:  0.0219
r2:  0.0219
MAE:  12.6311
MSE:  277.6029
RMSE:  16.6614
```



1.27 Multilinear

O3 Linear Regression

[ ]:
```
## We get back the data that we are analyzing ##
regression_columnNames = column_names.drop(['CONT'])

## Predicting Variable ##
x  = df[regression_columnNames]


## Y model of O3 ##
y = O3.values
```

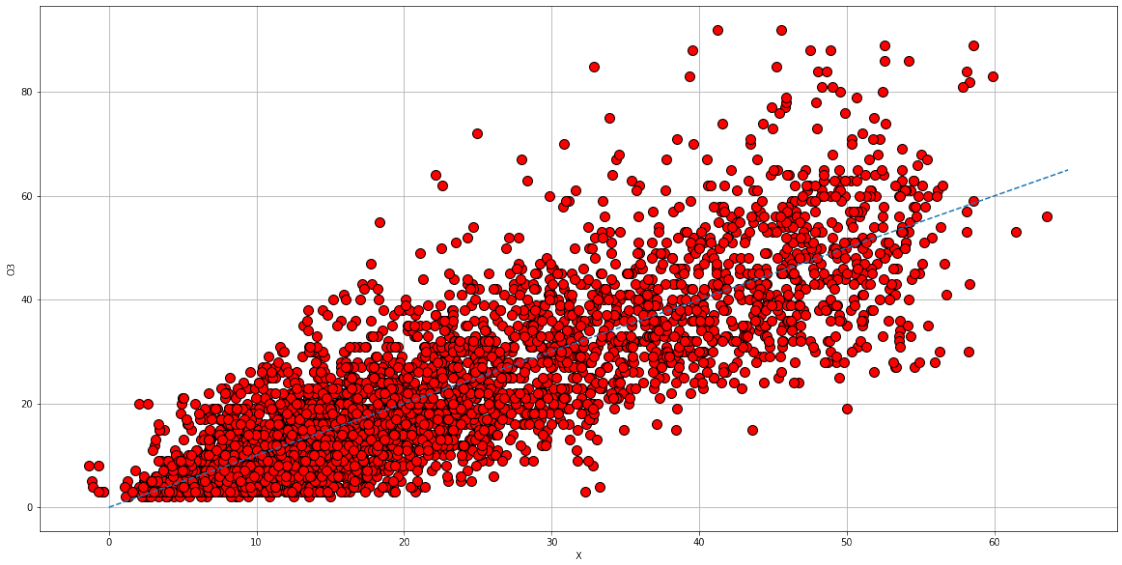
[ ]:
```
#Reshape the Values#
y = y.reshape(-1,1)
y.shape
```

[ ]:
```
(3859, 1)
```

[ ]:
```
lm = sk.linear_model.LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)
```

```
explained_variance:  0.6833
r2:  0.6833
MAE:  6.956
MSE:  86.3181
RMSE:  9.2908
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(lmp,y,"ok",markersize =10, linewidth = 4 , markerfacecolor = 'red')
plt.plot([0,65],[0,65],"--")
plt.ylabel("O3")
plt.xlabel("X")
plt.grid(True)
```



Scaling the dataset

[ ]:
```
## We get back the data that we are analyzing ##
regression_columnNames = column_names.drop(['CONT'])
```

```
## Predicting Variable ##
x  = df[regression_columnNames]


## Y model of O3 ##
y = O3.values
```

[ ]:
```
#Reshape the Values#
y = y.reshape(-1,1)
y.shape
```

[ ]: (3859, 1)

[ ]:
```
## Scales ##
scaling_procedure_1 = sk.preprocessing.MinMaxScaler(feature_range= (0,1))
scaling_procedure_2 = sk.preprocessing.MinMaxScaler(feature_range= (0,1))
```

[ ]:
```
#Datos ya escalados#
x_scaled = scaling_procedure_1.fit_transform(x)
y_scaled = scaling_procedure_1.fit_transform(y)
```

[ ]:
```
##Datos de separados##

##Dividio los datos Entrenamiento##
x_train, x_test, y_train, y_test = sk.model_selection.
 ↪train_test_split(x_scaled,y_scaled,test_size= 0.2, random_state = 42)
```

[ ]:
```
print("Size of the full data set: ",x_scaled.shape)
print("Size of the training data set: ",x_train.shape)
print("Size of the test data set: ",x_test.shape)
```

```
Size of the full data set:  (3859, 9)
Size of the training data set:  (3087, 9)
Size of the test data set:  (772, 9)
```

[ ]:
```
#Linear Model#
LR = LinearRegression()
```

[ ]:
```
#Fit the training data#
LR.fit(x_train,y_train)
```

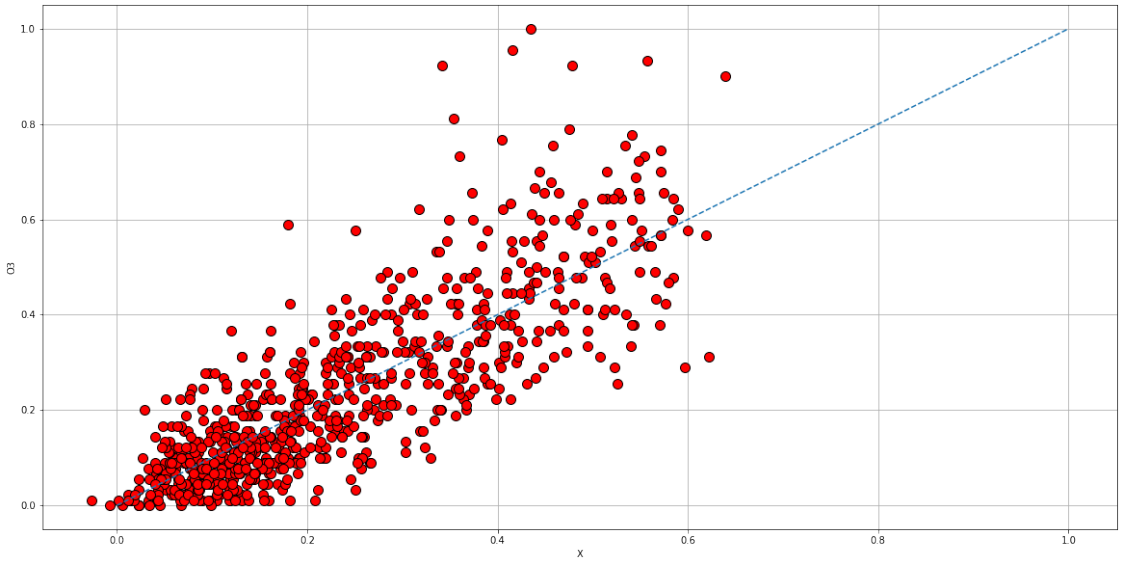
[ ]: LinearRegression()

[ ]:
```
y_prediction =  LR.predict(x_test)
regression_results(y_test,y_prediction)
```

```
explained_variance:  0.6803
r2:  0.6795
MAE:  0.0791
```

```
MSE:  0.0116
RMSE:  0.1075
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(y_prediction,y_test,"ok",markersize =10, linewidth = 4 ,␣
 ↪markerfacecolor = 'red')
plt.plot([0,1],[0,1],"--")
plt.ylabel("O3")
plt.xlabel("X")
plt.grid(True)
```



1.28 PM_25 Linear Regression

[ ]:
```
df = pd.read_csv("Monterrey Pollution Data 2.csv")
df.head()

df = df.drop(["PM2.5Diff"],axis=1)

PM_25 = df["PM2.5"]

column_names = df.columns
column_names = column_names.drop(["O3","PM2.5","PM10"])
```

[ ]:
```
## We get back the data that we are analyzing ##
regression_columnNames = column_names.drop(['CONT'])

## Predicting Variable ##
```

```
x  = df[regression_columnNames]


## Y model of O3 ##
y = PM_25.values
```

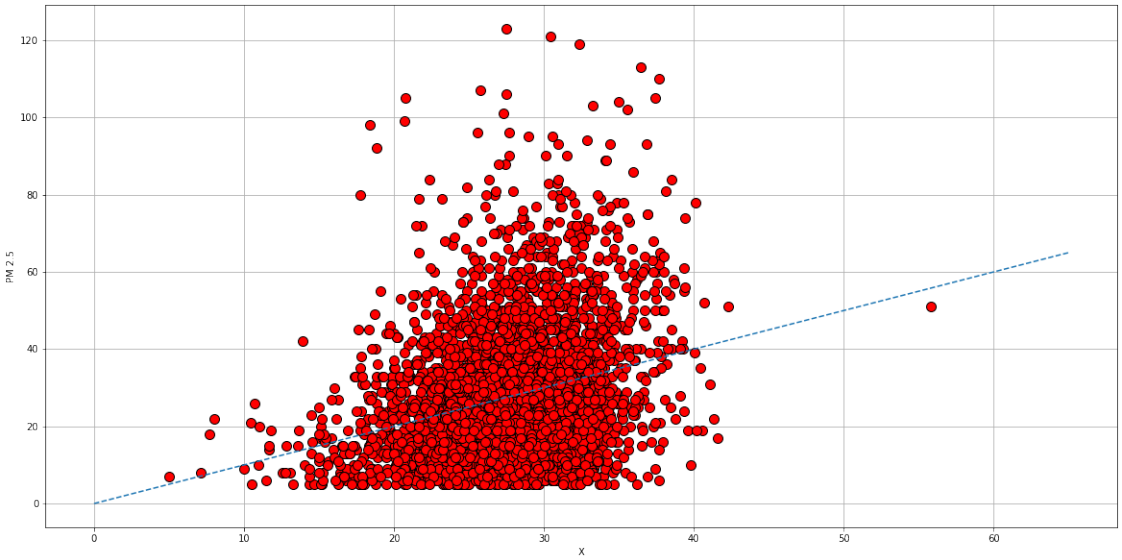
[ ]:
```
#Reshape the Values#
y = y.reshape(-1,1)
y.shape
```

[ ]: (3859, 1)

[ ]:
```
lm = sk.linear_model.LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)
```

```
explained_variance:  0.0728
r2:  0.0728
MAE:  12.2903
MSE:  263.1581
RMSE:  16.2221
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(lmp,y,"ok",markersize =10, linewidth = 4 , markerfacecolor = 'red')
plt.plot([0,65],[0,65],"--")
plt.ylabel("PM 2.5")
plt.xlabel("X")
plt.grid(True)
```



Scaling the dataset

[ ]:
```
df = pd.read_csv("Monterrey Pollution Data 2.csv")
df.head()

df = df.drop(["PM2.5Diff"],axis=1)

PM_25 = df["PM2.5"]

column_names = df.columns
column_names = column_names.drop(["O3","PM2.5","PM10"])
```

[ ]:
```
## We get back the data that we are analyzing ##
regression_columnNames = column_names.drop(['CONT'])

## Predicting Variable ##
x  = df[regression_columnNames]

## Y model of O3 ##
y = PM_25.values
```

[ ]:
```
#Reshape the Values#
y = y.reshape(-1,1)
y.shape
```

[ ]:
```
(3859, 1)
```

[ ]:
```
## Scales ##
scaling_procedure_1 = sk.preprocessing.MinMaxScaler(feature_range= (0,1))
scaling_procedure_2 = sk.preprocessing.MinMaxScaler(feature_range= (0,1))
```

[ ]:
```
#Datos ya escalados#
x_scaled = scaling_procedure_1.fit_transform(x)
y_scaled = scaling_procedure_1.fit_transform(y)
```

[ ]:
```
##Datos de separados##

##Dividio los datos Entrenamiento##
x_train, x_test, y_train, y_test = sk.model_selection.
 ↪train_test_split(x_scaled,y_scaled,test_size= 0.2, random_state = 42)
```

[ ]:
```
print("Size of the full data set: ",x_scaled.shape)
print("Size of the training data set: ",x_train.shape)
print("Size of the test data set: ",x_test.shape)
```

```
Size of the full data set:  (3859, 9)
Size of the training data set:  (3087, 9)
Size of the test data set:  (772, 9)
```

[ ]:
```
#Linear Model#
LR = LinearRegression()
```

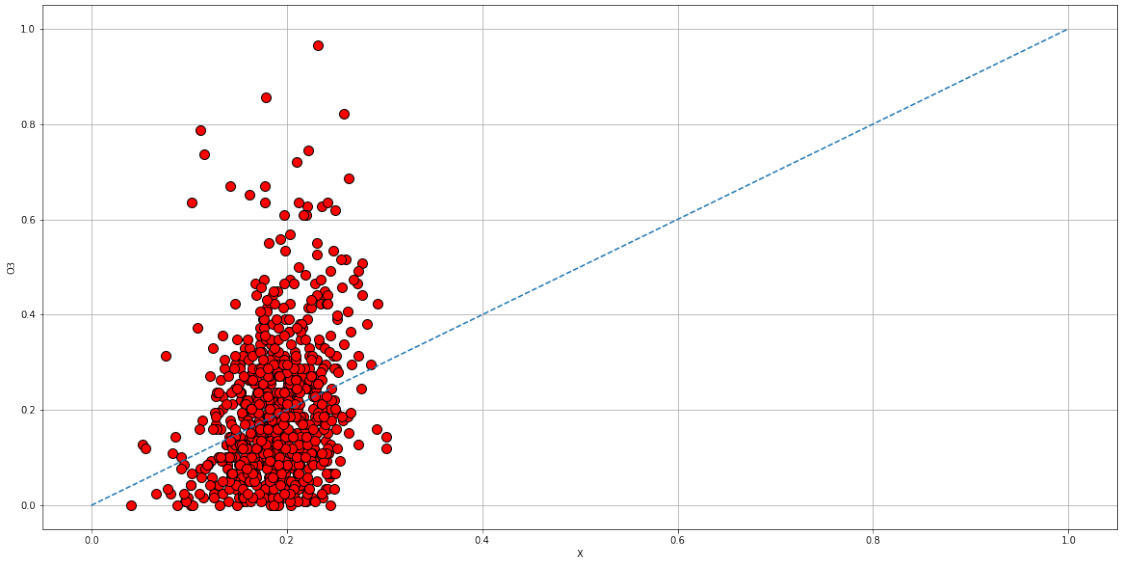
[ ]:
```
#Fit the training data#
LR.fit(x_train,y_train)
```

[ ]:
```
LinearRegression()
```

[ ]:
```
y_prediction =  LR.predict(x_test)
regression_results(y_test,y_prediction)
```

```
explained_variance:  0.0615
r2:  0.0572
MAE:  0.1073
MSE:  0.0206
RMSE:  0.1436
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(y_prediction,y_test,"ok",markersize =10, linewidth = 4 ,␣
 ↪markerfacecolor = 'red')
plt.plot([0,1],[0,1],"--")
plt.ylabel("O3")
plt.xlabel("X")
plt.grid(True)
```



1.29 Machine Learning

As we can see the model for PM 2.5 isn't the best fitted for the analysis so we will make a classification model for the PM 2.5

[ ]:
```
#Read data
df = pd.read_csv('Monterrey Pollution Data 3.csv')
df.head()


df = df.drop(["PM2.5Diff"],axis=1)

O3 = df ["O3"]
PM_25 = df["PM2.5"]
PM_10 = df["PM10"]
CONT  = df["CONT"]

column_names = df.columns
column_names = column_names.drop(["O3","PM2.5","PM10","CONT"])

x  = df[column_names]
```

[ ]:
```
#Dependant
x = x
#Independant#
y = CONT
```

[ ]:
```
#Separate train and test data
x_train,x_test,y_train,y_test = sk.model_selection.
 ↪train_test_split(x,y,train_size=0.20,random_state=0)

#Standarize data. Subtract mean and devide by standard deviation
sc = sk.preprocessing.StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

[ ]:
```
from sklearn.naive_bayes import GaussianNB
#Fit classifier with train data
NB = GaussianNB()
NB.fit(x_train,y_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
#Predict test data
y_pred=NB.predict(x_test)
```

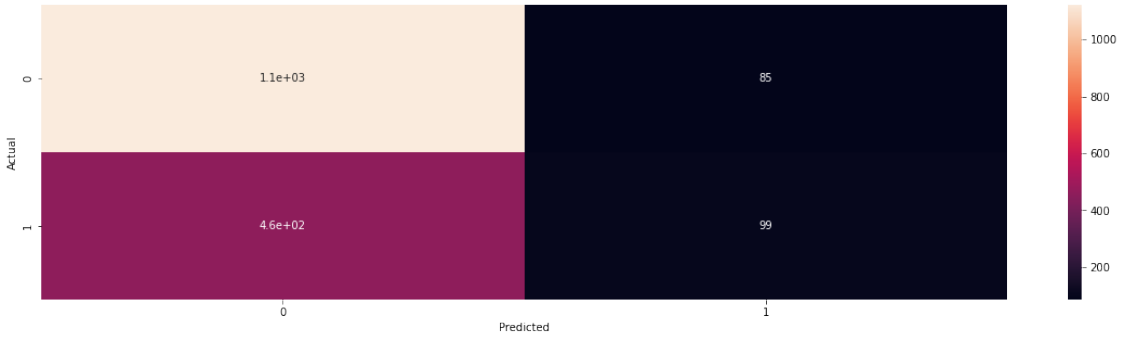
[ ]:
```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test,y_pred)
print(cm)
tn, fp, fn, tp = cm.ravel()
```

```
[[1123   85]
 [ 455   99]]
```

[ ]:
```
cm2 = pd.crosstab(y_test,y_pred,rownames=['Actual'],colnames=['Predicted'])
sns.heatmap(cm2,annot=True)
```

[ ]:
```
<AxesSubplot:xlabel='Predicted', ylabel='Actual'>
```



[ ]:
```
#Get performance measures
from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.71      0.93      0.81      1208
           1       0.54      0.18      0.27       554

    accuracy                           0.69      1762
   macro avg       0.62      0.55      0.54      1762
weighted avg       0.66      0.69      0.64      1762
```

[ ]:
```
recall=tp/(tp+fn)
print("Likelihood Ratio",recall/(1-recall))
print("Accuracy=",(tp+tn)/(tp+tn+fp+fn))
```

```
Likelihood Ratio 0.2175824175824176
Accuracy= 0.6935300794551645
```

1.30 Logistic Regresion

[ ]:
```
#Read data
df = pd.read_csv('Monterrey Pollution Data 3.csv')
df.head()


df = df.drop(["PM2.5Diff"],axis=1)


column_names = df.columns
column_names = column_names.drop(["CONT"])

x  = df[column_names]
```

[ ]:
```
## Predicting Variable ##
x  = x.values


## Y model of O3 ##
y = df['CONT']
```

[ ]:
```
## Escalas ##
scaling_procedure_1 = sk.preprocessing.MinMaxScaler(feature_range= (0,1))
```

[ ]:
```
#Datos ya escalados#
x_scaled = scaling_procedure_1.fit_transform(x)
y_scaled = y
```

[ ]:
```
##Divididr la base de datos##

##Entrenamiento##

train_ratio = 0.7
validation_ratio = 0.15
test_ratio = 0.15
```

[ ]:
```
##Datos de separados##

##Dividio los datos Entrenamiento##
x_train, x_test, y_train, y_test = sk.model_selection.
 ↪train_test_split(x_scaled,y_scaled,test_size=1-train_ratio)
```

[ ]:
```
##Dividio los datos Validacion y prueba ##
x_val,x_test,y_val,y_test = sk.model_selection.
 ↪train_test_split(x_test,y_test,test_size= test_ratio/
 ↪(test_ratio+validation_ratio))
```

[ ]:
```
print("Size of the full data set: ",x_scaled.shape)
print("Size of the training data set: ",x_train.shape)
print("Size of the validation data set: ",x_val.shape)
print("Size of the test data set: ",x_test.shape)
```

```
Size of the full data set:  (2202, 12)
Size of the training data set:  (1541, 12)
Size of the validation data set:  (330, 12)
Size of the test data set:  (331, 12)
```

[ ]:
```
polution_model = tf.keras.models.Sequential()
##Capa de entrada##
polution_model.add(tf.keras.layers.Dense(20,input_dim = 12, activation =␣
↪"relu"))
polution_model.add(tf.keras.layers.Dense(10, activation = "relu"))
## Salida ##
polution_model.add(tf.keras.layers.Dense(1, activation = "sigmoid"))
```

[ ]:
```
##Problemas de una sola salida ##
polution_model.compile(loss="binary_crossentropy", optimizer="adam" ,␣
↪metrics="accuracy")
```

[ ]:
```
polution_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 20)                260
_________________________________________________________________
dense_16 (Dense)             (None, 10)                210
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 11
=================================================================
Total params: 481
Trainable params: 481
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
history = polution_model.fit(x_train,y_train,epochs= 150, batch_size=␣
↪16,validation_data=(x_val,y_val))
```

```
Epoch 1/150
97/97 [==============================] - 1s 2ms/step - loss: 0.6567 - accuracy:
0.6347 - val_loss: 0.6352 - val_accuracy: 0.6394
Epoch 2/150
97/97 [==============================] - 0s 1ms/step - loss: 0.5725 - accuracy:
0.6866 - val_loss: 0.5843 - val_accuracy: 0.6394
Epoch 3/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.5030 - accuracy:
0.7430 - val_loss: 0.5039 - val_accuracy: 0.7273
Epoch 4/150
97/97 [==============================] - 0s 1ms/step - loss: 0.4078 - accuracy:
0.8287 - val_loss: 0.3962 - val_accuracy: 0.8394
Epoch 5/150
97/97 [==============================] - 0s 1ms/step - loss: 0.3163 - accuracy:
0.8923 - val_loss: 0.3091 - val_accuracy: 0.8788
Epoch 6/150
97/97 [==============================] - 0s 1ms/step - loss: 0.2461 - accuracy:
0.9182 - val_loss: 0.2470 - val_accuracy: 0.9121
Epoch 7/150
97/97 [==============================] - 0s 999us/step - loss: 0.2030 -
accuracy: 0.9422 - val_loss: 0.2153 - val_accuracy: 0.9030
Epoch 8/150
97/97 [==============================] - 0s 1ms/step - loss: 0.1704 - accuracy:
0.9481 - val_loss: 0.1827 - val_accuracy: 0.9364
Epoch 9/150
97/97 [==============================] - 0s 1ms/step - loss: 0.1522 - accuracy:
0.9526 - val_loss: 0.1701 - val_accuracy: 0.9303
Epoch 10/150
97/97 [==============================] - 0s 1ms/step - loss: 0.1337 - accuracy:
0.9578 - val_loss: 0.1488 - val_accuracy: 0.9424
Epoch 11/150
97/97 [==============================] - 0s 1ms/step - loss: 0.1222 - accuracy:
0.9624 - val_loss: 0.1631 - val_accuracy: 0.9182
Epoch 12/150
97/97 [==============================] - 0s 999us/step - loss: 0.1107 -
accuracy: 0.9695 - val_loss: 0.1385 - val_accuracy: 0.9424
Epoch 13/150
97/97 [==============================] - 0s 1ms/step - loss: 0.1024 - accuracy:
0.9721 - val_loss: 0.1229 - val_accuracy: 0.9485
Epoch 14/150
97/97 [==============================] - 0s 999us/step - loss: 0.0929 -
accuracy: 0.9721 - val_loss: 0.1158 - val_accuracy: 0.9485
Epoch 15/150
97/97 [==============================] - 0s 989us/step - loss: 0.0861 -
accuracy: 0.9766 - val_loss: 0.1033 - val_accuracy: 0.9727
Epoch 16/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0804 - accuracy:
0.9753 - val_loss: 0.1101 - val_accuracy: 0.9576
Epoch 17/150
97/97 [==============================] - 0s 999us/step - loss: 0.0777 -
accuracy: 0.9799 - val_loss: 0.0929 - val_accuracy: 0.9697
Epoch 18/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0731 - accuracy:
0.9786 - val_loss: 0.0929 - val_accuracy: 0.9727
Epoch 19/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0674 - accuracy:
0.9825 - val_loss: 0.0859 - val_accuracy: 0.9788
Epoch 20/150
97/97 [==============================] - 0s 989us/step - loss: 0.0654 -
accuracy: 0.9805 - val_loss: 0.0780 - val_accuracy: 0.9818
Epoch 21/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0620 - accuracy:
0.9851 - val_loss: 0.0815 - val_accuracy: 0.9788
Epoch 22/150
97/97 [==============================] - 0s 989us/step - loss: 0.0576 -
accuracy: 0.9838 - val_loss: 0.0722 - val_accuracy: 0.9818
Epoch 23/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0556 - accuracy:
0.9851 - val_loss: 0.0820 - val_accuracy: 0.9636
Epoch 24/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0535 - accuracy:
0.9883 - val_loss: 0.0794 - val_accuracy: 0.9727
Epoch 25/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0539 - accuracy:
0.9857 - val_loss: 0.0654 - val_accuracy: 0.9788
Epoch 26/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0519 - accuracy:
0.9851 - val_loss: 0.1001 - val_accuracy: 0.9424
Epoch 27/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0492 - accuracy:
0.9877 - val_loss: 0.0657 - val_accuracy: 0.9879
Epoch 28/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0510 - accuracy:
0.9838 - val_loss: 0.0824 - val_accuracy: 0.9667
Epoch 29/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0487 - accuracy:
0.9825 - val_loss: 0.0583 - val_accuracy: 0.9909
Epoch 30/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0447 - accuracy:
0.9883 - val_loss: 0.0583 - val_accuracy: 0.9788
Epoch 31/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0445 - accuracy:
0.9877 - val_loss: 0.0555 - val_accuracy: 0.9909
Epoch 32/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0436 - accuracy:
0.9883 - val_loss: 0.0552 - val_accuracy: 0.9879
Epoch 33/150
97/97 [==============================] - 0s 999us/step - loss: 0.0409 -
accuracy: 0.9922 - val_loss: 0.0525 - val_accuracy: 0.9909
Epoch 34/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0411 - accuracy:
0.9883 - val_loss: 0.0594 - val_accuracy: 0.9848
Epoch 35/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0401 - accuracy:
0.9890 - val_loss: 0.0502 - val_accuracy: 0.9909
Epoch 36/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0382 - accuracy:
0.9903 - val_loss: 0.0494 - val_accuracy: 0.9909
Epoch 37/150
97/97 [==============================] - 0s 979us/step - loss: 0.0380 -
accuracy: 0.9896 - val_loss: 0.0480 - val_accuracy: 0.9879
Epoch 38/150
97/97 [==============================] - 0s 999us/step - loss: 0.0374 -
accuracy: 0.9890 - val_loss: 0.0549 - val_accuracy: 0.9879
Epoch 39/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0356 - accuracy:
0.9909 - val_loss: 0.0647 - val_accuracy: 0.9788
Epoch 40/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0366 - accuracy:
0.9877 - val_loss: 0.0623 - val_accuracy: 0.9758
Epoch 41/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0361 - accuracy:
0.9903 - val_loss: 0.0441 - val_accuracy: 0.9909
Epoch 42/150
97/97 [==============================] - 0s 968us/step - loss: 0.0363 -
accuracy: 0.9909 - val_loss: 0.0627 - val_accuracy: 0.9758
Epoch 43/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0341 - accuracy:
0.9922 - val_loss: 0.0422 - val_accuracy: 0.9909
Epoch 44/150
97/97 [==============================] - 0s 979us/step - loss: 0.0330 -
accuracy: 0.9922 - val_loss: 0.0462 - val_accuracy: 0.9909
Epoch 45/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0327 - accuracy:
0.9916 - val_loss: 0.0410 - val_accuracy: 0.9909
Epoch 46/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0321 - accuracy:
0.9916 - val_loss: 0.0728 - val_accuracy: 0.9667
Epoch 47/150
97/97 [==============================] - 0s 968us/step - loss: 0.0331 -
accuracy: 0.9903 - val_loss: 0.0391 - val_accuracy: 0.9909
Epoch 48/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0374 - accuracy:
0.9870 - val_loss: 0.0487 - val_accuracy: 0.9879
Epoch 49/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0339 - accuracy:
0.9890 - val_loss: 0.0514 - val_accuracy: 0.9848
Epoch 50/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0327 - accuracy:
0.9903 - val_loss: 0.0379 - val_accuracy: 0.9909
Epoch 51/150
```

```
97/97 [==============================] - 0s 968us/step - loss: 0.0297 -
accuracy: 0.9916 - val_loss: 0.0468 - val_accuracy: 0.9848
Epoch 52/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0303 - accuracy:
0.9903 - val_loss: 0.0723 - val_accuracy: 0.9636
Epoch 53/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0321 - accuracy:
0.9883 - val_loss: 0.0425 - val_accuracy: 0.9909
Epoch 54/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0291 - accuracy:
0.9916 - val_loss: 0.0354 - val_accuracy: 0.9939
Epoch 55/150
97/97 [==============================] - 0s 989us/step - loss: 0.0322 -
accuracy: 0.9864 - val_loss: 0.0683 - val_accuracy: 0.9636
Epoch 56/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0283 - accuracy:
0.9909 - val_loss: 0.0346 - val_accuracy: 0.9939
Epoch 57/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0284 - accuracy:
0.9916 - val_loss: 0.0376 - val_accuracy: 0.9909
Epoch 58/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0265 - accuracy:
0.9929 - val_loss: 0.0400 - val_accuracy: 0.9909
Epoch 59/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0290 - accuracy:
0.9903 - val_loss: 0.0317 - val_accuracy: 0.9909
Epoch 60/150
97/97 [==============================] - 0s 968us/step - loss: 0.0291 -
accuracy: 0.9916 - val_loss: 0.0336 - val_accuracy: 0.9939
Epoch 61/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0267 - accuracy:
0.9909 - val_loss: 0.0316 - val_accuracy: 0.9939
Epoch 62/150
97/97 [==============================] - 0s 958us/step - loss: 0.0253 -
accuracy: 0.9922 - val_loss: 0.0488 - val_accuracy: 0.9848
Epoch 63/150
97/97 [==============================] - 0s 999us/step - loss: 0.0333 -
accuracy: 0.9870 - val_loss: 0.0463 - val_accuracy: 0.9848
Epoch 64/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0266 - accuracy:
0.9916 - val_loss: 0.0294 - val_accuracy: 0.9939
Epoch 65/150
97/97 [==============================] - 0s 999us/step - loss: 0.0253 -
accuracy: 0.9935 - val_loss: 0.0312 - val_accuracy: 0.9970
Epoch 66/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0261 - accuracy:
0.9896 - val_loss: 0.0393 - val_accuracy: 0.9879
Epoch 67/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0271 - accuracy:
0.9909 - val_loss: 0.0276 - val_accuracy: 0.9939
Epoch 68/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0256 - accuracy:
0.9929 - val_loss: 0.0364 - val_accuracy: 0.9939
Epoch 69/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0237 - accuracy:
0.9929 - val_loss: 0.0316 - val_accuracy: 0.9939
Epoch 70/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0234 - accuracy:
0.9916 - val_loss: 0.0649 - val_accuracy: 0.9667
Epoch 71/150
97/97 [==============================] - 0s 999us/step - loss: 0.0248 -
accuracy: 0.9929 - val_loss: 0.0317 - val_accuracy: 0.9909
Epoch 72/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0248 - accuracy:
0.9916 - val_loss: 0.0505 - val_accuracy: 0.9788
Epoch 73/150
97/97 [==============================] - 0s 999us/step - loss: 0.0221 -
accuracy: 0.9961 - val_loss: 0.0259 - val_accuracy: 0.9970
Epoch 74/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0239 - accuracy:
0.9903 - val_loss: 0.0270 - val_accuracy: 0.9939
Epoch 75/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0233 - accuracy:
0.9935 - val_loss: 0.0316 - val_accuracy: 0.9939
Epoch 76/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0242 - accuracy:
0.9903 - val_loss: 0.0354 - val_accuracy: 0.9909
Epoch 77/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0233 - accuracy:
0.9942 - val_loss: 0.0256 - val_accuracy: 0.9970
Epoch 78/150
97/97 [==============================] - 0s 968us/step - loss: 0.0244 -
accuracy: 0.9929 - val_loss: 0.0277 - val_accuracy: 0.9970
Epoch 79/150
97/97 [==============================] - 0s 989us/step - loss: 0.0224 -
accuracy: 0.9935 - val_loss: 0.0317 - val_accuracy: 0.9939
Epoch 80/150
97/97 [==============================] - 0s 999us/step - loss: 0.0223 -
accuracy: 0.9955 - val_loss: 0.0325 - val_accuracy: 0.9939
Epoch 81/150
97/97 [==============================] - 0s 2ms/step - loss: 0.0211 - accuracy:
0.9948 - val_loss: 0.0296 - val_accuracy: 0.9879
Epoch 82/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0215 - accuracy:
0.9942 - val_loss: 0.0457 - val_accuracy: 0.9848
Epoch 83/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0228 - accuracy:
0.9922 - val_loss: 0.0322 - val_accuracy: 0.9909
Epoch 84/150
97/97 [==============================] - 0s 989us/step - loss: 0.0201 -
accuracy: 0.9935 - val_loss: 0.0445 - val_accuracy: 0.9848
Epoch 85/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0213 - accuracy:
0.9922 - val_loss: 0.0714 - val_accuracy: 0.9636
Epoch 86/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0259 - accuracy:
0.9896 - val_loss: 0.0381 - val_accuracy: 0.9909
Epoch 87/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0197 - accuracy:
0.9948 - val_loss: 0.0582 - val_accuracy: 0.9727
Epoch 88/150
97/97 [==============================] - 0s 999us/step - loss: 0.0228 -
accuracy: 0.9909 - val_loss: 0.0550 - val_accuracy: 0.9727
Epoch 89/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0238 - accuracy:
0.9909 - val_loss: 0.0235 - val_accuracy: 0.9970
Epoch 90/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0217 - accuracy:
0.9935 - val_loss: 0.0327 - val_accuracy: 0.9909
Epoch 91/150
97/97 [==============================] - 0s 989us/step - loss: 0.0226 -
accuracy: 0.9903 - val_loss: 0.0466 - val_accuracy: 0.9818
Epoch 92/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0199 - accuracy:
0.9942 - val_loss: 0.0243 - val_accuracy: 0.9970
Epoch 93/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0202 - accuracy:
0.9935 - val_loss: 0.0376 - val_accuracy: 0.9848
Epoch 94/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0187 - accuracy:
0.9942 - val_loss: 0.0262 - val_accuracy: 0.9970
Epoch 95/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0201 - accuracy:
0.9942 - val_loss: 0.0211 - val_accuracy: 0.9970
Epoch 96/150
97/97 [==============================] - 0s 999us/step - loss: 0.0212 -
accuracy: 0.9922 - val_loss: 0.0232 - val_accuracy: 0.9909
Epoch 97/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0200 - accuracy:
0.9948 - val_loss: 0.0272 - val_accuracy: 0.9970
Epoch 98/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0183 - accuracy:
0.9935 - val_loss: 0.0338 - val_accuracy: 0.9909
Epoch 99/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0177 - accuracy:
0.9955 - val_loss: 0.0280 - val_accuracy: 0.9939
Epoch 100/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0169 - accuracy:
0.9955 - val_loss: 0.0199 - val_accuracy: 0.9970
Epoch 101/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0220 - accuracy:
0.9877 - val_loss: 0.0311 - val_accuracy: 0.9909
Epoch 102/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0178 - accuracy:
0.9955 - val_loss: 0.0217 - val_accuracy: 0.9970
Epoch 103/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0182 - accuracy:
0.9955 - val_loss: 0.0211 - val_accuracy: 0.9970
Epoch 104/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0190 - accuracy:
0.9942 - val_loss: 0.0319 - val_accuracy: 0.9909
Epoch 105/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0190 - accuracy:
0.9968 - val_loss: 0.0283 - val_accuracy: 0.9909
Epoch 106/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0171 - accuracy:
0.9968 - val_loss: 0.0246 - val_accuracy: 0.9939
Epoch 107/150
97/97 [==============================] - 0s 999us/step - loss: 0.0238 -
accuracy: 0.9903 - val_loss: 0.0217 - val_accuracy: 0.9939
Epoch 108/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0204 - accuracy:
0.9922 - val_loss: 0.0207 - val_accuracy: 0.9970
Epoch 109/150
97/97 [==============================] - 0s 999us/step - loss: 0.0210 -
accuracy: 0.9929 - val_loss: 0.0356 - val_accuracy: 0.9879
Epoch 110/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0171 - accuracy:
0.9929 - val_loss: 0.0317 - val_accuracy: 0.9909
Epoch 111/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0173 - accuracy:
0.9961 - val_loss: 0.0189 - val_accuracy: 0.9970
Epoch 112/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0184 - accuracy:
0.9955 - val_loss: 0.0245 - val_accuracy: 0.9970
Epoch 113/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0170 - accuracy:
0.9955 - val_loss: 0.0258 - val_accuracy: 0.9909
Epoch 114/150
97/97 [==============================] - 0s 989us/step - loss: 0.0167 -
accuracy: 0.9935 - val_loss: 0.0261 - val_accuracy: 0.9939
Epoch 115/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0191 - accuracy:
0.9916 - val_loss: 0.0182 - val_accuracy: 0.9970
Epoch 116/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0183 - accuracy:
0.9935 - val_loss: 0.0427 - val_accuracy: 0.9758
Epoch 117/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0185 - accuracy:
0.9935 - val_loss: 0.0204 - val_accuracy: 0.9970
Epoch 118/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0166 - accuracy:
0.9948 - val_loss: 0.0159 - val_accuracy: 0.9970
Epoch 119/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0176 - accuracy:
0.9929 - val_loss: 0.0157 - val_accuracy: 0.9970
Epoch 120/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0164 - accuracy:
0.9955 - val_loss: 0.0160 - val_accuracy: 1.0000
Epoch 121/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0166 - accuracy:
0.9955 - val_loss: 0.0174 - val_accuracy: 0.9970
Epoch 122/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0186 - accuracy:
0.9929 - val_loss: 0.0170 - val_accuracy: 0.9970
Epoch 123/150
97/97 [==============================] - 0s 979us/step - loss: 0.0168 -
accuracy: 0.9948 - val_loss: 0.0467 - val_accuracy: 0.9788
Epoch 124/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0164 - accuracy:
0.9955 - val_loss: 0.0324 - val_accuracy: 0.9909
Epoch 125/150
97/97 [==============================] - 0s 989us/step - loss: 0.0154 -
accuracy: 0.9948 - val_loss: 0.0170 - val_accuracy: 0.9970
Epoch 126/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0165 - accuracy:
0.9955 - val_loss: 0.0303 - val_accuracy: 0.9909
Epoch 127/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0167 - accuracy:
0.9955 - val_loss: 0.0313 - val_accuracy: 0.9909
Epoch 128/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0159 - accuracy:
0.9955 - val_loss: 0.0701 - val_accuracy: 0.9667
Epoch 129/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0158 - accuracy:
0.9948 - val_loss: 0.0241 - val_accuracy: 0.9909
Epoch 130/150
97/97 [==============================] - 0s 999us/step - loss: 0.0185 -
accuracy: 0.9935 - val_loss: 0.0161 - val_accuracy: 0.9970
Epoch 131/150
```

```
97/97 [==============================] - 0s 1ms/step - loss: 0.0176 - accuracy:
0.9948 - val_loss: 0.0161 - val_accuracy: 0.9970
Epoch 132/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0186 - accuracy:
0.9968 - val_loss: 0.0345 - val_accuracy: 0.9909
Epoch 133/150
97/97 [==============================] - 0s 979us/step - loss: 0.0161 -
accuracy: 0.9961 - val_loss: 0.0165 - val_accuracy: 0.9970
Epoch 134/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0185 - accuracy:
0.9935 - val_loss: 0.0230 - val_accuracy: 0.9909
Epoch 135/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0142 - accuracy:
0.9955 - val_loss: 0.0306 - val_accuracy: 0.9909
Epoch 136/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0160 - accuracy:
0.9935 - val_loss: 0.0222 - val_accuracy: 0.9939
Epoch 137/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0158 - accuracy:
0.9942 - val_loss: 0.0382 - val_accuracy: 0.9818
Epoch 138/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0155 - accuracy:
0.9935 - val_loss: 0.0244 - val_accuracy: 0.9939
Epoch 139/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0138 - accuracy:
0.9955 - val_loss: 0.0368 - val_accuracy: 0.9818
Epoch 140/150
97/97 [==============================] - 0s 999us/step - loss: 0.0153 -
accuracy: 0.9948 - val_loss: 0.0185 - val_accuracy: 0.9970
Epoch 141/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0145 - accuracy:
0.9955 - val_loss: 0.0143 - val_accuracy: 0.9970
Epoch 142/150
97/97 [==============================] - 0s 979us/step - loss: 0.0134 -
accuracy: 0.9948 - val_loss: 0.0283 - val_accuracy: 0.9909
Epoch 143/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0150 - accuracy:
0.9968 - val_loss: 0.0185 - val_accuracy: 0.9970
Epoch 144/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0151 - accuracy:
0.9942 - val_loss: 0.0166 - val_accuracy: 1.0000
Epoch 145/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0165 - accuracy:
0.9942 - val_loss: 0.0186 - val_accuracy: 0.9970
Epoch 146/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0146 - accuracy:
0.9961 - val_loss: 0.0261 - val_accuracy: 0.9909
Epoch 147/150
```

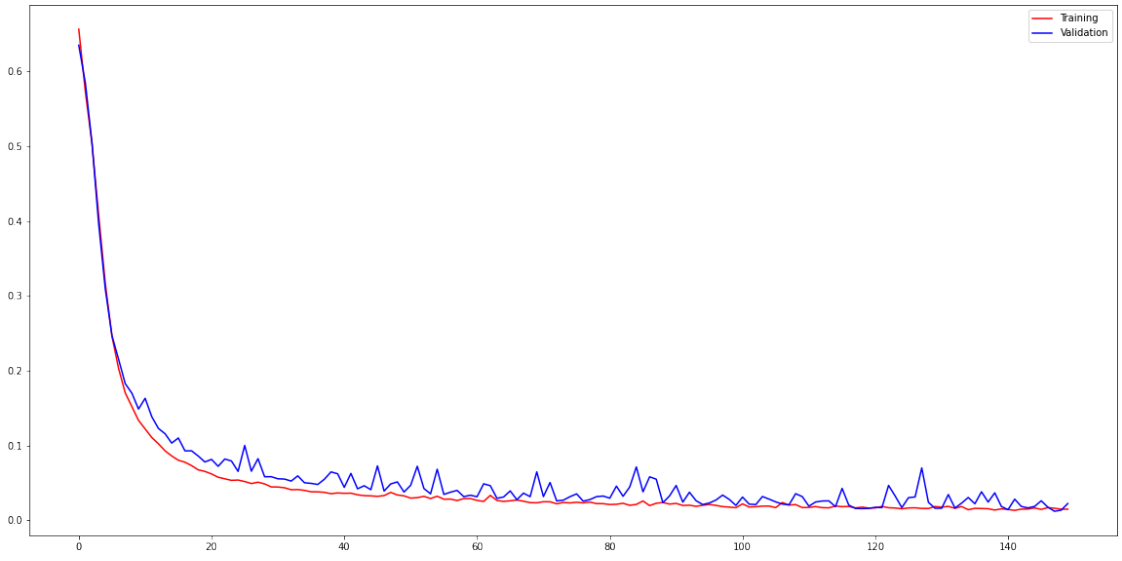
```
97/97 [==============================] - 0s 2ms/step - loss: 0.0170 - accuracy:
0.9929 - val_loss: 0.0176 - val_accuracy: 1.0000
Epoch 148/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0161 - accuracy:
0.9942 - val_loss: 0.0121 - val_accuracy: 0.9970
Epoch 149/150
97/97 [==============================] - 0s 1ms/step - loss: 0.0148 - accuracy:
0.9942 - val_loss: 0.0136 - val_accuracy: 1.0000
Epoch 150/150
97/97 [==============================] - 0s 979us/step - loss: 0.0149 -
accuracy: 0.9948 - val_loss: 0.0225 - val_accuracy: 0.9939
```

[ ]:
```
_,accuracy  = polution_model.evaluate(x_train,y_train)
print("Acuracy: %2f " % (accuracy*100))
```

```
49/49 [==============================] - 0s 644us/step - loss: 0.0115 -
accuracy: 0.9981
Acuracy: 99.805319
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(history.history['loss'], c="r", label ="Training")

plt.plot(history.history['val_loss'], c="b", label ="Validation")

plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x252362811c0>
```



[ ]:
```
predictions = polution_model.predict(x_test)
```

[ ]:
```
for i in range(330):
    print("%s => %d (expected %d)" % (x[i].tolist(),predictions[i],y[i]))
```

```
[42.0, 46.0, 20.0, 7.0, 16.0, 722.7, 0.0, 56.0, 0.163, 26.85, 9.8, 60.0] => 0
(expected 0)
[17.0, 90.0, 32.0, 3.0, 23.0, 718.3, 0.0, 76.0, 0.0, 22.05, 8.4, 126.0] => 0
(expected 0)
[33.0, 184.0, 18.0, 2.0, 18.0, 719.0, 0.0, 49.0, 0.002, 21.82, 9.3, 132.0] => 0
(expected 0)
[12.0, 133.0, 39.0, 6.0, 0.0, 716.8, 0.0, 80.0, 0.0, 24.32, 4.2, 43.0] => 0
(expected 0)
[35.0, 114.0, 17.0, 4.0, 18.0, 715.0, 0.06, 74.0, 0.001, 25.64, 15.0, 239.0] =>
0 (expected 0)
[8.0, 80.0, 35.0, 4.0, 4.0, 719.0, 0.0, 93.0, 0.0, 17.78, 4.4, 354.0] => 0
(expected 0)
[64.0, 61.0, 25.0, 2.0, 13.0, 716.6, 0.0, 44.0, 0.79, 34.04, 10.5, 99.0] => 0
(expected 0)
[36.0, 105.0, 21.0, 3.0, 17.0, 726.8, 0.0, 66.0, 0.036, 15.98, 6.7, 105.0] => 1
(expected 0)
[13.0, 79.0, 34.0, 3.0, 1.0, 719.3, 0.0, 82.0, 0.0, 23.69, 5.4, 65.0] => 0
(expected 0)
[30.0, 51.0, 17.0, 6.0, 22.0, 730.1, 0.0, 50.0, 0.0, 8.48, 7.5, 119.0] => 0
(expected 0)
[28.0, 40.0, 15.0, 1.0, 0.0, 718.6, 0.0, 86.0, 0.0, 19.93, 10.1, 84.0] => 0
(expected 0)
[6.0, 77.0, 16.0, 1.0, 21.0, 716.2, 0.0, 85.0, 0.002, 22.72, 3.5, 12.0] => 0
(expected 0)
[25.0, 46.0, 29.0, 7.0, 23.0, 720.0, 0.0, 70.0, 0.001, 26.66, 12.7, 67.0] => 0
(expected 0)
[32.0, 62.0, 13.0, 1.0, 12.0, 718.8, 0.0, 78.0, 0.439, 22.55, 9.6, 93.0] => 0
(expected 0)
[11.0, 59.0, 13.0, 5.0, 17.0, 727.4, 0.0, 88.0, 0.01, 3.93, 5.1, 122.0] => 0
(expected 0)
[8.0, 87.0, 30.0, 6.0, 22.0, 718.7, 0.0, 68.0, 0.016, 27.53, 11.3, 94.0] => 0
(expected 0)
[14.0, 71.0, 20.0, 4.0, 0.0, 717.0, 0.0, 92.0, 0.026, 23.2, 10.2, 106.0] => 0
(expected 0)
[10.0, 40.0, 37.0, 2.0, 5.0, 718.2, 0.0, 82.0, 0.0, 23.02, 5.0, 274.0] => 0
(expected 0)
[22.0, 113.0, 24.0, 7.0, 23.0, 713.9, 0.0, 79.0, 0.0, 25.7, 7.4, 122.0] => 0
(expected 0)
[7.0, 44.0, 30.0, 3.0, 5.0, 717.9, 0.0, 85.0, 0.0, 23.83, 6.4, 83.0] => 0
(expected 0)
[37.0, 80.0, 23.0, 5.0, 15.0, 719.8, 0.0, 42.0, 0.537, 34.43, 13.5, 110.0] => 0
(expected 0)
[3.0, 113.0, 40.0, 5.0, 5.0, 717.3, 0.0, 80.0, 0.0, 8.66, 0.9, 207.0] => 0
```

```
(expected 0)
[13.0, 69.0, 28.0, 1.0, 19.0, 715.7, 0.0, 77.0, 0.002, 23.63, 6.4, 87.0] => 0
(expected 0)
[18.0, 23.0, 9.0, 2.0, 10.0, 728.7, 0.04, 89.0, 0.043, 6.46, 6.7, 332.0] => 0
(expected 0)
[12.0, 14.0, 14.0, 4.0, 2.0, 716.3, 0.0, 95.0, 0.0, 4.07, 4.1, 13.0] => 0
(expected 0)
[35.0, 68.0, 25.0, 1.0, 15.0, 718.1, 0.0, 60.0, 0.693000019, 30.92, 13.9, 115.0]
=> 1 (expected 0)
[12.0, 36.0, 29.0, 5.0, 1.0, 720.5, 0.0, 82.0, 0.002, 25.13, 9.9, 71.0] => 0
(expected 0)
[8.0, 111.0, 24.0, 4.0, 0.0, 718.4, 0.0, 86.0, 0.019, 20.94, 6.0, 92.0] => 0
(expected 0)
[9.0, 90.0, 25.0, 6.0, 23.0, 716.4, 0.0, 87.0, 0.0, 24.05, 9.1, 111.0] => 0
(expected 0)
[32.0, 62.0, 25.0, 4.0, 17.0, 716.0, 0.0, 50.0, 0.335000008, 32.98, 12.6, 125.0]
=> 0 (expected 0)
[12.0, 61.0, 19.0, 3.0, 11.0, 720.5, 0.01, 89.0, 0.077, 16.1, 6.3, 103.0] => 0
(expected 0)
[23.0, 39.0, 26.0, 6.0, 22.0, 722.9, 0.0, 94.0, 0.0, 15.44, 16.9, 98.0] => 0
(expected 0)
[32.0, 53.0, 18.0, 1.0, 18.0, 708.7, 0.0, 40.0, 0.044, 31.88, 11.9, 119.0] => 0
(expected 0)
[63.0, 47.0, 28.0, 2.0, 13.0, 717.5, 0.0, 39.0, 0.811, 33.88, 9.8, 58.0] => 0
(expected 0)
[16.0, 108.0, 35.0, 3.0, 19.0, 717.9, 0.0, 71.0, 0.0, 25.09, 8.3, 106.0] => 0
(expected 0)
[35.0, 83.0, 23.0, 2.0, 17.0, 718.9, 0.0, 45.0, 0.199, 27.96, 13.5, 122.0] => 0
(expected 0)
[22.0, 35.0, 16.0, 4.0, 22.0, 717.8, 0.0, 42.0, 0.001, 24.2, 4.2, 312.0] => 0
(expected 0)
[38.0, 43.0, 36.0, 1.0, 17.0, 717.2, 0.0, 48.0, 0.055, 30.72, 10.0, 101.0] => 0
(expected 0)
[13.0, 65.0, 10.0, 7.0, 16.0, 719.4, 0.0, 73.0, 0.136, 14.63, 1.6, 197.0] => 0
(expected 0)
[36.0, 115.0, 36.0, 3.0, 10.0, 717.6, 0.0, 71.0, 0.714999974, 29.01, 4.9, 66.0]
=> 0 (expected 0)
[31.0, 63.0, 7.0, 7.0, 10.0, 729.2, 0.0, 57.0, 0.285, 8.74, 7.5, 108.0] => 0
(expected 0)
[11.0, 125.0, 24.0, 5.0, 9.0, 719.7, 0.0, 59.0, 0.374, 16.29, 2.5, 331.0] => 0
(expected 0)
[14.0, 72.0, 24.0, 3.0, 1.0, 717.0, 0.0, 77.0, 0.0, 25.99, 6.6, 76.0] => 0
(expected 0)
[25.0, 63.0, 20.0, 6.0, 9.0, 732.5, 0.0, 69.0, 0.09, 6.38, 11.1, 115.0] => 0
(expected 0)
[36.0, 78.0, 7.0, 5.0, 10.0, 718.4, 0.0, 68.0, 0.687, 30.0, 4.4, 55.0] => 0
(expected 0)
[27.0, 69.0, 38.0, 6.0, 14.0, 718.6, 0.0, 68.0, 0.413, 27.38, 11.2, 104.0] => 0
```

```
(expected 0)
[23.0, 87.0, 26.0, 7.0, 8.0, 721.2, 0.0, 91.0, 0.118, 18.09, 4.3, 15.0] => 0
(expected 0)
[6.0, 55.0, 17.0, 2.0, 6.0, 718.2, 0.0, 86.0, 0.018, 25.38, 4.2, 84.0] => 1
(expected 0)
[38.0, 93.0, 24.0, 1.0, 17.0, 719.6, 0.0, 46.0, 0.224, 29.32, 14.0, 107.0] => 0
(expected 0)
[39.0, 76.0, 6.0, 1.0, 10.0, 712.7, 0.0, 56.0, 0.618, 30.77, 4.9, 337.0] => 0
(expected 0)
[49.0, 92.0, 16.0, 3.0, 13.0, 719.4, 0.0, 39.0, 0.649, 23.63, 12.4, 126.0] => 0
(expected 0)
[14.0, 127.0, 29.0, 2.0, 22.0, 719.1, 0.0, 84.0, 0.0, 23.71, 9.7, 107.0] => 0
(expected 0)
[14.0, 71.0, 25.0, 5.0, 23.0, 716.9, 0.0, 71.0, 0.0, 28.09, 13.0, 104.0] => 0
(expected 0)
[64.0, 46.0, 28.0, 5.0, 12.0, 721.5, 0.0, 42.0, 0.878, 32.06, 6.3, 39.0] => 0
(expected 0)
[9.0, 43.0, 27.0, 5.0, 4.0, 718.0, 0.0, 88.0, 0.0, 23.52, 5.7, 64.0] => 1
(expected 0)
[39.0, 75.0, 27.0, 3.0, 12.0, 717.0, 0.0, 60.0, 0.882000029, 30.96, 9.0, 96.0]
=> 0 (expected 0)
[24.0, 86.0, 27.0, 6.0, 18.0, 718.3, 0.0, 38.0, 0.082000002, 32.99, 18.2, 115.0]
=> 0 (expected 0)
[29.0, 152.0, 36.0, 5.0, 21.0, 721.0, 0.0, 66.0, 0.0, 18.14, 8.1, 104.0] => 0
(expected 0)
[23.0, 72.0, 5.0, 6.0, 14.0, 723.5, 0.0, 78.0, 0.041, 18.99, 10.7, 83.0] => 0
(expected 0)
[36.0, 86.0, 7.0, 1.0, 14.0, 720.9, 0.0, 68.0, 0.297, 15.67, 5.6, 74.0] => 1
(expected 0)
[6.0, 27.0, 16.0, 3.0, 8.0, 725.2, 0.0, 94.0, 0.045, 3.36, 3.7, 29.0] => 0
(expected 0)
[24.0, 79.0, 38.0, 2.0, 17.0, 717.9, 0.0, 63.0, 0.171, 29.23, 13.1, 118.0] => 0
(expected 0)
[6.0, 86.0, 29.0, 1.0, 21.0, 732.3, 0.0, 71.0, 0.0, 10.88, 5.9, 313.0] => 0
(expected 0)
[5.0, 19.0, 15.0, 1.0, 2.0, 724.3, 0.0, 94.0, 0.0, 3.16, 3.9, 335.0] => 0
(expected 0)
[18.0, 47.0, 10.0, 1.0, 22.0, 720.8, 0.0, 69.0, 0.0, 28.13, 15.5, 102.0] => 0
(expected 0)
[32.0, 53.0, 32.0, 5.0, 18.0, 716.2, 0.0, 44.0, 0.090999998, 33.25, 12.6, 127.0]
=> 0 (expected 0)
[4.0, 64.0, 23.0, 1.0, 6.0, 721.1, 0.0, 82.0, 0.003, 14.68, 3.0, 298.0] => 0
(expected 0)
[19.0, 42.0, 7.0, 6.0, 23.0, 719.6, 0.0, 60.0, 0.0, 28.59, 12.8, 90.0] => 0
(expected 0)
[22.0, 47.0, 9.0, 7.0, 18.0, 716.6, 0.0, 45.0, 0.066, 34.23, 14.2, 125.0] => 0
(expected 0)
[20.0, 79.0, 23.0, 3.0, 11.0, 718.9, 0.0, 95.0, 0.128, 17.92, 2.4, 13.0] => 1
```

```
(expected 0)
[32.0, 61.0, 32.0, 4.0, 10.0, 720.7, 0.0, 74.0, 0.649, 23.64, 11.1, 86.0] => 0
(expected 0)
[32.0, 52.0, 7.0, 2.0, 0.0, 732.6, 0.0, 72.0, 0.0, 10.23, 13.3, 119.0] => 0
(expected 0)
[37.0, 97.0, 25.0, 2.0, 14.0, 719.1, 0.0, 70.0, 0.507, 28.79, 12.3, 104.0] => 0
(expected 0)
[13.0, 30.0, 10.0, 7.0, 2.0, 720.9, 0.0, 93.0, 0.0, 22.31, 8.9, 109.0] => 0
(expected 0)
[5.0, 73.0, 13.0, 2.0, 4.0, 718.4, 0.0, 90.0, 0.0, 23.61, 4.7, 33.0] => 0
(expected 0)
[61.0, 84.0, 22.0, 7.0, 14.0, 716.0, 0.0, 47.0, 0.799, 34.56, 10.3, 66.0] => 0
(expected 0)
[58.0, 49.0, 19.0, 6.0, 16.0, 718.9, 0.0, 45.0, 0.446, 27.06, 11.1, 88.0] => 0
(expected 0)
[14.0, 56.0, 5.0, 5.0, 17.0, 720.6, 0.0, 86.0, 0.034, 11.23, 12.4, 360.0] => 0
(expected 0)
[8.0, 54.0, 32.0, 1.0, 7.0, 721.1, 0.0, 96.0, 0.021, 17.79, 2.0, 347.0] => 0
(expected 0)
[11.0, 35.0, 8.0, 5.0, 10.0, 729.1, 0.0, 79.0, 0.242, 7.73, 10.0, 358.0] => 0
(expected 0)
[4.0, 35.0, 12.0, 7.0, 7.0, 718.8, 0.0, 95.0, 0.001, 7.18, 1.5, 22.0] => 0
(expected 0)
[4.0, 124.0, 10.0, 4.0, 8.0, 718.0, 0.0, 59.0, 0.16, 16.18, 2.7, 297.0] => 0
(expected 0)
[19.0, 35.0, 17.0, 4.0, 3.0, 719.5, 0.0, 90.0, 0.001, 21.55, 8.7, 99.0] => 0
(expected 0)
[39.0, 63.0, 30.0, 7.0, 18.0, 714.2, 0.0, 52.0, 0.0, 29.51, 14.3, 99.0] => 0
(expected 0)
[8.0, 50.0, 6.0, 5.0, 7.0, 721.1, 0.0, 82.0, 0.112999998, 25.4, 4.4, 57.0] => 0
(expected 0)
[7.0, 65.0, 17.0, 3.0, 8.0, 730.0, 0.0, 84.0, 0.052, 8.47, 2.8, 251.0] => 0
(expected 0)
[9.0, 22.0, 12.0, 4.0, 4.0, 717.7, 0.0, 94.0, 0.0, 4.68, 7.4, 2.0] => 0
(expected 0)
[6.0, 84.0, 31.0, 3.0, 22.0, 720.7, 0.0, 83.0, 0.0, 18.99, 4.6, 47.0] => 0
(expected 0)
[5.0, 98.0, 28.0, 5.0, 7.0, 723.1, 0.0, 77.0, 0.011, 13.45, 4.4, 301.0] => 0
(expected 0)
[25.0, 99.0, 21.0, 1.0, 10.0, 719.1, 0.0, 75.0, 0.436, 23.18, 3.8, 11.0] => 0
(expected 0)
[26.0, 72.0, 24.0, 7.0, 10.0, 722.5, 0.0, 75.0, 0.537, 25.81, 9.3, 111.0] => 0
(expected 0)
[64.0, 94.0, 20.0, 6.0, 12.0, 717.2, 0.0, 55.0, 0.777, 30.06, 6.6, 121.0] => 0
(expected 0)
[15.0, 172.0, 16.0, 3.0, 22.0, 718.1, 0.0, 54.0, 0.0, 22.21, 10.1, 114.0] => 0
(expected 0)
[28.0, 75.0, 8.0, 7.0, 11.0, 722.4, 0.0, 68.0, 0.856999993, 27.62, 14.0, 111.0]
```

```
=> 0 (expected 0)
[42.0, 81.0, 26.0, 3.0, 13.0, 720.4, 0.0, 54.0, 0.822, 32.5, 10.4, 71.0] => 0
(expected 0)
[11.0, 115.0, 8.0, 4.0, 8.0, 719.8, 0.0, 92.0, 0.12, 18.89, 4.8, 107.0] => 0
(expected 0)
[13.0, 85.0, 20.0, 1.0, 5.0, 717.3, 0.0, 91.0, 0.0, 16.55, 4.6, 111.0] => 0
(expected 0)
[20.0, 115.0, 21.0, 2.0, 21.0, 717.6, 0.0, 56.0, 0.0, 31.0, 17.1, 84.0] => 0
(expected 0)
[9.0, 146.0, 22.0, 5.0, 0.0, 718.1, 0.0, 86.0, 0.0, 20.72, 6.7, 357.0] => 0
(expected 0)
[11.0, 59.0, 22.0, 2.0, 18.0, 717.3, 0.15, 95.0, 0.071, 21.62, 8.2, 115.0] => 0
(expected 0)
[9.0, 55.0, 33.0, 3.0, 2.0, 729.1, 0.0, 77.0, 0.0, 8.34, 3.1, 28.0] => 0
(expected 0)
[3.0, 48.0, 30.0, 3.0, 6.0, 718.5, 0.0, 91.0, 0.02, 21.16, 4.2, 298.0] => 0
(expected 0)
[37.0, 110.0, 20.0, 4.0, 17.0, 721.4, 0.0, 39.0, 0.061, 23.5, 13.1, 116.0] => 0
(expected 0)
[42.0, 134.0, 21.0, 2.0, 16.0, 719.6, 0.0, 45.0, 0.406, 29.62, 14.9, 107.0] => 0
(expected 0)
[7.0, 37.0, 17.0, 7.0, 5.0, 719.1, 0.0, 90.0, 0.066, 23.15, 5.2, 56.0] => 0
(expected 0)
[27.0, 31.0, 29.0, 3.0, 12.0, 721.9, 0.0, 83.0, 0.225, 22.73, 11.9, 87.0] => 0
(expected 0)
[24.0, 16.0, 5.0, 2.0, 7.0, 726.7, 0.0, 80.0, 0.486, 8.57, 15.5, 10.0] => 0
(expected 0)
[16.0, 66.0, 28.0, 7.0, 0.0, 721.1, 0.04, 87.0, 0.0, 24.04, 9.1, 107.0] => 0
(expected 0)
[6.0, 97.0, 11.0, 2.0, 7.0, 719.8, 0.0, 89.0, 0.128000006, 23.77, 4.5, 44.0] =>
0 (expected 0)
[4.0, 85.0, 18.0, 7.0, 4.0, 715.6, 0.0, 90.0, 0.0, 14.05, 2.6, 311.0] => 0
(expected 0)
[9.0, 91.0, 18.0, 5.0, 21.0, 715.9, 0.0, 56.0, 0.002, 31.14, 13.9, 91.0] => 0
(expected 0)
[20.0, 43.0, 5.0, 5.0, 10.0, 732.7, 0.0, 69.0, 0.103, 3.98, 6.2, 0.0] => 0
(expected 0)
[34.0, 61.0, 25.0, 2.0, 21.0, 714.7, 0.0, 34.0, 0.0, 24.71, 4.1, 139.0] => 0
(expected 0)
[22.0, 74.0, 25.0, 4.0, 10.0, 722.4, 0.0, 68.0, 0.709, 29.25, 7.4, 69.0] => 0
(expected 0)
[14.0, 66.0, 27.0, 5.0, 2.0, 717.8, 0.0, 86.0, 0.0, 24.0, 8.9, 101.0] => 1
(expected 0)
[14.0, 111.0, 20.0, 5.0, 22.0, 716.5, 0.0, 65.0, 0.0, 29.31, 14.7, 99.0] => 0
(expected 0)
[20.0, 203.0, 33.0, 4.0, 10.0, 727.8, 0.0, 70.0, 0.495, 11.94, 7.8, 114.0] => 0
(expected 0)
[57.0, 34.0, 11.0, 1.0, 13.0, 718.8, 0.0, 45.0, 0.729, 31.28, 6.4, 71.0] => 0
```

```
(expected 0)
[3.0, 184.0, 40.0, 6.0, 8.0, 721.6, 0.03, 83.0, 0.022, 15.14, 1.6, 244.0] => 0
(expected 0)
[56.0, 51.0, 19.0, 4.0, 14.0, 715.7, 0.0, 24.0, 0.599, 34.07, 7.0, 28.0] => 0
(expected 0)
[49.0, 95.0, 19.0, 7.0, 12.0, 722.7, 0.0, 38.0, 0.765, 24.2, 6.9, 93.0] => 0
(expected 0)
[24.0, 72.0, 6.0, 6.0, 14.0, 724.5, 0.0, 62.0, 0.345, 11.67, 15.3, 2.0] => 0
(expected 0)
[8.0, 81.0, 16.0, 6.0, 10.0, 727.8, 0.0, 90.0, 0.051, 6.33, 5.5, 348.0] => 0
(expected 0)
[52.0, 66.0, 13.0, 6.0, 11.0, 721.2, 0.0, 56.0, 0.782, 23.68, 7.7, 101.0] => 0
(expected 0)
[31.0, 51.0, 12.0, 2.0, 15.0, 719.0, 0.0, 39.0, 0.686, 34.89, 11.8, 66.0] => 0
(expected 0)
[6.0, 73.0, 38.0, 5.0, 8.0, 719.1, 0.0, 95.0, 0.114, 22.06, 5.6, 87.0] => 0
(expected 0)
[4.0, 51.0, 33.0, 1.0, 1.0, 723.0, 0.0, 94.0, 0.0, 8.1, 3.7, 93.0] => 0
(expected 0)
[22.0, 41.0, 14.0, 3.0, 23.0, 719.9, 0.0, 84.0, 0.0, 23.21, 13.5, 119.0] => 0
(expected 0)
[33.0, 43.0, 8.0, 1.0, 16.0, 717.7, 0.0, 50.0, 0.535000026, 33.94, 12.7, 116.0]
=> 0 (expected 0)
[8.0, 69.0, 12.0, 6.0, 8.0, 726.8, 0.0, 92.0, 0.022, 15.59, 5.2, 46.0] => 0
(expected 0)
[13.0, 39.0, 15.0, 1.0, 5.0, 719.1, 0.0, 84.0, 0.0, 23.55, 5.1, 64.0] => 0
(expected 0)
[23.0, 24.0, 14.0, 3.0, 2.0, 721.2, 0.0, 72.0, 0.302, 9.43, 6.0, 335.0] => 0
(expected 0)
[8.0, 137.0, 28.0, 4.0, 21.0, 716.2, 0.0, 87.0, 0.055, 24.01, 3.3, 85.0] => 0
(expected 0)
[9.0, 65.0, 18.0, 7.0, 8.0, 720.4, 0.0, 96.0, 0.207, 20.94, 6.3, 93.0] => 0
(expected 0)
[8.0, 65.0, 26.0, 3.0, 4.0, 719.0, 0.0, 88.0, 0.042, 22.84, 2.4, 48.0] => 0
(expected 0)
[37.0, 63.0, 18.0, 4.0, 14.0, 725.1, 0.0, 55.0, 0.605, 16.39, 11.5, 115.0] => 0
(expected 0)
[5.0, 120.0, 33.0, 2.0, 6.0, 719.2, 0.0, 89.0, 0.006, 23.22, 5.0, 63.0] => 0
(expected 0)
[23.0, 30.0, 8.0, 1.0, 23.0, 719.0, 0.0, 70.0, 0.0, 25.64, 8.1, 72.0] => 0
(expected 0)
[19.0, 143.0, 36.0, 3.0, 21.0, 718.7, 0.0, 72.0, 0.0, 22.98, 14.1, 104.0] => 0
(expected 0)
[81.0, 94.0, 30.0, 3.0, 13.0, 716.7, 0.0, 36.0, 0.697, 28.62, 4.9, 69.0] => 0
(expected 0)
[51.0, 53.0, 15.0, 5.0, 15.0, 718.8, 0.0, 46.0, 0.305, 23.68, 12.1, 126.0] => 0
(expected 0)
[12.0, 129.0, 28.0, 5.0, 21.0, 718.2, 0.0, 85.0, 0.003, 23.97, 7.4, 70.0] => 0
```

```
(expected 0)
[21.0, 70.0, 26.0, 1.0, 0.0, 718.8, 0.0, 76.0, 0.0, 25.39, 8.9, 94.0] => 0
(expected 0)
[17.0, 85.0, 36.0, 5.0, 3.0, 721.0, 0.0, 83.0, 0.0, 23.78, 4.0, 351.0] => 0
(expected 0)
[23.0, 85.0, 35.0, 1.0, 18.0, 718.3, 0.0, 63.0, 0.068, 29.74, 8.8, 93.0] => 0
(expected 0)
[18.0, 99.0, 10.0, 5.0, 1.0, 720.1, 0.0, 85.0, 0.013, 15.86, 15.5, 359.0] => 0
(expected 0)
[7.0, 101.0, 38.0, 7.0, 7.0, 719.9, 0.0, 96.0, 0.116, 20.7, 3.7, 80.0] => 0
(expected 0)
[24.0, 68.0, 23.0, 1.0, 22.0, 719.1, 0.0, 75.0, 0.001, 25.12, 11.4, 69.0] => 0
(expected 0)
[61.0, 45.0, 15.0, 2.0, 16.0, 711.8, 0.0, 27.0, 0.389, 29.25, 7.3, 42.0] => 0
(expected 0)
[45.0, 48.0, 25.0, 1.0, 15.0, 717.5, 0.0, 40.0, 0.545, 33.66, 8.6, 71.0] => 0
(expected 0)
[3.0, 130.0, 22.0, 3.0, 22.0, 714.8, 0.0, 92.0, 0.109, 17.05, 1.1, 329.0] => 0
(expected 0)
[58.0, 61.0, 30.0, 6.0, 17.0, 717.7, 0.0, 27.0, 0.276, 33.39, 12.7, 55.0] => 0
(expected 0)
[12.0, 165.0, 12.0, 4.0, 9.0, 718.3, 0.0, 54.0, 0.325, 18.79, 5.3, 317.0] => 0
(expected 0)
[35.0, 46.0, 21.0, 2.0, 23.0, 719.5, 0.0, 84.0, 0.0, 17.38, 7.8, 103.0] => 0
(expected 0)
[18.0, 30.0, 9.0, 6.0, 3.0, 727.9, 0.0, 93.0, 0.0, 9.48, 5.9, 106.0] => 0
(expected 0)
[17.0, 90.0, 27.0, 4.0, 4.0, 716.8, 0.0, 93.0, 0.0, 17.92, 4.7, 323.0] => 0
(expected 0)
[22.0, 41.0, 11.0, 6.0, 0.0, 725.1, 0.0, 59.0, 0.0, 15.4, 3.4, 9.0] => 0
(expected 0)
[6.0, 61.0, 20.0, 6.0, 20.0, 719.6, 0.0, 74.0, 0.0, 24.34, 5.6, 29.0] => 0
(expected 0)
[15.0, 13.0, 7.0, 1.0, 5.0, 720.0, 0.0, 87.0, 0.082, 21.88, 4.0, 44.0] => 1
(expected 0)
[13.0, 88.0, 38.0, 5.0, 0.0, 717.6, 0.0, 80.0, 0.0, 25.38, 9.9, 87.0] => 0
(expected 0)
[39.0, 58.0, 9.0, 7.0, 10.0, 725.3, 0.0, 68.0, 0.639, 24.75, 7.6, 79.0] => 0
(expected 0)
[14.0, 82.0, 8.0, 5.0, 10.0, 719.3, 0.0, 92.0, 0.317, 24.15, 6.3, 87.0] => 0
(expected 0)
[41.0, 47.0, 8.0, 4.0, 21.0, 720.9, 0.0, 49.0, 0.0, 19.26, 11.1, 124.0] => 0
(expected 0)
[41.0, 69.0, 6.0, 7.0, 19.0, 714.7, 0.0, 53.0, 0.0, 28.71, 13.7, 102.0] => 0
(expected 0)
[9.0, 114.0, 37.0, 7.0, 1.0, 719.5, 0.0, 70.0, 0.0, 26.56, 4.2, 69.0] => 0
(expected 0)
[12.0, 93.0, 30.0, 7.0, 22.0, 721.7, 0.0, 68.0, 0.0, 27.53, 14.1, 102.0] => 0
```

```
(expected 0)
[37.0, 83.0, 22.0, 6.0, 17.0, 717.5, 0.0, 55.0, 0.323, 32.31, 12.4, 109.0] => 0
(expected 0)
[7.0, 112.0, 18.0, 5.0, 8.0, 724.0, 0.0, 96.0, 0.174, 10.68, 4.0, 343.0] => 0
(expected 0)
[28.0, 34.0, 13.0, 2.0, 18.0, 718.3, 0.0, 42.0, 0.001, 24.24, 8.3, 88.0] => 0
(expected 0)
[22.0, 76.0, 22.0, 4.0, 13.0, 719.6, 0.0, 85.0, 0.269, 20.79, 4.1, 18.0] => 0
(expected 0)
[28.0, 66.0, 20.0, 4.0, 12.0, 717.6, 0.0, 47.0, 0.681, 17.52, 6.6, 59.0] => 0
(expected 0)
[9.0, 125.0, 20.0, 2.0, 1.0, 724.3, 0.0, 58.0, 0.0, 13.26, 1.4, 223.0] => 0
(expected 0)
[38.0, 94.0, 26.0, 5.0, 17.0, 714.9, 0.0, 66.0, 0.166, 26.83, 11.4, 120.0] => 0
(expected 0)
[5.0, 71.0, 22.0, 4.0, 6.0, 716.9, 0.0, 94.0, 0.0, 17.73, 5.5, 331.0] => 0
(expected 0)
[3.0, 85.0, 26.0, 4.0, 20.0, 723.9, 0.0, 75.0, 0.0, 15.72, 3.3, 20.0] => 0
(expected 0)
[25.0, 73.0, 10.0, 4.0, 10.0, 720.3, 0.0, 71.0, 0.412999988, 26.59, 5.7, 104.0]
=> 0 (expected 0)
[34.0, 57.0, 8.0, 5.0, 23.0, 719.6, 0.0, 62.0, 0.0, 20.07, 7.1, 96.0] => 0
(expected 0)
[11.0, 94.0, 21.0, 4.0, 23.0, 717.4, 0.04, 87.0, 0.001, 22.36, 5.1, 98.0] => 0
(expected 0)
[46.0, 50.0, 21.0, 7.0, 16.0, 717.4, 0.0, 33.0, 0.216, 35.0, 9.5, 96.0] => 0
(expected 0)
[3.0, 102.0, 32.0, 6.0, 4.0, 715.6, 0.0, 93.0, 0.0, 20.81, 2.5, 313.0] => 0
(expected 0)
[37.0, 78.0, 10.0, 2.0, 9.0, 717.6, 0.0, 62.0, 0.533, 30.27, 4.3, 18.0] => 0
(expected 0)
[4.0, 41.0, 28.0, 1.0, 2.0, 717.7, 0.0, 95.0, 0.0, 19.64, 4.1, 99.0] => 0
(expected 0)
[12.0, 148.0, 16.0, 6.0, 17.0, 715.1, 0.02, 70.0, 0.02, 27.67, 5.1, 103.0] => 0
(expected 0)
[13.0, 57.0, 15.0, 2.0, 4.0, 719.1, 0.0, 88.0, 0.005, 22.2, 4.4, 52.0] => 0
(expected 0)
[16.0, 61.0, 14.0, 5.0, 21.0, 726.6, 0.0, 85.0, 0.0, 4.25, 6.2, 107.0] => 1
(expected 0)
[9.0, 45.0, 8.0, 6.0, 7.0, 717.5, 0.0, 82.0, 0.127, 25.31, 3.6, 59.0] => 0
(expected 0)
[3.0, 173.0, 34.0, 4.0, 8.0, 715.7, 0.0, 92.0, 0.258, 17.23, 4.3, 326.0] => 0
(expected 0)
[36.0, 69.0, 15.0, 3.0, 13.0, 718.4, 0.0, 58.0, 0.787, 29.76, 12.3, 107.0] => 0
(expected 0)
[3.0, 57.0, 12.0, 2.0, 7.0, 716.9, 0.0, 84.0, 0.095, 23.74, 1.4, 284.0] => 0
(expected 0)
[11.0, 191.0, 32.0, 4.0, 20.0, 718.6, 0.0, 87.0, 0.23, 19.71, 5.9, 122.0] => 0
```

```
(expected 0)
[10.0, 62.0, 15.0, 7.0, 20.0, 720.7, 0.0, 50.0, 0.0, 30.38, 10.1, 111.0] => 0
(expected 0)
[26.0, 53.0, 13.0, 7.0, 17.0, 719.9, 0.0, 40.0, 0.363000005, 33.16, 16.0, 119.0]
=> 0 (expected 0)
[29.0, 91.0, 16.0, 2.0, 17.0, 716.9, 0.0, 74.0, 0.115, 24.84, 9.9, 117.0] => 0
(expected 0)
[43.0, 86.0, 5.0, 3.0, 16.0, 717.5, 0.0, 37.0, 0.307, 25.5, 12.8, 112.0] => 0
(expected 0)
[15.0, 60.0, 8.0, 6.0, 8.0, 717.7, 0.0, 79.0, 0.296, 26.48, 4.6, 31.0] => 0
(expected 0)
[14.0, 61.0, 19.0, 1.0, 0.0, 719.2, 0.0, 77.0, 0.0, 25.63, 6.5, 65.0] => 0
(expected 0)
[27.0, 45.0, 22.0, 1.0, 10.0, 719.2, 0.0, 85.0, 0.15, 20.78, 6.1, 72.0] => 0
(expected 0)
[9.0, 96.0, 19.0, 4.0, 5.0, 716.2, 0.0, 94.0, 0.0, 19.12, 3.3, 36.0] => 0
(expected 0)
[22.0, 64.0, 13.0, 1.0, 19.0, 717.2, 0.0, 78.0, 0.0, 23.14, 8.6, 93.0] => 0
(expected 0)
[5.0, 104.0, 31.0, 5.0, 21.0, 728.6, 0.0, 71.0, 0.0, 9.81, 3.0, 321.0] => 1
(expected 0)
[16.0, 99.0, 21.0, 4.0, 19.0, 716.1, 0.0, 58.0, 0.0, 14.92, 10.4, 106.0] => 1
(expected 0)
[8.0, 85.0, 14.0, 7.0, 21.0, 721.2, 0.0, 56.0, 0.0, 29.12, 11.9, 104.0] => 0
(expected 0)
[11.0, 54.0, 25.0, 3.0, 2.0, 716.9, 0.0, 85.0, 0.0, 25.39, 6.7, 68.0] => 0
(expected 0)
[35.0, 50.0, 5.0, 1.0, 0.0, 715.9, 0.0, 73.0, 0.0, 23.76, 11.1, 83.0] => 0
(expected 0)
[20.0, 86.0, 12.0, 1.0, 20.0, 720.9, 0.0, 56.0, 0.0, 25.72, 9.0, 107.0] => 0
(expected 0)
[11.0, 100.0, 13.0, 2.0, 9.0, 734.2, 0.0, 78.0, 0.088, 7.63, 4.5, 84.0] => 0
(expected 0)
[8.0, 67.0, 37.0, 5.0, 18.0, 721.0, 0.0, 93.0, 0.008, 6.7, 3.7, 58.0] => 0
(expected 0)
[32.0, 69.0, 25.0, 1.0, 17.0, 717.2, 0.0, 59.0, 0.326999992, 31.92, 14.0, 117.0]
=> 0 (expected 0)
[38.0, 45.0, 9.0, 3.0, 10.0, 720.5, 0.0, 44.0, 0.676, 19.63, 12.1, 351.0] => 0
(expected 0)
[19.0, 26.0, 7.0, 6.0, 11.0, 724.3, 0.0, 96.0, 0.09, 7.07, 7.9, 107.0] => 0
(expected 0)
[23.0, 68.0, 26.0, 6.0, 19.0, 714.3, 0.0, 39.0, 0.005, 35.15, 10.5, 96.0] => 0
(expected 0)
[58.0, 60.0, 24.0, 6.0, 17.0, 718.8, 0.0, 45.0, 0.268, 26.71, 11.8, 115.0] => 0
(expected 0)
[10.0, 112.0, 37.0, 3.0, 7.0, 720.3, 0.0, 93.0, 0.036, 18.88, 4.6, 73.0] => 0
(expected 0)
[37.0, 84.0, 20.0, 5.0, 15.0, 717.3, 0.0, 52.0, 0.688000023, 32.3, 14.5, 116.0]
```

```
=> 0 (expected 0)
[15.0, 219.0, 18.0, 2.0, 11.0, 722.3, 0.0, 47.0, 0.315, 21.09, 3.2, 105.0] => 0
(expected 0)
[3.0, 48.0, 15.0, 3.0, 1.0, 719.3, 0.0, 62.0, 0.0, 17.46, 3.3, 325.0] => 1
(expected 0)
[21.0, 89.0, 34.0, 5.0, 0.0, 717.4, 0.0, 90.0, 0.0, 17.31, 7.3, 124.0] => 0
(expected 0)
[9.0, 34.0, 5.0, 7.0, 3.0, 719.3, 0.25, 95.0, 0.009, 20.73, 3.7, 339.0] => 0
(expected 0)
[38.0, 98.0, 27.0, 7.0, 10.0, 717.7, 0.0, 69.0, 0.691, 29.51, 7.1, 66.0] => 0
(expected 0)
[4.0, 44.0, 40.0, 2.0, 19.0, 726.4, 0.0, 87.0, 0.0, 4.83, 7.8, 352.0] => 0
(expected 0)
[34.0, 83.0, 36.0, 6.0, 20.0, 716.8, 0.0, 60.0, 0.0, 29.89, 8.7, 73.0] => 0
(expected 0)
[40.0, 20.0, 10.0, 6.0, 3.0, 723.2, 0.08, 94.0, 0.02, 8.44, 12.6, 109.0] => 0
(expected 0)
[11.0, 14.0, 5.0, 5.0, 3.0, 725.3, 0.01, 94.0, 0.0, 4.56, 8.1, 352.0] => 0
(expected 0)
[52.0, 81.0, 10.0, 5.0, 11.0, 722.1, 0.0, 47.0, 0.799, 31.15, 6.2, 59.0] => 0
(expected 0)
[16.0, 113.0, 33.0, 2.0, 0.0, 722.6, 0.0, 81.0, 0.0, 20.45, 8.8, 356.0] => 0
(expected 0)
[6.0, 60.0, 38.0, 7.0, 23.0, 720.5, 0.0, 95.0, 0.015, 18.42, 6.2, 124.0] => 1
(expected 0)
[21.0, 41.0, 28.0, 5.0, 1.0, 725.7, 0.0, 75.0, 0.0, 9.36, 3.9, 55.0] => 0
(expected 0)
[4.0, 111.0, 27.0, 5.0, 6.0, 719.7, 0.0, 90.0, 0.034000002, 21.55, 4.7, 49.0] =>
0 (expected 0)
[28.0, 67.0, 33.0, 6.0, 17.0, 729.7, 0.0, 58.0, 0.061, 8.88, 7.9, 142.0] => 0
(expected 0)
[16.0, 54.0, 9.0, 4.0, 8.0, 719.9, 0.0, 75.0, 0.177000001, 25.5, 5.0, 61.0] => 0
(expected 0)
[11.0, 111.0, 15.0, 5.0, 22.0, 717.5, 0.0, 85.0, 0.03, 25.27, 12.7, 112.0] => 0
(expected 0)
[30.0, 131.0, 30.0, 4.0, 11.0, 727.2, 0.0, 63.0, 0.62, 13.39, 10.2, 120.0] => 0
(expected 0)
[31.0, 74.0, 21.0, 4.0, 17.0, 717.9, 0.0, 73.0, 0.168, 22.86, 4.9, 121.0] => 0
(expected 0)
[14.0, 58.0, 25.0, 2.0, 23.0, 715.7, 0.0, 41.0, 0.0, 23.01, 4.5, 354.0] => 0
(expected 0)
[10.0, 95.0, 34.0, 6.0, 1.0, 718.1, 0.0, 92.0, 0.016, 22.16, 5.5, 85.0] => 0
(expected 0)
[28.0, 69.0, 20.0, 6.0, 15.0, 718.9, 0.0, 40.0, 0.718999982, 33.47, 15.2, 127.0]
=> 0 (expected 0)
[54.0, 45.0, 14.0, 1.0, 14.0, 717.3, 0.0, 35.0, 0.775, 35.25, 7.5, 28.0] => 0
(expected 0)
[32.0, 29.0, 23.0, 3.0, 11.0, 721.3, 0.0, 65.0, 0.827, 30.07, 9.2, 63.0] => 0
```

```
(expected 0)
[48.0, 67.0, 16.0, 2.0, 17.0, 715.4, 0.0, 36.0, 0.226, 34.41, 10.4, 84.0] => 0
(expected 0)
[19.0, 132.0, 32.0, 3.0, 22.0, 718.2, 0.0, 48.0, 0.0, 19.0, 10.6, 105.0] => 0
(expected 0)
[14.0, 57.0, 37.0, 3.0, 23.0, 720.6, 0.0, 82.0, 0.0, 23.43, 2.9, 50.0] => 0
(expected 0)
[9.0, 101.0, 27.0, 6.0, 8.0, 716.4, 0.0, 94.0, 0.03, 23.14, 10.0, 93.0] => 0
(expected 0)
[38.0, 35.0, 10.0, 3.0, 19.0, 720.3, 0.0, 43.0, 0.001, 20.64, 10.7, 20.0] => 0
(expected 0)
[28.0, 86.0, 20.0, 2.0, 22.0, 717.3, 0.0, 54.0, 0.0, 28.9, 11.0, 95.0] => 0
(expected 0)
[27.0, 80.0, 33.0, 7.0, 19.0, 719.0, 0.0, 69.0, 0.011, 28.92, 7.3, 92.0] => 0
(expected 0)
[8.0, 20.0, 16.0, 3.0, 22.0, 727.1, 0.0, 94.0, 0.0, 5.51, 7.9, 101.0] => 0
(expected 0)
[10.0, 94.0, 23.0, 3.0, 22.0, 721.2, 0.0, 73.0, 0.0, 26.23, 8.2, 67.0] => 0
(expected 0)
[3.0, 134.0, 34.0, 6.0, 6.0, 726.1, 0.0, 65.0, 0.001, 14.87, 2.3, 332.0] => 0
(expected 0)
[10.0, 17.0, 16.0, 4.0, 3.0, 716.3, 0.0, 95.0, 0.0, 4.32, 5.6, 356.0] => 0
(expected 0)
[4.0, 101.0, 22.0, 4.0, 1.0, 727.1, 0.0, 86.0, 0.0, 11.99, 3.4, 355.0] => 0
(expected 0)
[31.0, 95.0, 26.0, 6.0, 16.0, 718.8, 0.0, 64.0, 0.338, 26.68, 12.9, 127.0] => 0
(expected 0)
[18.0, 34.0, 12.0, 5.0, 22.0, 723.7, 0.0, 88.0, 0.0, 7.58, 11.7, 3.0] => 0
(expected 0)
[5.0, 84.0, 35.0, 4.0, 3.0, 722.1, 0.0, 95.0, 0.0, 8.36, 1.1, 271.0] => 0
(expected 0)
[3.0, 160.0, 26.0, 6.0, 7.0, 719.7, 0.0, 96.0, 0.042, 19.36, 2.3, 321.0] => 0
(expected 0)
[54.0, 75.0, 39.0, 5.0, 16.0, 719.6, 0.0, 52.0, 0.355, 22.34, 9.8, 119.0] => 0
(expected 0)
[18.0, 108.0, 30.0, 5.0, 18.0, 717.2, 0.0, 68.0, 0.009, 28.23, 6.1, 118.0] => 0
(expected 0)
[16.0, 101.0, 9.0, 1.0, 3.0, 717.9, 0.0, 81.0, 0.0, 24.1, 5.3, 307.0] => 0
(expected 0)
[7.0, 80.0, 33.0, 7.0, 0.0, 720.7, 0.12, 95.0, 0.0, 19.36, 5.2, 335.0] => 0
(expected 0)
[92.0, 89.0, 29.0, 3.0, 15.0, 715.5, 0.0, 34.0, 0.458, 29.58, 8.4, 93.0] => 1
(expected 0)
[12.0, 99.0, 23.0, 6.0, 23.0, 716.5, 0.0, 70.0, 0.001, 28.29, 11.2, 67.0] => 0
(expected 0)
[29.0, 131.0, 23.0, 7.0, 18.0, 715.5, 0.0, 48.0, 0.007, 26.34, 9.9, 132.0] => 0
(expected 0)
[50.0, 93.0, 5.0, 5.0, 14.0, 717.3, 0.0, 68.0, 0.395, 29.05, 3.9, 110.0] => 0
```

```
(expected 0)
[22.0, 68.0, 28.0, 4.0, 18.0, 714.9, 0.0, 71.0, 0.041, 26.15, 8.2, 119.0] => 0
(expected 0)
[14.0, 36.0, 9.0, 3.0, 6.0, 716.4, 0.01, 96.0, 0.108, 19.98, 10.3, 86.0] => 0
(expected 0)
[15.0, 53.0, 17.0, 5.0, 3.0, 725.3, 0.0, 82.0, 0.001, 8.04, 3.6, 12.0] => 0
(expected 0)
[6.0, 73.0, 19.0, 6.0, 5.0, 720.7, 0.0, 78.0, 0.0, 24.18, 3.3, 25.0] => 0
(expected 0)
[36.0, 78.0, 22.0, 4.0, 12.0, 716.1, 0.0, 77.0, 0.573, 26.35, 4.2, 57.0] => 0
(expected 0)
[25.0, 49.0, 27.0, 2.0, 18.0, 716.0, 0.0, 45.0, 0.077, 33.97, 12.7, 121.0] => 0
(expected 0)
[14.0, 73.0, 15.0, 2.0, 1.0, 724.7, 0.0, 92.0, 0.0, 11.28, 5.0, 94.0] => 0
(expected 0)
[26.0, 87.0, 22.0, 6.0, 9.0, 720.9, 0.0, 76.0, 0.310000002, 25.31, 4.8, 12.0] =>
0 (expected 0)
[12.0, 82.0, 36.0, 4.0, 5.0, 716.6, 0.0, 93.0, 0.0, 17.9, 5.3, 322.0] => 0
(expected 0)
[38.0, 48.0, 15.0, 7.0, 16.0, 718.1, 0.0, 64.0, 0.402, 25.17, 12.1, 101.0] => 0
(expected 0)
[59.0, 105.0, 32.0, 2.0, 14.0, 717.5, 0.0, 54.0, 0.813000023, 32.7, 10.7, 104.0]
=> 0 (expected 0)
[39.0, 112.0, 19.0, 4.0, 15.0, 716.0, 0.0, 69.0, 0.517, 25.66, 13.1, 126.0] => 0
(expected 0)
[24.0, 40.0, 25.0, 4.0, 11.0, 716.8, 0.0, 82.0, 0.493, 24.89, 5.1, 13.0] => 0
(expected 0)
[6.0, 21.0, 8.0, 3.0, 20.0, 726.9, 0.0, 94.0, 0.0, 5.82, 8.5, 108.0] => 0
(expected 0)
[25.0, 128.0, 25.0, 4.0, 18.0, 717.0, 0.0, 44.0, 0.101000004, 31.34, 16.4,
120.0] => 0 (expected 0)
[20.0, 27.0, 6.0, 2.0, 8.0, 718.1, 0.01, 96.0, 0.035, 19.7, 10.7, 87.0] => 0
(expected 0)
[74.0, 92.0, 17.0, 6.0, 17.0, 722.5, 0.0, 26.0, 0.233, 24.88, 10.7, 117.0] => 0
(expected 0)
[21.0, 79.0, 18.0, 3.0, 20.0, 716.6, 0.0, 36.0, 0.0, 33.26, 11.1, 128.0] => 0
(expected 0)
[10.0, 116.0, 25.0, 2.0, 1.0, 721.5, 0.0, 85.0, 0.0, 19.86, 1.1, 182.0] => 0
(expected 0)
[32.0, 86.0, 31.0, 5.0, 17.0, 716.5, 0.0, 48.0, 0.331, 33.23, 12.8, 123.0] => 0
(expected 0)
[6.0, 66.0, 38.0, 6.0, 9.0, 727.0, 0.0, 88.0, 0.034, 6.81, 4.3, 323.0] => 0
(expected 0)
[38.0, 92.0, 13.0, 2.0, 16.0, 715.0, 0.0, 64.0, 0.142, 21.83, 12.2, 105.0] => 0
(expected 0)
[22.0, 31.0, 19.0, 7.0, 4.0, 728.5, 0.0, 62.0, 0.0, 6.77, 4.4, 44.0] => 0
(expected 0)
[6.0, 187.0, 37.0, 4.0, 2.0, 723.0, 0.0, 54.0, 0.001, 15.41, 1.5, 304.0] => 0
```

```
(expected 0)
[22.0, 43.0, 10.0, 2.0, 1.0, 718.7, 0.0, 75.0, 0.0, 24.11, 4.7, 43.0] => 0
(expected 0)
[3.0, 95.0, 35.0, 4.0, 0.0, 717.3, 0.0, 68.0, 0.0, 20.63, 2.1, 5.0] => 0
(expected 0)
[18.0, 90.0, 9.0, 3.0, 3.0, 719.5, 0.0, 94.0, 0.0, 18.98, 6.1, 82.0] => 0
(expected 0)
[17.0, 49.0, 25.0, 2.0, 9.0, 718.1, 0.0, 83.0, 0.121, 22.34, 6.1, 341.0] => 0
(expected 0)
[27.0, 121.0, 9.0, 3.0, 18.0, 727.0, 0.0, 66.0, 0.0, 15.22, 5.5, 101.0] => 0
(expected 0)
[12.0, 208.0, 32.0, 3.0, 9.0, 725.5, 0.0, 44.0, 0.277, 19.45, 2.7, 319.0] => 0
(expected 0)
[8.0, 33.0, 22.0, 6.0, 18.0, 722.8, 0.01, 95.0, 0.002, 6.22, 8.1, 108.0] => 0
(expected 0)
[18.0, 123.0, 36.0, 3.0, 19.0, 718.0, 0.0, 72.0, 0.0, 22.77, 8.1, 102.0] => 0
(expected 0)
[6.0, 50.0, 25.0, 2.0, 18.0, 726.4, 0.0, 84.0, 0.124, 5.68, 7.6, 348.0] => 0
(expected 0)
[5.0, 109.0, 24.0, 3.0, 6.0, 717.7, 0.0, 96.0, 0.001, 19.22, 4.9, 103.0] => 0
(expected 0)
[22.0, 88.0, 18.0, 2.0, 20.0, 719.3, 0.0, 47.0, 0.0, 22.23, 8.2, 97.0] => 0
(expected 0)
[22.0, 103.0, 19.0, 6.0, 19.0, 719.0, 0.0, 76.0, 0.0, 24.06, 9.1, 114.0] => 0
(expected 0)
[20.0, 67.0, 34.0, 5.0, 13.0, 719.2, 0.0, 80.0, 0.069, 15.09, 13.9, 359.0] => 0
(expected 0)
[14.0, 46.0, 12.0, 5.0, 20.0, 728.1, 0.0, 92.0, 0.0, 10.17, 10.2, 120.0] => 0
(expected 0)
[9.0, 92.0, 15.0, 5.0, 7.0, 718.2, 0.0, 84.0, 0.062, 24.78, 5.2, 64.0] => 0
(expected 0)
[14.0, 54.0, 11.0, 3.0, 14.0, 719.5, 0.02, 94.0, 0.073, 16.02, 7.4, 130.0] => 0
(expected 0)
[25.0, 35.0, 8.0, 4.0, 23.0, 721.8, 0.0, 68.0, 0.0, 27.0, 11.7, 86.0] => 0
(expected 0)
[17.0, 47.0, 32.0, 5.0, 9.0, 720.6, 0.0, 78.0, 0.238999993, 24.25, 4.8, 78.0] =>
0 (expected 0)
[42.0, 20.0, 6.0, 5.0, 0.0, 720.6, 0.0, 51.0, 0.0, 18.21, 6.3, 115.0] => 0
(expected 0)
[9.0, 86.0, 16.0, 6.0, 3.0, 719.4, 0.0, 83.0, 0.0, 14.52, 0.9, 319.0] => 0
(expected 0)
[45.0, 63.0, 6.0, 6.0, 11.0, 726.0, 0.0, 40.0, 0.68, 21.68, 5.0, 42.0] => 0
(expected 0)
[14.0, 125.0, 7.0, 3.0, 22.0, 718.5, 0.0, 76.0, 0.0, 22.06, 7.8, 128.0] => 0
(expected 0)
[36.0, 68.0, 14.0, 2.0, 14.0, 730.9, 0.0, 63.0, 0.278, 12.6, 8.1, 100.0] => 0
(expected 0)
[51.0, 22.0, 8.0, 1.0, 12.0, 718.6, 0.0, 40.0, 0.89, 33.42, 6.3, 324.0] => 0
```

```
(expected 0)
[46.0, 46.0, 16.0, 4.0, 19.0, 719.9, 0.0, 49.0, 0.001, 20.67, 12.9, 121.0] => 0
(expected 0)
[28.0, 31.0, 18.0, 5.0, 3.0, 734.1, 0.0, 74.0, 0.034, 6.08, 10.5, 6.0] => 0
(expected 0)
[23.0, 85.0, 30.0, 7.0, 7.0, 720.9, 0.0, 93.0, 0.026, 17.27, 3.3, 85.0] => 0
(expected 0)
[38.0, 59.0, 9.0, 5.0, 12.0, 722.1, 0.0, 60.0, 0.735, 20.86, 4.0, 12.0] => 0
(expected 0)
[16.0, 66.0, 18.0, 2.0, 9.0, 720.4, 0.0, 80.0, 0.279, 24.87, 4.6, 37.0] => 0
(expected 0)
[13.0, 45.0, 11.0, 1.0, 2.0, 719.1, 0.0, 84.0, 0.0, 23.61, 4.2, 340.0] => 0
(expected 0)
[21.0, 91.0, 23.0, 5.0, 19.0, 714.6, 0.0, 35.0, 0.006, 35.23, 12.8, 133.0] => 0
(expected 0)
[8.0, 77.0, 5.0, 1.0, 5.0, 721.8, 0.0, 96.0, 0.0, 14.09, 1.3, 350.0] => 0
(expected 0)
[28.0, 70.0, 35.0, 3.0, 13.0, 717.6, 0.0, 74.0, 0.598, 28.34, 11.2, 123.0] => 1
(expected 0)
[5.0, 135.0, 7.0, 7.0, 8.0, 716.5, 0.0, 90.0, 0.093, 16.29, 2.3, 17.0] => 0
(expected 0)
[11.0, 51.0, 29.0, 1.0, 22.0, 718.5, 0.0, 68.0, 0.0, 28.5, 13.4, 103.0] => 0
(expected 0)
[20.0, 78.0, 11.0, 3.0, 18.0, 720.4, 0.0, 85.0, 0.009, 19.17, 8.7, 118.0] => 0
(expected 0)
[30.0, 74.0, 23.0, 5.0, 10.0, 718.8, 0.0, 73.0, 0.507000029, 27.18, 5.4, 57.0]
=> 0 (expected 0)
[28.0, 146.0, 40.0, 5.0, 16.0, 716.6, 0.0, 72.0, 0.219, 23.5, 10.8, 111.0] => 0
(expected 0)
[18.0, 81.0, 19.0, 4.0, 9.0, 715.2, 0.0, 69.0, 0.545, 28.93, 4.6, 349.0] => 0
(expected 0)
[5.0, 98.0, 38.0, 7.0, 5.0, 717.7, 0.0, 91.0, 0.0, 23.5, 2.8, 17.0] => 0
(expected 0)
[6.0, 171.0, 32.0, 4.0, 23.0, 722.4, 0.0, 63.0, 0.0, 16.46, 3.4, 74.0] => 0
(expected 0)
[12.0, 26.0, 19.0, 6.0, 20.0, 729.8, 0.0, 88.0, 0.0, 5.23, 9.1, 355.0] => 0
(expected 0)
[20.0, 87.0, 28.0, 6.0, 20.0, 719.6, 0.0, 78.0, 0.0, 23.7, 8.7, 122.0] => 0
(expected 0)
[6.0, 77.0, 25.0, 2.0, 6.0, 726.2, 0.0, 51.0, 0.0, 13.11, 1.1, 257.0] => 0
(expected 0)
```

1.31 Reclassifying the data with min and max values

[ ]:
```
set_printoptions(precision=6,suppress=True)

# define a matrix
```

```
df_2 = pd.read_csv('Monterrey Pollution Data w Days.csv',sep=",",encoding =␣
 ↪'unicode_escape')

header_row=df_2.columns.values
```

[ ]:
```
## Group data by date##
pm25_max=df_2.
 ↪groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV','PM2.5'].max()
```

```
C:\Users\Usuario\AppData\Local\Temp/ipykernel_46288/2376389202.py:2:
FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of
keys) will be deprecated, use a list instead.
  pm25_max=df_2.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV','P
M2.5'].max()
```

[ ]:
```
pm25_max=pm25_max.rename(columns={'PRS':'PRS_max','RAINF':'RAINF_max',\
                          'RH':'RH_max','SR':'SR_max','TOUT':'TOUT_max','WSR':
 ↪'WSR_max',\
                          'WDV':'WDV_max','PM2.5':'PM2.5_max'})
```

[ ]:
```
pm25_max.head()
```

[ ]:
```
           PRS_max  RAINF_max  RH_max  SR_max  TOUT_max  WSR_max  WDV_max  \
Date
01-abr-15    718.1       0.00      94   0.641     26.04     14.5      352
01-ene-15    725.4       0.02      96   0.175      7.43      8.2      354
01-feb-15    722.0       0.01      95   0.277     24.59      5.8      357
01-jul-15    720.4       0.03      92   0.413     26.59     14.2      125
01-mar-15    725.1       0.01      96   0.214     12.85      7.6      284

           PM2.5_max
Date
01-abr-15         68
01-ene-15         80
01-feb-15         56
01-jul-15         40
01-mar-15         70
```

[ ]:
```
pm25_min=df_2.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV'].
 ↪min()
```

```
C:\Users\Usuario\AppData\Local\Temp/ipykernel_46288/3024540620.py:1:
FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of
keys) will be deprecated, use a list instead.
  pm25_min=df_2.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV'].m
in()
```

[ ]:
```
pm25_min=pm25_min.rename(columns={'PRS':'PRS_min','RAINF':'RAINF_min',\
                          'RH':'RH_min','SR':'SR_min','TOUT':'TOUT_min','WSR':
 ↪'WSR_min',\
                          'WDV':'WDV_min'})
```

[ ]:
```
pm25_min.head()
```

[ ]:

Date	PRS_min	RAINF_min	RH_min	SR_min	TOUT_min	WSR_min	WDV_min
01-abr-15	715.7	0.0	67	0.0	18.92	1.2	21
01-ene-15	720.6	0.0	93	0.0	3.79	2.9	2
01-feb-15	716.9	0.0	55	0.0	13.84	1.4	27
01-jul-15	718.8	0.0	71	0.0	21.35	5.0	31
01-mar-15	722.3	0.0	87	0.0	8.03	1.2	26

[ ]:
```
pm25 = pm25_max.join(pm25_min)
pm25.head()
```

[ ]:

Date	PRS_max	RAINF_max	RH_max	SR_max	TOUT_max	WSR_max	WDV_max	\
01-abr-15	718.1	0.00	94	0.641	26.04	14.5	352	
01-ene-15	725.4	0.02	96	0.175	7.43	8.2	354	
01-feb-15	722.0	0.01	95	0.277	24.59	5.8	357	
01-jul-15	720.4	0.03	92	0.413	26.59	14.2	125	
01-mar-15	725.1	0.01	96	0.214	12.85	7.6	284	

Date	PM2.5_max	PRS_min	RAINF_min	RH_min	SR_min	TOUT_min	WSR_min	\
01-abr-15	68	715.7	0.0	67	0.0	18.92	1.2	
01-ene-15	80	720.6	0.0	93	0.0	3.79	2.9	
01-feb-15	56	716.9	0.0	55	0.0	13.84	1.4	
01-jul-15	40	718.8	0.0	71	0.0	21.35	5.0	
01-mar-15	70	722.3	0.0	87	0.0	8.03	1.2	

Date	WDV_min
01-abr-15	21
01-ene-15	2
01-feb-15	27
01-jul-15	31
01-mar-15	26

[ ]:
```
pm25_avg=df_2.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV'].
 ↪mean()
```

```
C:\Users\Usuario\AppData\Local\Temp/ipykernel_46288/2988305909.py:1:
FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of
keys) will be deprecated, use a list instead.
```

```
pm25_avg=df_2.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV'].m
ean()
```

[ ]:
```
pm25_avg=pm25_avg.rename(columns={'PRS':'PRS_avg','RAINF':'RAINF_avg',\
                        'RH':'RH_avg','SR':'SR_avg','TOUT':'TOUT_avg','WSR':
↪'WSR_avg',\
                        'WDV':'WDV_avg'})
```

[ ]:
```
pm25 = pm25.join(pm25_avg)
pm25.head()
```

[ ]:
```
            PRS_max  RAINF_max  RH_max  SR_max  TOUT_max  WSR_max  WDV_max  \
Date
01-abr-15     718.1       0.00      94   0.641     26.04     14.5      352
01-ene-15     725.4       0.02      96   0.175      7.43      8.2      354
01-feb-15     722.0       0.01      95   0.277     24.59      5.8      357
01-jul-15     720.4       0.03      92   0.413     26.59     14.2      125
01-mar-15     725.1       0.01      96   0.214     12.85      7.6      284

            PM2.5_max  PRS_min  RAINF_min  …  TOUT_min  WSR_min  WDV_min  \
Date                                        …
01-abr-15          68    715.7        0.0  …     18.92      1.2       21
01-ene-15          80    720.6        0.0  …      3.79      2.9        2
01-feb-15          56    716.9        0.0  …     13.84      1.4       27
01-jul-15          40    718.8        0.0  …     21.35      5.0       31
01-mar-15          70    722.3        0.0  …      8.03      1.2       26

               PRS_avg  RAINF_avg     RH_avg    SR_avg   TOUT_avg   WSR_avg  \
Date
01-abr-15  716.900000   0.000000  77.850000  0.172650  22.892500  8.030000
01-ene-15  722.458824   0.002353  94.764706  0.044176   5.553529  5.300000
01-feb-15  718.369565   0.001739  82.478261  0.070435  17.874348  3.691304
01-jul-15  719.717647   0.001765  82.176471  0.118941  23.507647  8.705882
01-mar-15  723.317647   0.000588  91.352941  0.051824  10.544706  4.082353

               WDV_avg
Date
01-abr-15   99.900000
01-ene-15  145.705882
01-feb-15  263.000000
01-jul-15   92.294118
01-mar-15  105.294118

[5 rows x 22 columns]
```

[ ]:
```
pm25['CONT']=(pm25['PM2.5_max']>40.4)*1

pm25.head()
```

[ ]:
```
            PRS_max  RAINF_max  RH_max  SR_max  TOUT_max  WSR_max  WDV_max  \
Date
01-abr-15     718.1       0.00      94   0.641     26.04     14.5      352
01-ene-15     725.4       0.02      96   0.175      7.43      8.2      354
01-feb-15     722.0       0.01      95   0.277     24.59      5.8      357
01-jul-15     720.4       0.03      92   0.413     26.59     14.2      125
01-mar-15     725.1       0.01      96   0.214     12.85      7.6      284

            PM2.5_max  PRS_min  RAINF_min  …  WSR_min  WDV_min     PRS_avg  \
Date                                       …
01-abr-15          68    715.7        0.0  …      1.2       21  716.900000
01-ene-15          80    720.6        0.0  …      2.9        2  722.458824
01-feb-15          56    716.9        0.0  …      1.4       27  718.369565
01-jul-15          40    718.8        0.0  …      5.0       31  719.717647
01-mar-15          70    722.3        0.0  …      1.2       26  723.317647

            RAINF_avg     RH_avg    SR_avg   TOUT_avg   WSR_avg     WDV_avg  \
Date
01-abr-15    0.000000  77.850000  0.172650  22.892500  8.030000   99.900000
01-ene-15    0.002353  94.764706  0.044176   5.553529  5.300000  145.705882
01-feb-15    0.001739  82.478261  0.070435  17.874348  3.691304  263.000000
01-jul-15    0.001765  82.176471  0.118941  23.507647  8.705882   92.294118
01-mar-15    0.000588  91.352941  0.051824  10.544706  4.082353  105.294118

            CONT
Date
01-abr-15      1
01-ene-15      1
01-feb-15      1
01-jul-15      0
01-mar-15      1

[5 rows x 23 columns]
```

Now that the data is limited to the minimum maximum and average we will do another classification model

[ ]:
```
column_names = pm25.columns
column_names = column_names.drop(["CONT"])

x  = pm25[column_names]
```

[ ]:
```
#Dependant
x = x
x.head()
```

[ ]:
```
            PRS_max  RAINF_max  RH_max  SR_max  TOUT_max  WSR_max  WDV_max  \
Date
```

```
01-abr-15    718.1       0.00      94   0.641     26.04     14.5      352
01-ene-15    725.4       0.02      96   0.175      7.43      8.2      354
01-feb-15    722.0       0.01      95   0.277     24.59      5.8      357
01-jul-15    720.4       0.03      92   0.413     26.59     14.2      125
01-mar-15    725.1       0.01      96   0.214     12.85      7.6      284

            PM2.5_max  PRS_min  RAINF_min  …  TOUT_min  WSR_min  WDV_min  \
Date                                       …
01-abr-15          68    715.7        0.0  …     18.92      1.2       21
01-ene-15          80    720.6        0.0  …      3.79      2.9        2
01-feb-15          56    716.9        0.0  …     13.84      1.4       27
01-jul-15          40    718.8        0.0  …     21.35      5.0       31
01-mar-15          70    722.3        0.0  …      8.03      1.2       26

               PRS_avg  RAINF_avg     RH_avg    SR_avg   TOUT_avg   WSR_avg  \
Date
01-abr-15  716.900000   0.000000  77.850000  0.172650  22.892500  8.030000
01-ene-15  722.458824   0.002353  94.764706  0.044176   5.553529  5.300000
01-feb-15  718.369565   0.001739  82.478261  0.070435  17.874348  3.691304
01-jul-15  719.717647   0.001765  82.176471  0.118941  23.507647  8.705882
01-mar-15  723.317647   0.000588  91.352941  0.051824  10.544706  4.082353

               WDV_avg
Date
01-abr-15   99.900000
01-ene-15  145.705882
01-feb-15  263.000000
01-jul-15   92.294118
01-mar-15  105.294118

[5 rows x 22 columns]
```

[ ]:
```
#Independant#
y = pm25['CONT']
y.head()
```

[ ]:
```
Date
01-abr-15    1
01-ene-15    1
01-feb-15    1
01-jul-15    0
01-mar-15    1
Name: CONT, dtype: int32
```

[ ]:
```
#Separate train and test data
x_train,x_test,y_train,y_test = sk.model_selection.
 ↪train_test_split(x,y,train_size=0.20,random_state=0)
```

```
#Standarize data. Subtract mean and devide by standard deviation
sc = sk.preprocessing.StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

[ ]:
```
from sklearn.naive_bayes import GaussianNB
#Fit classifier with train data
NB = GaussianNB()
NB.fit(x_train,y_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
#Predict test data
y_pred=NB.predict(x_test)
```

[ ]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
tn, fp, fn, tp = cm.ravel()
```

```
[[ 24  27]
 [ 12 121]]
```

[ ]:
```
cm2 = pd.crosstab(y_test,y_pred,rownames=['Actual'],colnames=['Predicted'])
sns.heatmap(cm2,annot=True)
```

[ ]:
```
<AxesSubplot:xlabel='Predicted', ylabel='Actual'>
```

[ ]:
```
#Get performance measures
from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.67      0.47      0.55        51
           1       0.82      0.91      0.86       133

    accuracy                           0.79       184
   macro avg       0.74      0.69      0.71       184
weighted avg       0.78      0.79      0.78       184
```

[ ]:
```
recall=tp/(tp+fn)
print("Likelihood Ratio",recall/(1-recall))
print("Accuracy=",(tp+tn)/(tp+tn+fp+fn))
```

```
Likelihood Ratio 10.083333333333334
Accuracy= 0.7880434782608695
```

2 Datos Sima

In this case we will repeat the steps from above but from a new dataset

[ ]:
```
df = pd.read_csv("Datos SIMA 2015-Noroeste.csv",sep=",",encoding =␣
↪'unicode_escape')
column_names = df.columns
df = df.drop(['Date'],axis=1)
df.head()
```

[ ]:

	O3	PM10	PM2.5	PRS	RAINF	RH	SR	TOUT	WSR	WDV
0	4	199	169	717.9	0.0	91	0.0	5.17	3.1	247
1	4	107	60	717.6	0.0	92	0.0	4.93	2.2	220
2	4	51	26	717.0	0.0	92	0.0	4.73	2.7	255
3	4	33	17	716.7	0.0	92	0.0	4.73	3.7	260
4	4	30	17	716.7	0.0	92	0.0	4.41	5.5	277

[ ]:
```
## Describing the data ##
df.describe()
```

[ ]:

	O3	PM10	PM2.5	PRS	RAINF	\
count	4568.000000	4568.000000	4568.000000	4568.000000	4568.000000	
mean	21.918126	78.512478	26.525394	713.659173	0.001627	
std	19.799716	56.266545	17.691460	3.865896	0.018431	
min	1.000000	5.000000	5.000000	701.200000	0.000000	
25%	5.000000	42.000000	15.000000	711.200000	0.000000	
50%	16.000000	63.000000	23.000000	713.000000	0.000000	
75%	33.000000	97.000000	33.000000	716.100000	0.000000	
max	118.000000	593.000000	263.000000	726.900000	0.600000	

	RH	SR	TOUT	WSR	WDV

```
count  4568.000000  4568.000000  4568.000000  4568.000000  4568.000000
mean     66.011602     0.169129    21.012585     8.282903   177.051664
std      18.935894     0.252062     7.698031     5.418112    84.824669
min       3.000000     0.000000     3.460000     0.900000     2.000000
25%      52.000000     0.000000    15.180000     4.200000   108.000000
50%      68.000000     0.011000    21.750000     6.700000   142.000000
75%      82.000000     0.272250    26.672500    11.500000   264.000000
max      97.000000     0.918000    36.960000    33.200000   358.000000
```

[ ]:
```
O3 = df ["O3"]
PM_25 = df["PM2.5"]
PM_10 = df["PM10"]

column_names = df.columns
column_names = column_names.drop(["O3","PM2.5","PM10"])

x  = df[column_names]
x.head()
```

[ ]:
```
     PRS  RAINF  RH   SR  TOUT  WSR  WDV
0  717.9    0.0  91  0.0  5.17  3.1  247
1  717.6    0.0  92  0.0  4.93  2.2  220
2  717.0    0.0  92  0.0  4.73  2.7  255
3  716.7    0.0  92  0.0  4.73  3.7  260
4  716.7    0.0  92  0.0  4.41  5.5  277
```

[ ]:
```
#Only O3 Dataframe
O3_df = df.drop(["PM10","PM2.5"],axis=1)
header_row=O3_df.columns.values

y=O3_df["O3"]
y.head()
```

[ ]:
```
0    4
1    4
2    4
3    4
4    4
Name: O3, dtype: int64
```

[ ]:
```
M = mean(O3_df.T, axis=1)
print(M)

#Std Deviation
st = std(O3_df.T, axis=1,ddof=1)

#Normalize data
```

```
Xdat = (O3_df - M)/st

print(st)
print(Xdat)
```

```
O3        21.918126
PRS      713.659173
RAINF      0.001627
RH        66.011602
SR         0.169129
TOUT      21.012585
WSR        8.282903
WDV      177.051664
dtype: float64
O3       19.799716
PRS       3.865896
RAINF     0.018431
RH       18.935894
SR        0.252062
TOUT      7.698031
WSR       5.418112
WDV      84.824669
dtype: float64
            O3       PRS    RAINF        RH        SR      TOUT       WSR  \
0    -0.904969  1.096984 -0.08825  1.319631 -0.670983 -2.058005 -0.956588
1    -0.904969  1.019383 -0.08825  1.372441 -0.670983 -2.089182 -1.122698
2    -0.904969  0.864179 -0.08825  1.372441 -0.670983 -2.115162 -1.030415
3    -0.904969  0.786578 -0.08825  1.372441 -0.670983 -2.115162 -0.845849
4    -0.904969  0.786578 -0.08825  1.372441 -0.670983 -2.156731 -0.513630
…         …       …       …        …        …       …        …
4563 -0.955475  1.045250 -0.08825  1.319631 -0.670983 -1.195447 -0.365977
4564 -0.955475  1.148719 -0.08825  1.266822 -0.670983 -1.248707 -0.365977
4565 -0.955475  1.252188 -0.08825  1.161202 -0.670983 -1.300668 -0.236780
4566 -0.803957  1.562595 -0.08825  0.897153 -0.670983 -1.311061 -0.107584
4567 -0.702946  1.510860 -0.08825  1.108392 -0.670983 -1.463048 -0.476716

          WDV
0     0.824623
1     0.506319
2     0.918935
3     0.977880
4     1.178293
…         …
4563  0.730310
4564  0.305906
4565  0.753888
4566  1.213660
4567  1.260816
```

```
[4568 rows x 8 columns]
```

[ ]:
```
# create the PCA instance
pca = PCA()

# fit on data
pca = pca.fit(Xdat)

#Q eigenvectors
Q=pca.components_.T

#Explained deviations
s=sqrt(pca.explained_variance_)
```

[ ]:
```
# transform data obtain sample scores
F = pca.transform(Xdat)
print("Sample scores")

## Sample Scores ##
sample_scores = pd.DataFrame(F)
print(sample_scores)
```

```
Sample scores
             0         1         2         3         4         5         6  \
0    -2.973015  0.781343 -0.293940  0.160697 -0.191583 -0.256636  0.616214
1    -2.947366  0.863312 -0.243964  0.105630 -0.382404  0.037552  0.653964
2    -3.017666  0.574807 -0.308435  0.217721 -0.261207 -0.227693  0.760024
3    -2.949296  0.502979 -0.269612  0.143098 -0.222600 -0.368434  0.829817
4    -2.909534  0.452585 -0.236762  0.056849 -0.087086 -0.704797  0.918991
…          …         …         …         …         …         …         …
4563 -2.356444  0.518253 -0.074877 -0.218139 -0.226226 -0.611123  0.037623
4564 -2.250028  0.807271  0.007142 -0.376510 -0.290801 -0.369273  0.039238
4565 -2.353204  0.712971 -0.098954 -0.239667 -0.005931 -0.698476  0.048795
4566 -2.361532  0.790199 -0.259546 -0.031618  0.435557 -1.033298 -0.033373
4567 -2.601029  0.775102 -0.334546  0.175355  0.205479 -0.873953  0.067749

             7
0     0.091657
1     0.054221
2     0.102594
3     0.080628
4     0.067668
…          …
4563  0.025014
4564 -0.059383
4565 -0.006419
4566  0.161941
```

```
4567  0.318410

[4568 rows x 8 columns]
```

[ ]:
```
# Obtain Factor Loadings
L = zeros(shape=(Q.shape))
for col in range(0,len(s)-1):
    L[:,col] = Q[:,col] * s[col]
print("Factor Loadings")

columms_factorLoading = []

for i in range(0,8):
    nombre =  "F" + str(i)
    columms_factorLoading.append(nombre)


## Factor Loadings Dataframe ##
factor_loadings = pd.DataFrame(L, columns= columms_factorLoading)
print(factor_loadings)
```

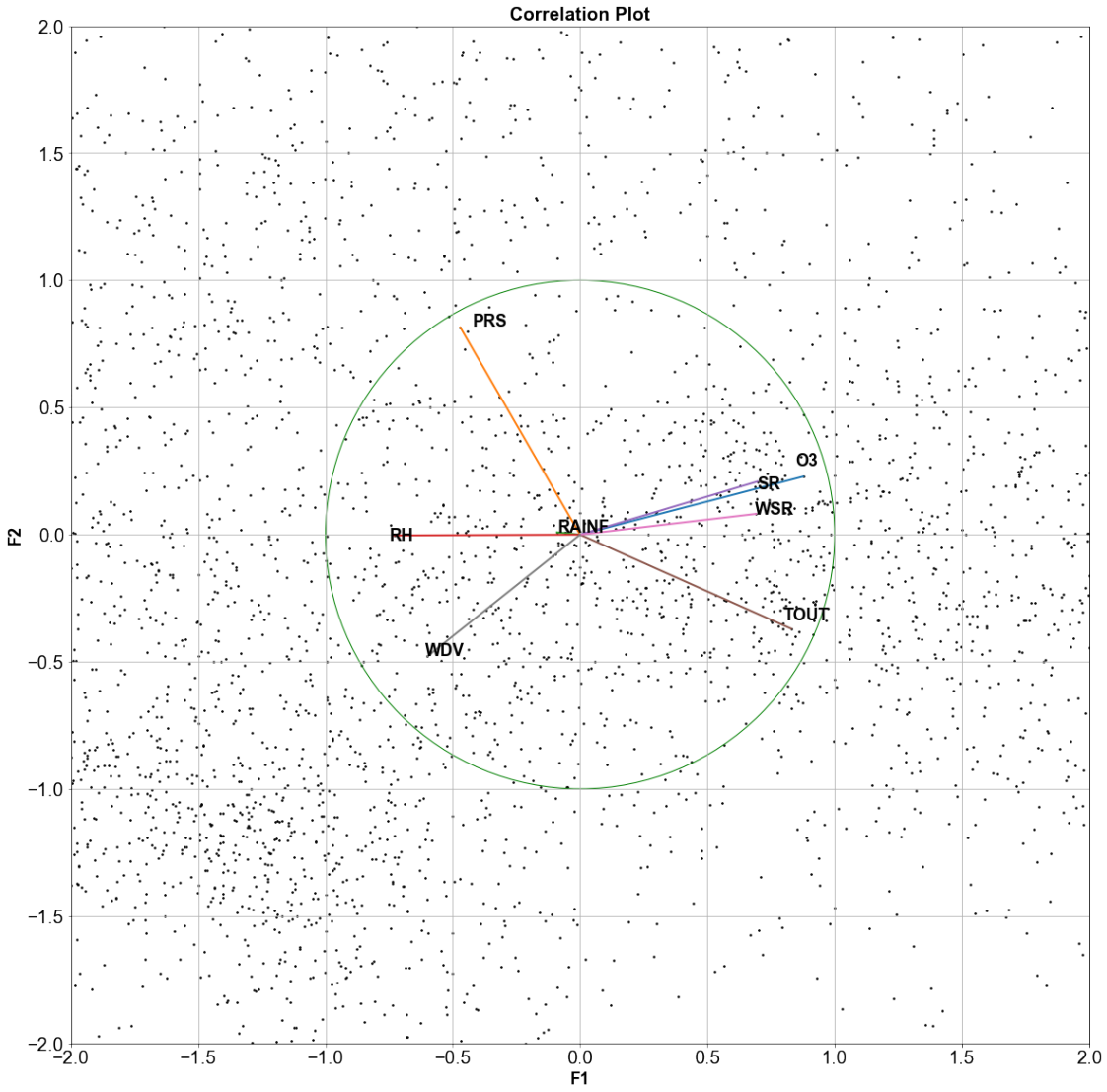
```
Factor Loadings
         F0        F1        F2        F3        F4        F5        F6   F7
0  0.876645  0.228066 -0.052062  0.229569 -0.074257 -0.028589  0.125902  0.0
1 -0.470489  0.814261 -0.088344  0.058841  0.161716 -0.135678 -0.241693  0.0
2 -0.090534  0.009189  0.898894  0.415382  0.104035  0.013251 -0.009485  0.0
3 -0.729911 -0.003111  0.176729 -0.114974 -0.609708 -0.214445 -0.014950  0.0
4  0.697085  0.208586 -0.211437  0.540917 -0.242148 -0.149118  0.042334  0.0
5  0.829793 -0.370418  0.047870 -0.027931 -0.099460 -0.019873 -0.399342  0.0
6  0.691044  0.081325  0.253585 -0.490102  0.147691 -0.422401  0.087850  0.0
7 -0.599970 -0.479186 -0.244812  0.391830  0.241952 -0.363747 -0.008827  0.0
```

[ ]:
```
#Obtain squared cosines
COS2=L**2
print("Square Cosines")

## COS2  Dataframe ##
COS2_DF = pd.DataFrame(COS2)
print(COS2_DF)
```

```
Square Cosines
          0         1         2         3         4         5         6    7
0  0.768507  0.052014  0.002710  0.052702  0.005514  0.000817  0.015851  0.0
1  0.221360  0.663021  0.007805  0.003462  0.026152  0.018408  0.058416  0.0
2  0.008196  0.000084  0.808010  0.172542  0.010823  0.000176  0.000090  0.0
3  0.532770  0.000010  0.031233  0.013219  0.371744  0.045987  0.000224  0.0
4  0.485927  0.043508  0.044706  0.292592  0.058635  0.022236  0.001792  0.0
5  0.688557  0.137209  0.002292  0.000780  0.009892  0.000395  0.159474  0.0
6  0.477542  0.006614  0.064306  0.240200  0.021813  0.178422  0.007718  0.0
```

```
7  0.359964  0.229620  0.059933  0.153531  0.058541  0.132312  0.000078  0.0
```

[ ]:
```
import random

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.grid()

circ=plt.Circle((0,0), radius=1, color='g', fill=False)
ax.add_patch(circ)
ax.set_aspect('equal')

plt.scatter(L[:,0],L[:,1],marker=".", c = "k" , s = 10)
plt.scatter(F[:,0],F[:,1],marker=".", c = "k" , s = 10)


for row in range(0,len(s)):
    plt.plot([0, L[row,0]], [0,L[row,1]], linewidth=2,label='X1')

for row in range(0,len(s)):
    #ax.annotate('X'+str(row+1), (L[row,0]+0.01,L[row,1]+0.01))
    ax.annotate(header_row[row], (L[row,0]+random.uniform(-0.05,0.
 ↪05),L[row,1]+random.uniform(-0.05,0.05)),**fontL)

plt.ylabel('F2',**fontL)
plt.xlabel('F1',**fontL)

plt.xlim(-2,2)
plt.ylim(-2,2)

plt.xticks(fontsize = 20 , family = "Arial")
plt.yticks(fontsize = 20 , family = "Arial")

plt.title('Correlation Plot',**fontT)
fig.set_size_inches(30, 20)
fig.savefig('PCA_Correlation_Plot.jpg', dpi=300)
plt.show()
```



2.1 PCA PM 2.5

[ ]:
```
#Only O3 Dataframe
PM25_df = df.drop(["PM10","O3"],axis=1)
header_row=PM25_df.columns.values

y= PM25_df.values
```

[ ]:
```
M = mean(PM25_df.T, axis=1)
print(M)

#Std Deviation
st = std(PM25_df.T, axis=1,ddof=1)
```

```
#Normalize data
Xdat = (PM25_df - M)/st

print(st)
print(Xdat)
```

```
PM2.5     26.525394
PRS      713.659173
RAINF      0.001627
RH        66.011602
SR         0.169129
TOUT      21.012585
WSR        8.282903
WDV      177.051664
dtype: float64
PM2.5    17.691460
PRS       3.865896
RAINF     0.018431
RH       18.935894
SR        0.252062
TOUT      7.698031
WSR       5.418112
WDV      84.824669
dtype: float64
         PM2.5       PRS    RAINF        RH        SR      TOUT       WSR  \
0     8.053298  1.096984 -0.08825  1.319631 -0.670983 -2.058005 -0.956588
1     1.892134  1.019383 -0.08825  1.372441 -0.670983 -2.089182 -1.122698
2    -0.029698  0.864179 -0.08825  1.372441 -0.670983 -2.115162 -1.030415
3    -0.538418  0.786578 -0.08825  1.372441 -0.670983 -2.115162 -0.845849
4    -0.538418  0.786578 -0.08825  1.372441 -0.670983 -2.156731 -0.513630
…         …       …       …        …        …       …        …
4563  0.705120  1.045250 -0.08825  1.319631 -0.670983 -1.195447 -0.365977
4564  0.592071  1.148719 -0.08825  1.266822 -0.670983 -1.248707 -0.365977
4565  2.061707  1.252188 -0.08825  1.161202 -0.670983 -1.300668 -0.236780
4566  3.418294  1.562595 -0.08825  0.897153 -0.670983 -1.311061 -0.107584
4567  2.740000  1.510860 -0.08825  1.108392 -0.670983 -1.463048 -0.476716

          WDV
0     0.824623
1     0.506319
2     0.918935
3     0.977880
4     1.178293
…        …
4563  0.730310
4564  0.305906
4565  0.753888
```

```
4566  1.213660
4567  1.260816

[4568 rows x 8 columns]
```

[ ]:
```
# create the PCA instance
pca = PCA()

# fit on data
pca = pca.fit(Xdat)

#Q eigenvectors
Q=pca.components_.T

#Explained deviations
s=sqrt(pca.explained_variance_)
```

[ ]:
```
# transform data obtain sample scores
F = pca.transform(Xdat)
print("Sample scores")

## Sample Scores ##
sample_scores = pd.DataFrame(F)
print(sample_scores)
```

```
Sample scores
             0         1         2         3         4         5         6  \
0    -3.793897  4.328294 -0.250655 -0.740379  6.313149  0.584714 -0.389229
1    -3.082075  0.498428 -0.246843 -0.186918  1.659648 -0.155361  0.019025
2    -2.939054 -0.428130 -0.276161  0.049092 -0.031072 -0.200549 -0.200446
3    -2.799625 -0.700463 -0.244835  0.017998 -0.453126 -0.200655 -0.329797
4    -2.748384 -0.683990 -0.219661 -0.071032 -0.487353 -0.054267 -0.662582
…          …         …         …         …         …         …         …
4563 -2.233655 -0.075080 -0.116989 -0.328899  0.609671 -0.127597 -0.616764
4564 -2.107943 -0.400250 -0.087583 -0.405623  0.746632 -0.202909 -0.381046
4565 -2.387128  0.592431 -0.156800 -0.424936  1.759240  0.211144 -0.732067
4566 -2.633073  1.407441 -0.284039 -0.343463  2.755407  0.774203 -1.083386
4567 -2.885175  1.075522 -0.319104 -0.167207  2.169503  0.498747 -0.901421

             7
0    -0.029261
1     0.469025
2     0.698586
3     0.807867
4     0.894233
…          …
4563 -0.021710
4564  0.009060
```

```
4565 -0.105899
4566 -0.338760
4567 -0.235464

[4568 rows x 8 columns]
```

[ ]:
```
# Obtain Factor Loadings
L = zeros(shape=(Q.shape))
for col in range(0,len(s)-1):
    L[:,col] = Q[:,col] * s[col]
print("Factor Loadings")

columms_factorLoading = []

for i in range(0,8):
    nombre =  "F" + str(i)
    columms_factorLoading.append(nombre)


## Factor Loadings Dataframe ##
factor_loadings = pd.DataFrame(L, columns= columms_factorLoading)
print(factor_loadings)
```

```
Factor Loadings
         F0        F1        F2        F3        F4        F5        F6   F7
0 -0.188521  0.684693  0.005049 -0.090498  0.693997  0.066472 -0.012062  0.0
1 -0.526756 -0.668908 -0.147707  0.189740  0.342684  0.146409 -0.142325  0.0
2 -0.103958 -0.023820  0.940739  0.307685  0.026740  0.089939  0.011519  0.0
3 -0.727287 -0.000886  0.178880 -0.257531 -0.021261 -0.572132 -0.209374  0.0
4  0.620298 -0.024450 -0.170263  0.638050  0.202806 -0.301640 -0.173132  0.0
5  0.862262  0.259744  0.061238 -0.006900 -0.151343 -0.128836 -0.026845  0.0
6  0.729885 -0.202157  0.166902 -0.411817  0.111837  0.168534 -0.422587  0.0
7 -0.597825  0.481148 -0.137216  0.270445 -0.370895  0.237384 -0.352375  0.0
```

[ ]:
```
#Obtain squared cosines
COS2=L**2
print("Square Cosines")

## COS2  Dataframe ##
COS2_DF = pd.DataFrame(COS2)
print(COS2_DF)
```

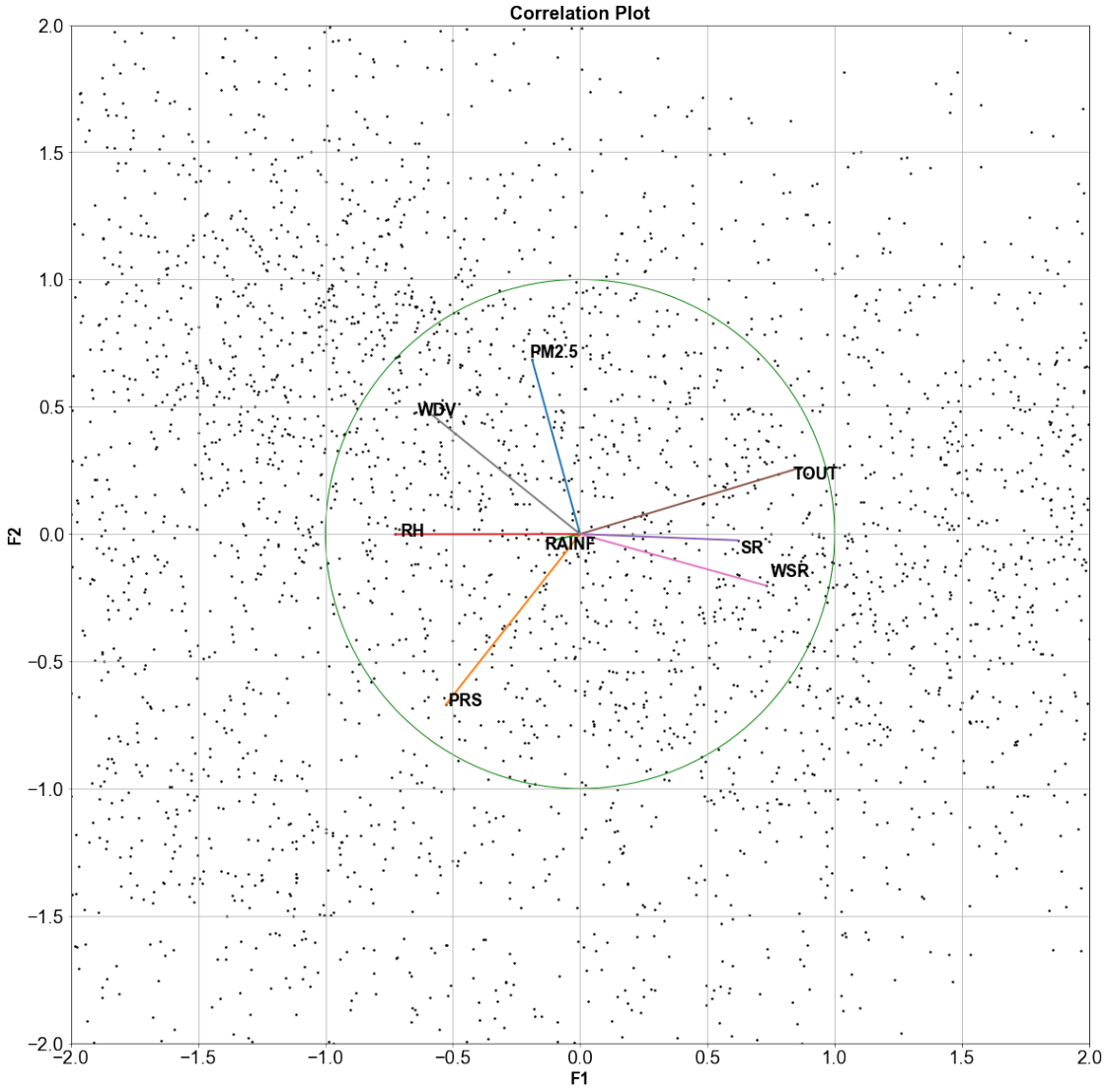
```
Square Cosines
          0             1         2         3         4         5         6  \
0  0.035540  4.688044e-01  0.000025  0.008190  0.481632  0.004418  0.000145
1  0.277471  4.474384e-01  0.021817  0.036001  0.117432  0.021436  0.020257
2  0.010807  5.673756e-04  0.884990  0.094670  0.000715  0.008089  0.000133
3  0.528946  7.842724e-07  0.031998  0.066322  0.000452  0.327335  0.043837
4  0.384769  5.978092e-04  0.028989  0.407107  0.041130  0.090987  0.029975
```

```
5  0.743497  6.746679e-02  0.003750  0.000048  0.022905  0.016599  0.000721
6  0.532732  4.086745e-02  0.027856  0.169594  0.012508  0.028404  0.178580
7  0.357395  2.315029e-01  0.018828  0.073140  0.137563  0.056351  0.124168
```

```
     7
0  0.0
1  0.0
2  0.0
3  0.0
4  0.0
5  0.0
6  0.0
7  0.0
```

[ ]:
```
import random

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.grid()

circ=plt.Circle((0,0), radius=1, color='g', fill=False)
ax.add_patch(circ)
ax.set_aspect('equal')

plt.scatter(L[:,0],L[:,1],marker=".", c = "k" , s = 10)
plt.scatter(F[:,0],F[:,1],marker=".", c = "k" , s = 10)


for row in range(0,len(s)):
    plt.plot([0, L[row,0]], [0,L[row,1]], linewidth=2,label='X1')

for row in range(0,len(s)):
    #ax.annotate('X'+str(row+1), (L[row,0]+0.01,L[row,1]+0.01))
    ax.annotate(header_row[row], (L[row,0]+random.uniform(-0.05,0.
 ↪05),L[row,1]+random.uniform(-0.05,0.05)),**fontL)

plt.ylabel('F2',**fontL)
plt.xlabel('F1',**fontL)

plt.xlim(-2,2)
plt.ylim(-2,2)

plt.xticks(fontsize = 20 , family = "Arial")
plt.yticks(fontsize = 20 , family = "Arial")

plt.title('Correlation Plot',**fontT)
fig.set_size_inches(30, 20)
```

```
fig.savefig('PCA_Correlation_Plot.jpg', dpi=300)
plt.show()
```



2.2 Regression Models

[ ]:
```
df = pd.read_csv("Datos SIMA 2015-Noroeste.csv",sep=",",encoding =␣
↪'unicode_escape')
column_names = df.columns
df = df.drop(['Date'],axis=1)
df.head()
```

[ ]:

	O3	PM10	PM2.5	PRS	RAINF	RH	SR	TOUT	WSR	WDV
0	4	199	169	717.9	0.0	91	0.0	5.17	3.1	247

```
1   4   107     60  717.6    0.0  92  0.0  4.93  2.2  220
2   4    51     26  717.0    0.0  92  0.0  4.73  2.7  255
3   4    33     17  716.7    0.0  92  0.0  4.73  3.7  260
4   4    30     17  716.7    0.0  92  0.0  4.41  5.5  277
```

[ ]:
```
O3 = df ["O3"]
PM_25 = df["PM2.5"]
PM_10 = df["PM10"]

column_names = df.columns
column_names = column_names.drop(["O3","PM2.5","PM10"])

x  = df[column_names]
x.head()
```

[ ]:
```
     PRS  RAINF  RH   SR  TOUT  WSR  WDV
0  717.9    0.0  91  0.0  5.17  3.1  247
1  717.6    0.0  92  0.0  4.93  2.2  220
2  717.0    0.0  92  0.0  4.73  2.7  255
3  716.7    0.0  92  0.0  4.73  3.7  260
4  716.7    0.0  92  0.0  4.41  5.5  277
```

2.3 O3 Model

Linear Regresion

[ ]:
```
## Predicting Variable ##
x  = df[column_names]

## Y model of O3 ##
y = O3.values
```

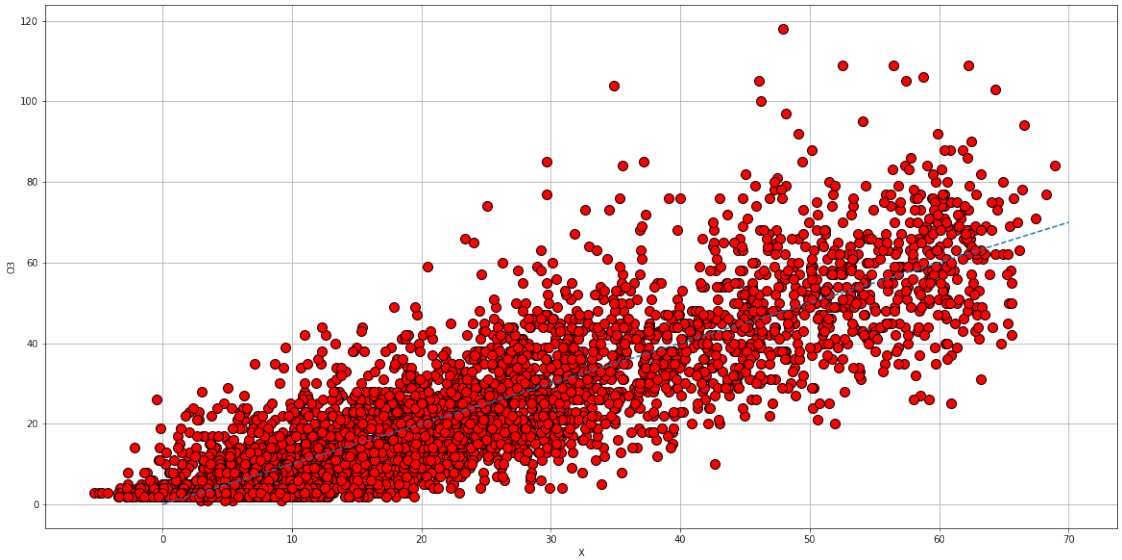
[ ]:
```
#Reshape the Values#
y = y.reshape(-1,1)
y.shape
```

[ ]:
```
(4568, 1)
```

[ ]:
```
lm = sk.linear_model.LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)
```

```
explained_variance:  0.7499
r2:  0.7499
MAE:  7.1157
MSE:  98.0229
RMSE:  9.9007
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(lmp,y,"ok",markersize =10, linewidth = 4 , markerfacecolor = 'red')
plt.plot([0,70],[0,70],"--")
plt.ylabel("O3")
plt.xlabel("X")
plt.grid(True)
```



2.4 PM 2.5 Model

[ ]:
```
## Predicting Variable ##
x  = df[column_names]

## Y model of O3 ##
y = PM_25.values
```

[ ]:
```
y = y.reshape(-1,1)
y.shape
```

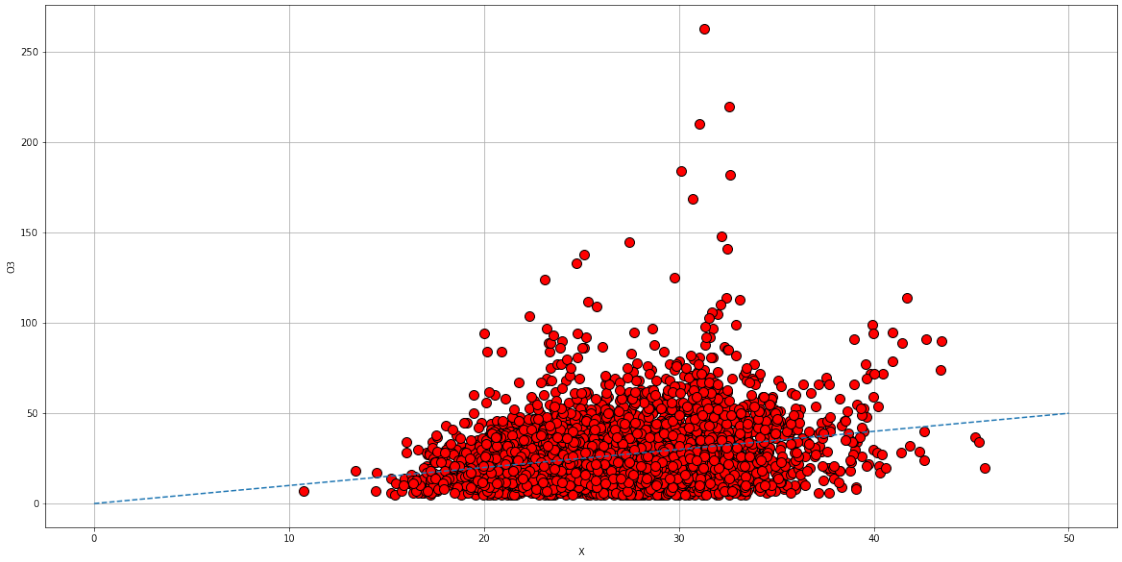
[ ]: (4568, 1)

[ ]:
```
lm = sk.linear_model.LinearRegression().fit(x,y)
lmp = lm.predict(x)
regression_results(y,lmp)
```

```
explained_variance:  0.0713
r2:  0.0713
MAE:  11.7951
```

```
MSE:  290.6145
RMSE:  17.0474
```

[ ]:
```
plt.figure(figsize=(20,10))

plt.plot(lmp,y,"ok",markersize =10, linewidth = 4 , markerfacecolor = 'red')
plt.plot([0,50],[0,50],"--")
plt.ylabel("O3")
plt.xlabel("X")
plt.grid(True)
```



2.5 Classification Model

[ ]:
```
df['CONT']=(df['PM2.5']>40.4)*1

df.head()
```

[ ]:

	O3	PM10	PM2.5	PRS	RAINF	RH	SR	TOUT	WSR	WDV	CONT
0	4	199	169	717.9	0.0	91	0.0	5.17	3.1	247	1
1	4	107	60	717.6	0.0	92	0.0	4.93	2.2	220	1
2	4	51	26	717.0	0.0	92	0.0	4.73	2.7	255	0
3	4	33	17	716.7	0.0	92	0.0	4.73	3.7	260	0
4	4	30	17	716.7	0.0	92	0.0	4.41	5.5	277	0

[ ]:
```
## Predicting Variable ##
x  = df[column_names]
x.head()
```

[ ]:

	PRS	RAINF	RH	SR	TOUT	WSR	WDV
0	717.9	0.0	91	0.0	5.17	3.1	247
1	717.6	0.0	92	0.0	4.93	2.2	220
2	717.0	0.0	92	0.0	4.73	2.7	255
3	716.7	0.0	92	0.0	4.73	3.7	260
4	716.7	0.0	92	0.0	4.41	5.5	277

[ ]:
```
#Independant#
y = df['CONT']
y.head()
```

[ ]:
```
0    1
1    1
2    0
3    0
4    0
Name: CONT, dtype: int32
```

[ ]:
```
#Separate train and test data
x_train,x_test,y_train,y_test = sk.model_selection.
 ↪train_test_split(x,y,train_size=0.20,random_state=0)

#Standarize data. Subtract mean and devide by standard deviation
sc = sk.preprocessing.StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

[ ]:
```
from sklearn.naive_bayes import GaussianNB
#Fit classifier with train data
NB = GaussianNB()
NB.fit(x_train,y_train)
```

[ ]:
```
GaussianNB()
```

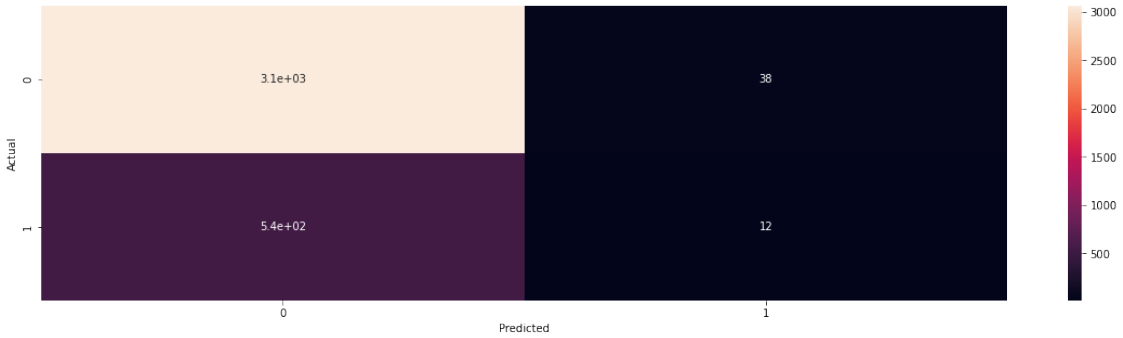
[ ]:
```
#Predict test data
y_pred=NB.predict(x_test)
```

[ ]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
tn, fp, fn, tp = cm.ravel()
```

```
[[3067   38]
 [ 538   12]]
```

[ ]:
```
cm2 = pd.crosstab(y_test,y_pred,rownames=['Actual'],colnames=['Predicted'])
sns.heatmap(cm2,annot=True)
```

[ ]: <AxesSubplot:xlabel='Predicted', ylabel='Actual'>



[ ]:
```
#Get performance measures
from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.91      3105
           1       0.24      0.02      0.04       550

    accuracy                           0.84      3655
   macro avg       0.55      0.50      0.48      3655
weighted avg       0.76      0.84      0.78      3655
```

[ ]:
```
recall=tp/(tp+fn)
print("Likelihood Ratio",recall/(1-recall))
print("Accuracy=",(tp+tn)/(tp+tn+fp+fn))
```

```
Likelihood Ratio 0.02230483271375465
Accuracy= 0.8424076607387141
```

2.6 Adjusting the data set

[ ]:
```
df = pd.read_csv("Datos SIMA 2015-Noroeste.csv",sep=",")


O3 = df ["O3"]
PM_25 = df["PM2.5"]
PM_10 = df["PM10"]

column_names = df.columns
column_names = column_names.drop(["O3","PM10","PM2.5"])
```

```
df_2  = df[column_names]

df.head()
```

[ ]:

	Date	O3	PM10	PM2.5	PRS	RAINF	RH	SR	TOUT	WSR	WDV
0	01-ene-15	4	199	169	717.9	0.0	91	0.0	5.17	3.1	247
1	01-ene-15	4	107	60	717.6	0.0	92	0.0	4.93	2.2	220
2	01-ene-15	4	51	26	717.0	0.0	92	0.0	4.73	2.7	255
3	01-ene-15	4	33	17	716.7	0.0	92	0.0	4.73	3.7	260
4	01-ene-15	4	30	17	716.7	0.0	92	0.0	4.41	5.5	277

[ ]:
```
## Group data by date##
pm25_max=df.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV','PM2.
 ↪5'].max()


pm25_max=pm25_max.rename(columns={'PRS':'PRS_max','RAINF':'RAINF_max',\
                          'RH':'RH_max','SR':'SR_max','TOUT':'TOUT_max','WSR':
 ↪'WSR_max',\
                          'WDV':'WDV_max','PM2.5':'PM2.5_max'})

pm25_max.head()
```

```
C:\Users\Usuario\AppData\Local\Temp/ipykernel_46288/4100089267.py:2:
FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of
keys) will be deprecated, use a list instead.
  pm25_max=df.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV','PM2
.5'].max()
```

[ ]:

Date	PRS_max	RAINF_max	RH_max	SR_max	TOUT_max	WSR_max	WDV_max	\
01-dic-15	713.7	0.00	92	0.291	17.65	7.6	324	
01-ene-15	717.9	0.00	94	0.121	7.91	9.7	294	
01-feb-15	715.9	0.02	82	0.001	17.52	10.2	291	
01-jul-15	713.7	0.10	89	0.257	25.70	18.4	305	
01-nov-15	713.8	0.00	87	0.051	24.45	8.2	340	

Date	PM2.5_max
01-dic-15	80
01-ene-15	169
01-feb-15	20
01-jul-15	33
01-nov-15	41

[ ]:
```
pm25_min=df.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV'].min()

pm25_min=pm25_min.rename(columns={'PRS':'PRS_min','RAINF':'RAINF_min',\
                          'RH':'RH_min','SR':'SR_min','TOUT':'TOUT_min','WSR':
 ↪'WSR_min',\
                          'WDV':'WDV_min'})
pm25_min.head()
```

```
C:\Users\Usuario\AppData\Local\Temp/ipykernel_46288/942626340.py:1:
FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of
keys) will be deprecated, use a list instead.
pm25_min=df.groupby(by='Date')['PRS','RAINF','RH','SR','TOUT','WSR','WDV'].min()
```

[ ]:

Date	PRS_min	RAINF_min	RH_min	SR_min	TOUT_min	WSR_min	WDV_min
01-dic-15	711.3	0.0	70	0.0	12.68	2.8	121
01-ene-15	713.3	0.0	85	0.0	4.31	2.0	111
01-feb-15	713.9	0.0	72	0.0	16.60	5.0	138
01-jul-15	712.1	0.0	69	0.0	21.31	4.1	102
01-nov-15	712.8	0.0	39	0.0	16.47	2.0	102

[ ]:
```
pm25 = pm25_max.join(pm25_min)
pm25.head()
```

[ ]:

Date	PRS_max	RAINF_max	RH_max	SR_max	TOUT_max	WSR_max	WDV_max	\
01-dic-15	713.7	0.00	92	0.291	17.65	7.6	324	
01-ene-15	717.9	0.00	94	0.121	7.91	9.7	294	
01-feb-15	715.9	0.02	82	0.001	17.52	10.2	291	
01-jul-15	713.7	0.10	89	0.257	25.70	18.4	305	
01-nov-15	713.8	0.00	87	0.051	24.45	8.2	340	

Date	PM2.5_max	PRS_min	RAINF_min	RH_min	SR_min	TOUT_min	WSR_min	\
01-dic-15	80	711.3	0.0	70	0.0	12.68	2.8	
01-ene-15	169	713.3	0.0	85	0.0	4.31	2.0	
01-feb-15	20	713.9	0.0	72	0.0	16.60	5.0	
01-jul-15	33	712.1	0.0	69	0.0	21.31	4.1	
01-nov-15	41	712.8	0.0	39	0.0	16.47	2.0	

Date	WDV_min
01-dic-15	121
01-ene-15	111
01-feb-15	138
01-jul-15	102
01-nov-15	102

[ ]:
```
pm25_avg=df.groupby(by='Date')[column_names].mean()
pm25_avg=pm25_avg.rename(columns={'PRS':'PRS_avg','RAINF':'RAINF_avg',\
                        'RH':'RH_avg','SR':'SR_avg','TOUT':'TOUT_avg','WSR':
↪'WSR_avg',\
                        'WDV':'WDV_avg'})
```

[ ]:
```
pm25 = pm25.join(pm25_avg)
pm25.head()
```

[ ]:

Date	PRS_max	RAINF_max	RH_max	SR_max	TOUT_max	WSR_max	WDV_max	\
01-dic-15	713.7	0.00	92	0.291	17.65	7.6	324	
01-ene-15	717.9	0.00	94	0.121	7.91	9.7	294	
01-feb-15	715.9	0.02	82	0.001	17.52	10.2	291	
01-jul-15	713.7	0.10	89	0.257	25.70	18.4	305	
01-nov-15	713.8	0.00	87	0.051	24.45	8.2	340	

Date	PM2.5_max	PRS_min	RAINF_min	…	TOUT_min	WSR_min	WDV_min	\
01-dic-15	80	711.3	0.0	…	12.68	2.8	121	
01-ene-15	169	713.3	0.0	…	4.31	2.0	111	
01-feb-15	20	713.9	0.0	…	16.60	5.0	138	
01-jul-15	33	712.1	0.0	…	21.31	4.1	102	
01-nov-15	41	712.8	0.0	…	16.47	2.0	102	

Date	PRS_avg	RAINF_avg	RH_avg	SR_avg	TOUT_avg	WSR_avg	\
01-dic-15	712.553333	0.000	82.20	0.080333	14.784667	5.146667	
01-ene-15	715.252381	0.000	90.00	0.032095	6.140000	4.423810	
01-feb-15	714.840000	0.006	79.00	0.000200	17.192000	7.160000	
01-jul-15	713.037500	0.020	78.25	0.122750	23.637500	11.275000	
01-nov-15	713.190000	0.000	72.30	0.005200	18.995000	4.840000	

Date	WDV_avg
01-dic-15	195.600000
01-ene-15	193.095238
01-feb-15	216.600000
01-jul-15	150.375000
01-nov-15	234.800000

[5 rows x 22 columns]

[ ]:
```
pm25['CONT']=(pm25['PM2.5_max']>40.4)*1

pm25.head()
```

[ ]:
```
            PRS_max  RAINF_max  RH_max  SR_max  TOUT_max  WSR_max  WDV_max  \
Date
01-dic-15     713.7       0.00      92   0.291     17.65      7.6      324
01-ene-15     717.9       0.00      94   0.121      7.91      9.7      294
01-feb-15     715.9       0.02      82   0.001     17.52     10.2      291
01-jul-15     713.7       0.10      89   0.257     25.70     18.4      305
01-nov-15     713.8       0.00      87   0.051     24.45      8.2      340

            PM2.5_max  PRS_min  RAINF_min  …  WSR_min  WDV_min     PRS_avg  \
Date                                        …
01-dic-15          80    711.3        0.0  …      2.8      121  712.553333
01-ene-15         169    713.3        0.0  …      2.0      111  715.252381
01-feb-15          20    713.9        0.0  …      5.0      138  714.840000
01-jul-15          33    712.1        0.0  …      4.1      102  713.037500
01-nov-15          41    712.8        0.0  …      2.0      102  713.190000

            RAINF_avg  RH_avg    SR_avg   TOUT_avg    WSR_avg     WDV_avg  CONT
Date
01-dic-15       0.000   82.20  0.080333  14.784667   5.146667  195.600000     1
01-ene-15       0.000   90.00  0.032095   6.140000   4.423810  193.095238     1
01-feb-15       0.006   79.00  0.000200  17.192000   7.160000  216.600000     0
01-jul-15       0.020   78.25  0.122750  23.637500  11.275000  150.375000     0
01-nov-15       0.000   72.30  0.005200  18.995000   4.840000  234.800000     1

[5 rows x 23 columns]
```

[ ]:
```
column_names = pm25.columns
column_names = column_names.drop(["CONT"])

x  = pm25[column_names]
```

[ ]:
```
#Dependant
x = x
x.head()
```

[ ]:
```
            PRS_max  RAINF_max  RH_max  SR_max  TOUT_max  WSR_max  WDV_max  \
Date
01-dic-15     713.7       0.00      92   0.291     17.65      7.6      324
01-ene-15     717.9       0.00      94   0.121      7.91      9.7      294
01-feb-15     715.9       0.02      82   0.001     17.52     10.2      291
01-jul-15     713.7       0.10      89   0.257     25.70     18.4      305
01-nov-15     713.8       0.00      87   0.051     24.45      8.2      340

            PM2.5_max  PRS_min  RAINF_min  …  TOUT_min  WSR_min  WDV_min  \
Date                                        …
01-dic-15          80    711.3        0.0  …     12.68      2.8      121
01-ene-15         169    713.3        0.0  …      4.31      2.0      111
```

```
01-feb-15         20    713.9        0.0  …     16.60      5.0      138
01-jul-15         33    712.1        0.0  …     21.31      4.1      102
01-nov-15         41    712.8        0.0  …     16.47      2.0      102

              PRS_avg  RAINF_avg  RH_avg    SR_avg   TOUT_avg    WSR_avg  \
Date
01-dic-15  712.553333      0.000   82.20  0.080333  14.784667   5.146667
01-ene-15  715.252381      0.000   90.00  0.032095   6.140000   4.423810
01-feb-15  714.840000      0.006   79.00  0.000200  17.192000   7.160000
01-jul-15  713.037500      0.020   78.25  0.122750  23.637500  11.275000
01-nov-15  713.190000      0.000   72.30  0.005200  18.995000   4.840000

              WDV_avg
Date
01-dic-15  195.600000
01-ene-15  193.095238
01-feb-15  216.600000
01-jul-15  150.375000
01-nov-15  234.800000

[5 rows x 22 columns]
```

[ ]:
```
#Independant#
y = pm25['CONT']
y.head()
```

[ ]:
```
Date
01-dic-15    1
01-ene-15    1
01-feb-15    0
01-jul-15    0
01-nov-15    1
Name: CONT, dtype: int32
```

[ ]:
```
#Separate train and test data
x_train,x_test,y_train,y_test = sk.model_selection.
 ↪train_test_split(x,y,train_size=0.20,random_state=0)

#Standarize data. Subtract mean and devide by standard deviation
sc = sk.preprocessing.StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

[ ]:
```
from sklearn.naive_bayes import GaussianNB
#Fit classifier with train data
NB = GaussianNB()
NB.fit(x_train,y_train)
```

[ ]:
```
GaussianNB()
```

[ ]:
```
#Predict test data
y_pred=NB.predict(x_test)
```

[ ]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
tn, fp, fn, tp = cm.ravel()
```

```
[[ 60  18]
 [ 21 116]]
```

[ ]:
```
cm2 = pd.crosstab(y_test,y_pred,rownames=['Actual'],colnames=['Predicted'])
sns.heatmap(cm2,annot=True)
```

[ ]:
```
<AxesSubplot:xlabel='Predicted', ylabel='Actual'>
```

[ ]:
```
#Get performance measures
from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.74      0.77      0.75        78
           1       0.87      0.85      0.86       137

    accuracy                           0.82       215
   macro avg       0.80      0.81      0.81       215
weighted avg       0.82      0.82      0.82       215
```

[ ]:
```
recall=tp/(tp+fn)
print("Likelihood Ratio",recall/(1-recall))
print("Accuracy=",(tp+tn)/(tp+tn+fp+fn))
```

```
Likelihood Ratio 5.523809523809522
Accuracy= 0.8186046511627907
```

Conclusions

Is pollution related to traffic?

As we can see in the results of the correlation analysis, we could say that, out of the 3 main pollutants, where we can most appreciate a correlation is between O3 and the hour of day.

Would a vehicle verification help reduce pollution in metropolitan area?

We believe that the control or restriction of traffic by zones could help reduce ozone emissions into the atmosphere, since driving less reduces the number of vehicles on the streets, which helps reduce pollutants. On the other hand, we also believe that speed control or restriction would greatly help reduce this pollutant. This could prevent sudden increases in speed or accelerations that generate more ozone into the atmosphere.

In summary, we do believe that regulating traffic could help reduce the emission of ozone into the atmosphere. The idea is good, that Mexicans respect the rules is something else.

Does weather affect pollution?

Considering the Monterrey pollution dataset, the PCA graph of the O3 variable, we can see how there is a relationship between this variable and the Wind Speed, Solar Radiation and Hour variables. We can also see how the variable O3 is negatively related to the variable Relative Humidity.

Now, talking about the variable PM2.5, we can see how the only related variable is CONT, which is highly related to PM2.5. Without considering this variable, no other seems to have a relationship with the pollutant PM2.5.

According to SIMA dataset analysis, in the case of ozone (O3), weather conditions do relate with this pollutant. Thanks to the regression model of this pollutant, it was possible to see that the R^2 is 0.7499. With this number, it can be concluded that the correlation between weather conditions and ozone is significant.

Going deeper through the Principal Component Analysis, it was found that there are three weather variables correlated: 2 highly correlated and 1 inversely correlated, and one time variable. The firsts are wind speed and solar radiation. Also, relative humidity corresponds to the second.

Can we predict pollution emergencies? Under which weather conditions would we have a pollution emergency?

Yes, it is possible to predict pollution emergencies. Thanks to the analysis of the minimum, maximum, and average values of the entries, it was possible to determine that the absence of rain fall will lead to a pollution emergency, also, wind speed is a variable to consider, due to the fact that if it is not enough, presumably less than 10, the conditions will favor a pollution emergency.

This makes sense if real life is considered, if it is raining, particles are less likely to be present in the air. In a similar way, if the wind is strong, it will wipe off these particles.