ASSIGNMENT - 1

Finding the independent and dependent parameters using Correlation - 1

In [1]:
```
# importing packages pandas and numpy
import pandas as pd
import numpy as np
```

In [2]:
```
# loading the dataset
income = pd.read_csv("Income1.csv")
```

In [3]:
```
income.head()
```

Out[3]:

	Unnamed: 0	Education	Income
0	1	10.000000	26.658839
1	2	10.401338	27.306435
2	3	10.842809	22.132410
3	4	11.244147	21.169841
4	5	11.645485	15.192634

In [4]:
```
# removing column "unnamed: 0" since it is of no use
income = income.drop(['Unnamed: 0'], axis=1)
```

In [5]:
```
income.info()
income.head()
```

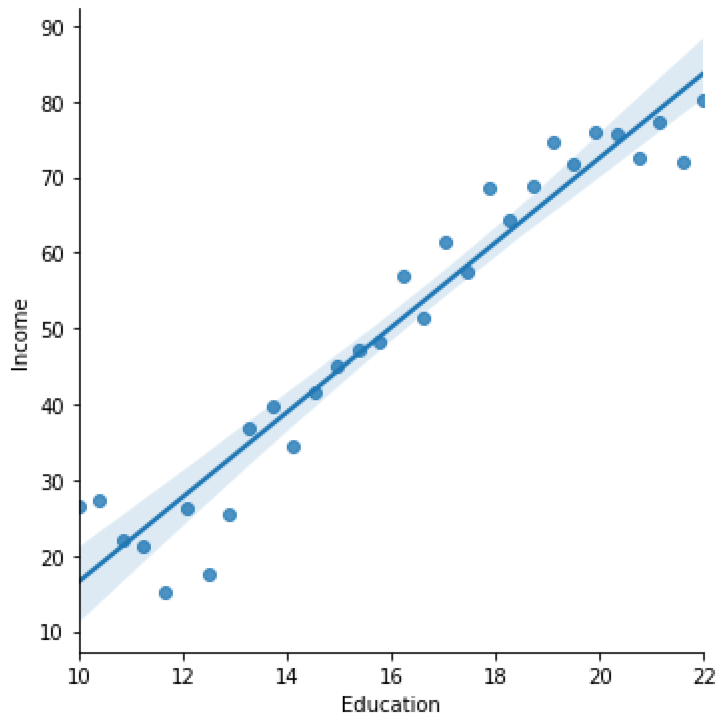
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Education  30 non-null     float64
 1   Income     30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

Out[5]:

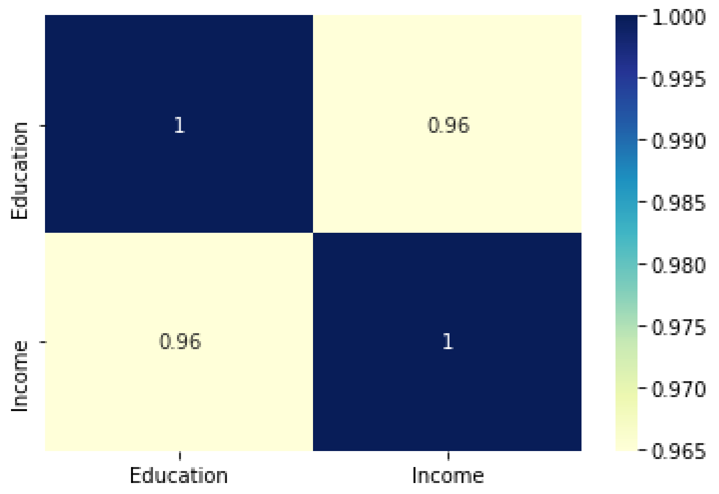
	Education	Income
0	10.000000	26.658839
1	10.401338	27.306435
2	10.842809	22.132410
3	11.244147	21.169841
4	11.645485	15.192634

In [6]:
```
# determining the relation between Education and Income
# importing packages seaborn and matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
sns.lmplot(x="Education", y="Income", data = income)
```

Out[6]: <seaborn.axisgrid.FacetGrid at 0x7fa53919f0d0>



In [7]:
```
# visualizing the data using heatmap
sns.heatmap(income.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



As we can see that the education is more correlated to income with correlation value 0.96 Hence Education can be considered as independent parameter in order to predict income which would be the dependent parameter.

In [8]:
```
X = income[['Education']] # taking independent parameter as X
y = income['Income'] # taking dependent parameter as y
```

ASSIGNMENT - 2

Finding the independent and dependent parameters using Correlation - 2

In [1]:
```
# importing packages pandas and numpy
import pandas as pd
import numpy as np
```

In [2]:
```
# loading the dataset
income = pd.read_csv("Income2.csv")
```

In [3]:
```
income.head()
```

Out[3]:

	Unnamed: 0	Education	Seniority	Income
0	1	21.586207	113.103448	99.917173
1	2	18.275862	119.310345	92.579135
2	3	12.068966	100.689655	34.678727
3	4	17.034483	187.586207	78.702806
4	5	19.931034	20.000000	68.009922

In [4]:
```
# removing column "unnamed: 0" since it is of no use
income = income.drop(['Unnamed: 0'], axis=1)
```

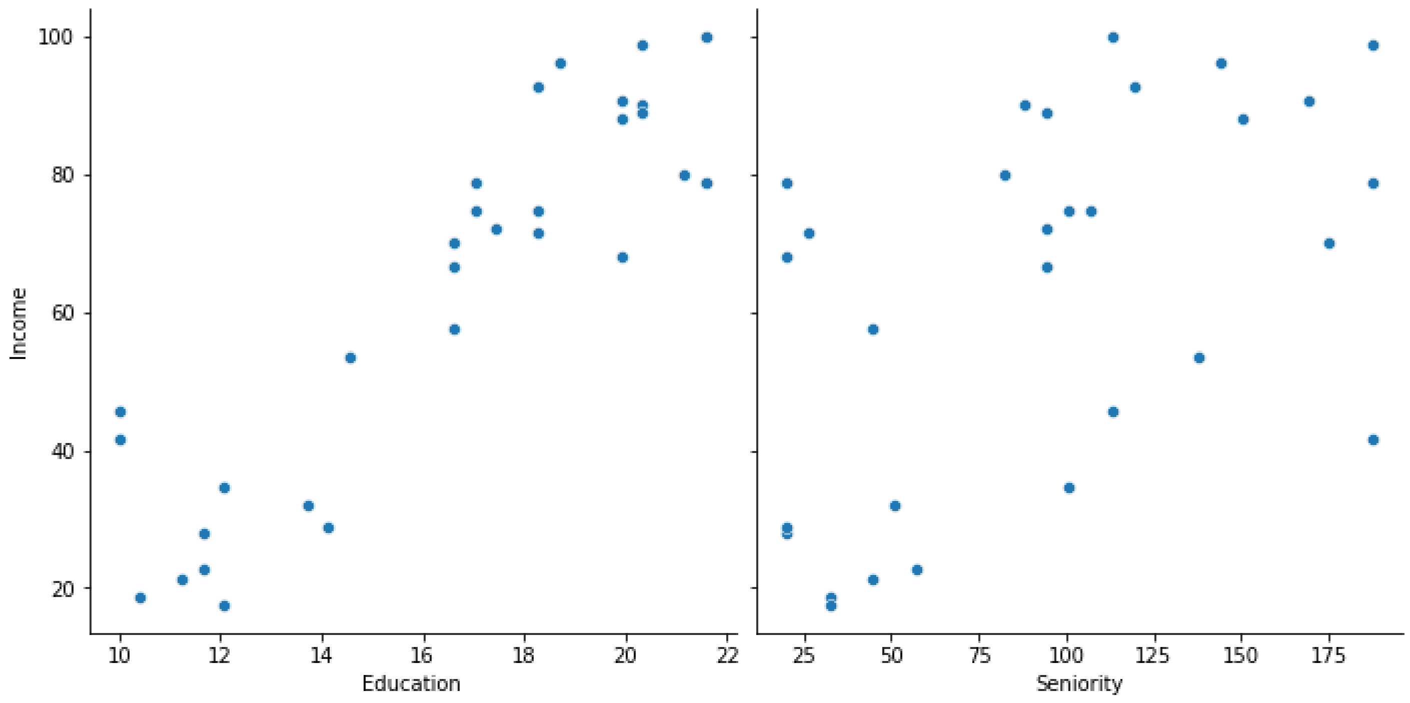
In [5]:
```
income.info()
income.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Education   30 non-null     float64
 1   Seniority   30 non-null     float64
 2   Income      30 non-null     float64
dtypes: float64(3)
memory usage: 848.0 bytes
```

Out[5]:

	Education	Seniority	Income
0	21.586207	113.103448	99.917173
1	18.275862	119.310345	92.579135
2	12.068966	100.689655	34.678727
3	17.034483	187.586207	78.702806
4	19.931034	20.000000	68.009922

In [6]:
```
# determining the relation between Education & Seniority, Seniority & Income and Education & Income
# importing packages seaborn and matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(income, x_vars=['Education', 'Seniority'], y_vars='Income', height = 5)
plt.show()
```



In [7]:
```
# visualizing the data using heatmap
sns.heatmap(income.corr(), cmap="YlGnBu", annot = True)
plt.show()
```

From the above correlation matrix, we could see that the Education is highly correlated to Income with 0.9 as its correlation value. That is, higher the education, the income would be higher. Hence Education can be considered as independent parameter in order to predict income which would be the dependent parameter.

In [8]:
```
X = income[['Education']] # taking independent parameter as X
y = income['Income'] # taking dependent parameter as y
```

ASSIGNMENT - 3

Finding the relationship between advertising media ans sales

In [1]:
```
# importing packages pandas and numpy
import pandas as pd
import numpy as np
```

In [2]:
```
# loading the dataset
advertising = pd.read_csv("Advertising.csv")
```

Advertising dataset provides us the information about the sales of a certain product in 200 different market along with its budget for each of those markets in 3 different media.

In [3]:
```
advertising.head()
```

Out[3]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [4]:
```
# removing column "unnamed: 0" since it is of no use
advertising = advertising.drop(['Unnamed: 0'], axis=1)
```

In [5]:
```
advertising.head()
```

Out[5]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

Advertising dataframe contains 4 columns and 200 entries with no missing values.

In [6]:
```
advertising.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   radio      200 non-null    float64
 2   newspaper  200 non-null    float64
 3   sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

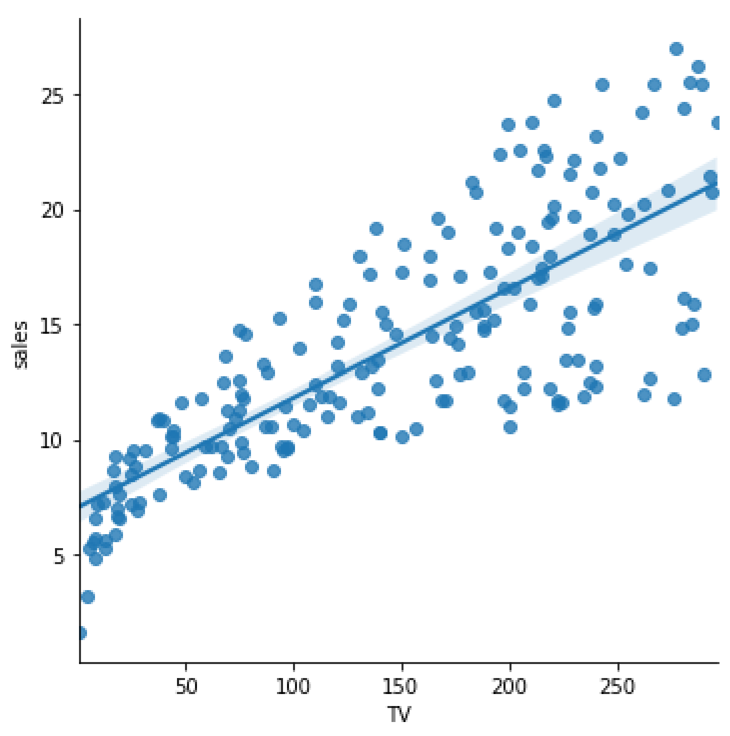
Our goal is to determine the relation between advertising and sales in order to indirectly increase the sales by adjusting the budget for advertisement.

In [7]:
```
# importing packages seaborn and matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
```

In [8]:
```
# visualizing the dataset
# plotting 3 different media with the sales in order to understand the relation between them.
```

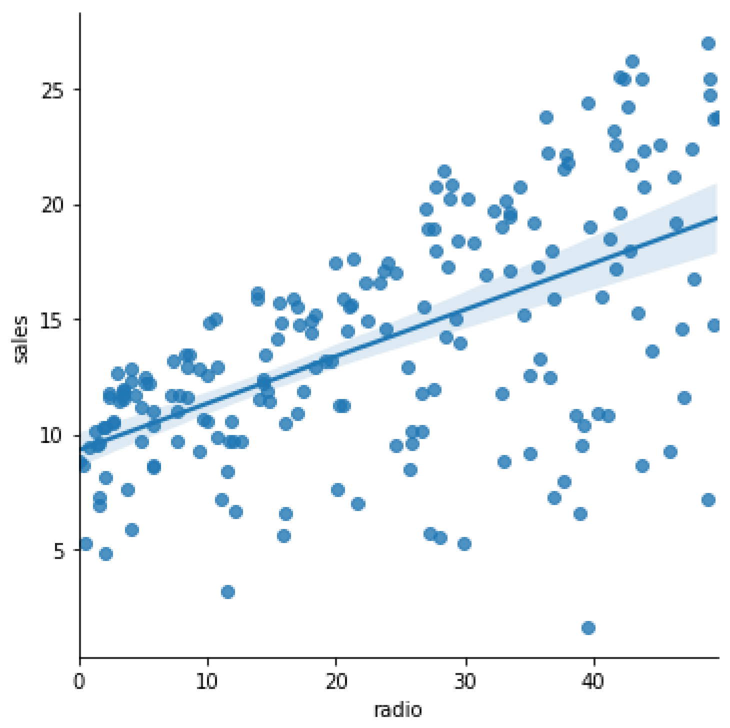
In [9]:
```
# TV vs sales
sns.lmplot(x="TV", y="sales", data = advertising)
```

Out[9]: `<seaborn.axisgrid.FacetGrid at 0x7f7d2f385d30>`



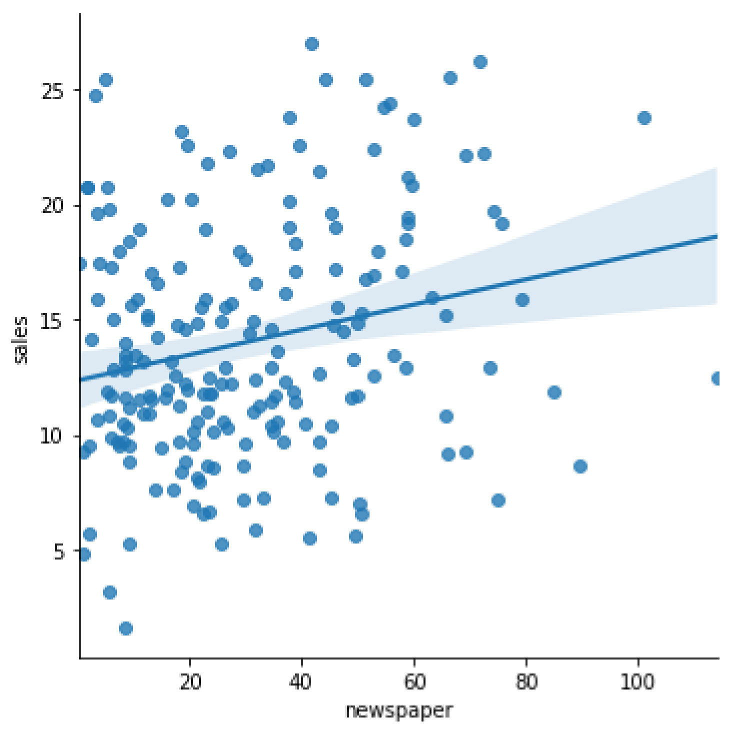
In [10]:
```
# radio vs sales
sns.lmplot(x="radio", y="sales", data = advertising)
```

Out[10]: `<seaborn.axisgrid.FacetGrid at 0x7f7d2f360250>`



In [11]:
```
# newspaper vs sales
sns.lmplot(x="newspaper", y="sales", data = advertising)
```

Out[11]: `<seaborn.axisgrid.FacetGrid at 0x7f7d2d2144f0>`



The relationship between the features and the predictor have to be linear. Hence, visually inspecting their scatter plots in order to check linearity.

ASSIGNMENT - 4

Finding the highest correlation factors

In [1]:
```python
# importing packages pandas, numpy, matplotlib and seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
# loading the dataset
airQuality = pd.read_excel('AirQualityUCI.xlsx')
```

In [3]:
```python
airQuality.head()
```

Out[3]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(
0	2004-03-10	18:00:00	2.6	1360.00	150	11.881723	1045.50	166.0	1056.25	113.0	1692.00	1267
1	2004-03-10	19:00:00	2.0	1292.25	112	9.397165	954.75	103.0	1173.75	92.0	1558.75	972
2	2004-03-10	20:00:00	2.2	1402.00	88	8.997817	939.25	131.0	1140.00	114.0	1554.50	1074
3	2004-03-10	21:00:00	2.2	1375.50	80	9.228796	948.25	172.0	1092.00	122.0	1583.75	1203
4	2004-03-10	22:00:00	1.6	1272.25	51	6.518224	835.50	131.0	1205.00	116.0	1490.00	1110

In [4]:
```python
airQuality.tail()
```

Out[4]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.
9352	2005-04-04	10:00:00	3.1	1314.25	-200	13.529605	1101.25	471.7	538.50	189.8	1374.25	1
9353	2005-04-04	11:00:00	2.4	1162.50	-200	11.355157	1027.00	353.3	603.75	179.2	1263.50	1
9354	2005-04-04	12:00:00	2.4	1142.00	-200	12.374538	1062.50	293.0	603.25	174.7	1240.75	1
9355	2005-04-04	13:00:00	2.1	1002.50	-200	9.547187	960.50	234.5	701.50	155.7	1041.00	
9356	2005-04-04	14:00:00	2.2	1070.75	-200	11.932060	1047.25	265.2	654.00	167.7	1128.50	

In [5]:
```python
#showing number of rows and columns in the dataset
airQuality.shape
```

Out[5]:
```
(9357, 17)
```

In [6]:
```python
# generating descriptive statistics
airQuality.describe()
```

Out[6]:

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.0
mean	-34.207524	1048.869652	-159.090093	1.865576	894.475963	168.604200	794.872333	58.135898	1391.363266	974.9
std	77.657170	329.817015	139.789093	41.380154	342.315902	257.424561	321.977031	126.931428	467.192382	456.9
min	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.0
25%	0.600000	921.000000	-200.000000	4.004958	711.000000	50.000000	637.000000	53.000000	1184.750000	699.7
50%	1.500000	1052.500000	-200.000000	7.886653	894.500000	141.000000	794.250000	96.000000	1445.500000	942.0
75%	2.600000	1221.250000	-200.000000	13.636091	1104.750000	284.200000	960.250000	133.000000	1662.000000	1255.2
max	11.900000	2039.750000	1189.000000	63.741476	2214.000000	1479.000000	2682.750000	339.700000	2775.000000	2522.7

In [7]:
```python
# deleting last two columns
airQuality.drop(airQuality.columns[[15, 16]], axis = 1, inplace = True)
```

In [8]:
```python
# generating descriptive statistics
airQuality.describe()
```

Out[8]:

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.0
mean	-34.207524	1048.869652	-159.090093	1.865576	894.475963	168.604200	794.872333	58.135898	1391.363266	974.9
std	77.657170	329.817015	139.789093	41.380154	342.315902	257.424561	321.977031	126.931428	467.192382	456.9
min	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.0
25%	0.600000	921.000000	-200.000000	4.004958	711.000000	50.000000	637.000000	53.000000	1184.750000	699.7
50%	1.500000	1052.500000	-200.000000	7.886653	894.500000	141.000000	794.250000	96.000000	1445.500000	942.0
75%	2.600000	1221.250000	-200.000000	13.636091	1104.750000	284.200000	960.250000	133.000000	1662.000000	1255.2
max	11.900000	2039.750000	1189.000000	63.741476	2214.000000	1479.000000	2682.750000	339.700000	2775.000000	2522.7

In [9]:
```python
#information about dataset
airQuality.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9357 entries, 0 to 9356
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           9357 non-null   datetime64[ns]
 1   Time           9357 non-null   object
 2   CO(GT)         9357 non-null   float64
 3   PT08.S1(CO)    9357 non-null   float64
 4   NMHC(GT)       9357 non-null   int64
 5   C6H6(GT)       9357 non-null   float64
 6   PT08.S2(NMHC)  9357 non-null   float64
 7   NOx(GT)        9357 non-null   float64
 8   PT08.S3(NOx)   9357 non-null   float64
 9   NO2(GT)        9357 non-null   float64
 10  PT08.S4(NO2)   9357 non-null   float64
 11  PT08.S5(O3)    9357 non-null   float64
 12  T              9357 non-null   float64
 13  RH             9357 non-null   float64
 14  AH             9357 non-null   float64
dtypes: datetime64[ns](1), float64(12), int64(1), object(1)
memory usage: 1.1+ MB
```

In [10]:
```python
# outliers in dataset are considered to be -200
airQuality.isin([-200]).any()
```

Out[10]:
```
Date             False
Time             False
CO(GT)            True
PT08.S1(CO)       True
NMHC(GT)          True
C6H6(GT)          True
PT08.S2(NMHC)     True
NOx(GT)           True
PT08.S3(NOx)      True
NO2(GT)           True
PT08.S4(NO2)      True
PT08.S5(O3)       True
T                 True
RH                True
AH                True
dtype: bool
```

In [11]:
```python
# replacing -200 with NaN values
airQuality.replace(to_replace = -200, value =np.nan,inplace=True)
```

In [12]:
```python
# checking and counting for missing data points for each column
airQuality.isnull().sum()
```

Out[12]:
```
Date                0
Time                0
CO(GT)           1683
PT08.S1(CO)       366
NMHC(GT)         8443
C6H6(GT)          366
PT08.S2(NMHC)     366
NOx(GT)          1639
PT08.S3(NOx)      366
NO2(GT)          1642
PT08.S4(NO2)      366
PT08.S5(O3)       366
T                 366
RH                366
AH                366
dtype: int64
```

In [13]:
```python
# replacing the null values with 0
airQuality.fillna(0,inplace=True)
```

In [14]:
```python
# now we can see all columns have zero missing values
airQuality.isnull().sum()
```

Out[14]:
```
Date             0
Time             0
CO(GT)           0
PT08.S1(CO)      0
NMHC(GT)         0
C6H6(GT)         0
PT08.S2(NMHC)    0
NOx(GT)          0
PT08.S3(NOx)     0
NO2(GT)          0
PT08.S4(NO2)     0
PT08.S5(O3)      0
T                0
RH               0
AH               0
dtype: int64
```

In [15]:
```python
# generating descriptive statistics
airQuality.describe()
```

Out[15]:

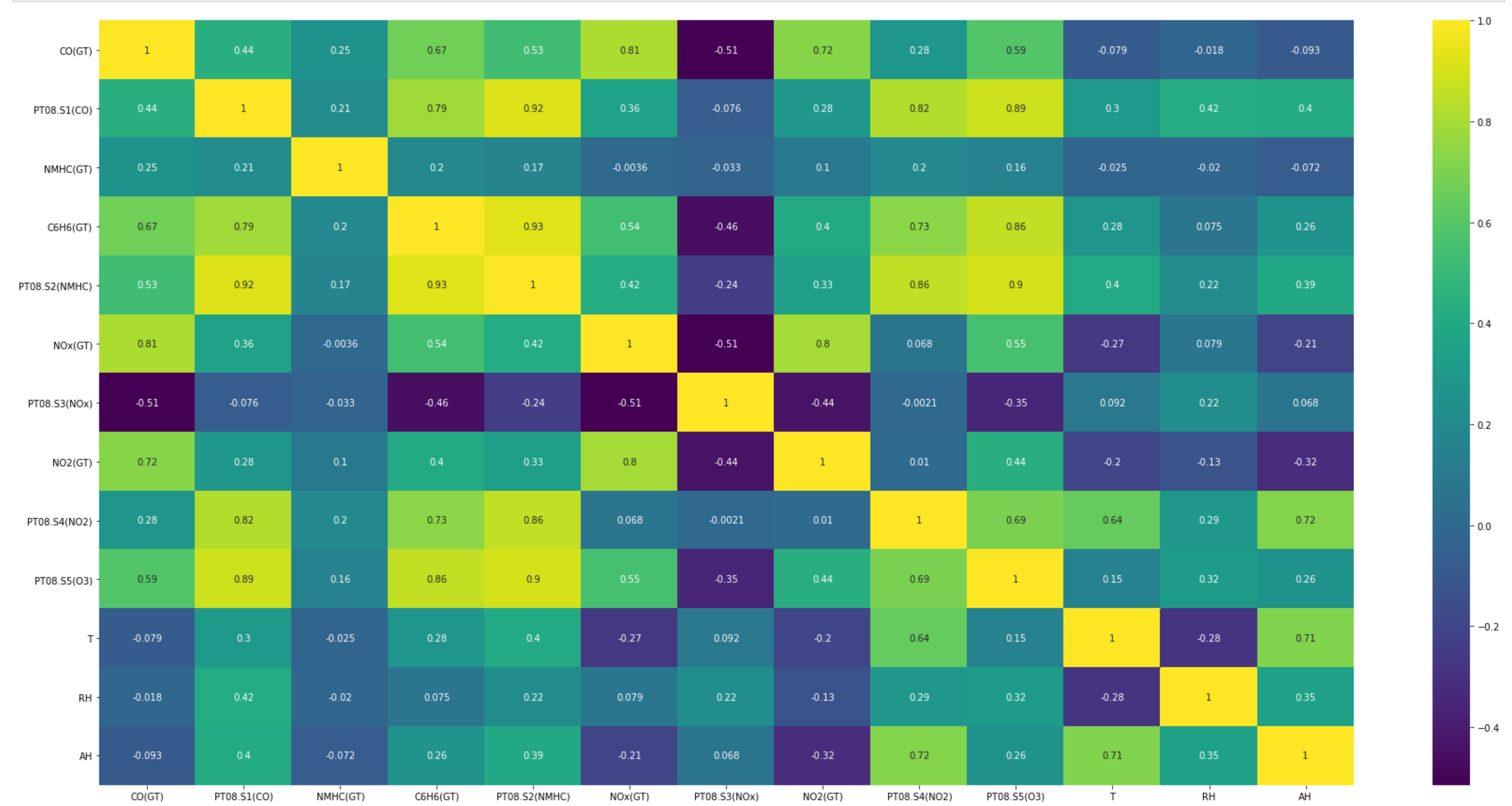
	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.0
mean	1.765545	1056.692672	21.373731	9.688596	902.298983	203.636796	802.695353	93.232617	1399.186287	982.7
std	1.554264	301.232260	91.103489	7.559609	318.681183	214.984126	299.341439	61.468588	441.442059	438.0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.600000	921.000000	0.000000	4.004958	711.000000	50.000000	637.000000	53.000000	1184.750000	699.7
50%	1.500000	1052.500000	0.000000	7.886653	894.500000	141.000000	794.250000	96.000000	1445.500000	942.0
75%	2.600000	1221.250000	0.000000	13.636091	1104.750000	284.200000	960.250000	133.000000	1662.000000	1255.2
max	11.900000	2039.750000	1189.000000	63.741476	2214.000000	1479.000000	2682.750000	339.700000	2775.000000	2522.7

In [16]:
```python
# finding correlations with other variables
corrMatrix = airQuality.corr()
corrMatrix
```

Out[16]:

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(
CO(GT)	1.000000	0.442803	0.249731	0.670790	0.533061	0.811449	-0.513070	0.723154	0.282080	0.586
PT08.S1(CO)	0.442803	1.000000	0.213250	0.786143	0.922093	0.356291	-0.075630	0.284508	0.823505	0.886
NMHC(GT)	0.249731	0.213250	1.000000	0.198346	0.170037	-0.003611	-0.033366	0.099541	0.196691	0.155
C6H6(GT)	0.670790	0.786143	0.198346	1.000000	0.926265	0.543665	-0.457762	0.402581	0.734014	0.862
PT08.S2(NMHC)	0.533061	0.922093	0.170037	0.926265	1.000000	0.419047	-0.240806	0.334108	0.855763	0.903
NOx(GT)	0.811449	0.356291	-0.003611	0.543665	0.419047	1.000000	-0.514602	0.795888	0.068429	0.553
PT08.S3(NOx)	-0.513070	-0.075630	-0.033366	-0.457762	-0.240806	-0.514602	1.000000	-0.440202	-0.002102	-0.352
NO2(GT)	0.723154	0.284508	0.099541	0.402581	0.334108	0.795888	-0.440202	1.000000	0.010185	0.439
PT08.S4(NO2)	0.282080	0.823505	0.196691	0.734014	0.855763	0.068429	-0.002102	0.010185	1.000000	0.694
PT08.S5(O3)	0.586753	0.886880	0.155224	0.862751	0.903060	0.553223	-0.352407	0.439057	0.694715	1.000
T	-0.079169	0.300361	-0.025172	0.275883	0.400037	-0.268696	0.092383	-0.195697	0.641935	0.149
RH	-0.018418	0.417492	-0.020121	0.074847	0.215377	0.079334	0.223613	-0.125245	0.291896	0.318
AH	-0.092964	0.403123	-0.071580	0.261013	0.393508	-0.210622	0.068493	-0.324221	0.719606	0.259

In [17]:
```python
# visualizing correlations with other variables
top_corr_feature=corrMatrix.index
# ploting heat map
plt.figure(figsize=(30,15))
g=sns.heatmap(airQuality[top_corr_feature].corr(),annot=True,cmap='viridis')
```



From the above correlation matrix we conclude that C6H6(GT) is highly correlated to PTO8.S2(NMHC) with 0.93 as its correlation value. Hence PTO8.S2(NMHC) can be considered as independent parameter in order to predict C6H6(GT) which would be the dependent parameter

In [18]:
```python
X = airQuality[['PT08.S2(NMHC)']] # taking independent parameter as X
y = airQuality['C6H6(GT)'] # taking dependent parameter as y
```

ASSIGNMENT - 5

Naive Bayes classifier - Probability Estimation:

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

P (PlayTennis = Yes) = 9/14
P (PlayTennis = No) = 5/14

OUTLOOK:
P (Outlook=Sunny | PlayTennis=Yes) = 2/9
P (Outlook=Sunny | PlayTennis=No) = 3/5
P (Outlook=Overcast | PlayTennis=Yes) = 4/9
P (Outlook=Overcast | PlayTennis=No) = 0/5
P (Outlook=Rain | PlayTennis=Yes) = 3/9
P (Outlook=Rain | PlayTennis=No) = 2/5

TEMPERATURE:
P (Temperature=Hot | PlayTennis=Yes) = 2/9

P (Temperature=Hot | PlayTennis=No) = 2/5
P (Temperature=Mild | PlayTennis=Yes) = 4/9
P (Temperature=Mild | PlayTennis=No) = 2/5
P (Temperature=Cool | PlayTennis=Yes) = 3/9
P (Temperature=Cool | PlayTennis=No) = 1/5

HUMIDITY:
P (Humidity=High | PlayTennis=Yes) = 3/9
P (Humidity=High | PlayTennis=No) = 4/5
P (Humidity=Normal | PlayTennis=Yes) = 6/9
P (Humidity=Normal | PlayTennis=No) = 1/5

WIND:
P (Wind=Weak | PlayTennis=Yes) = 6/9
P (Wind=Weak | PlayTennis=No) = 2/5
P (Wind=Strong | PlayTennis=Yes) = 3/9
P (Wind=Strong | PlayTennis=No) = 3/5

Considering an instance
x' = {Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong}

Therefore, **P(Yes | x')** = P (PlayTennis = Yes) * P (Outlook=Sunny | PlayTennis=Yes) * P (Temperature=Cool | PlayTennis=Yes) * P (Humidity=High | PlayTennis=Yes) * P (Wind=Strong | PlayTennis=Yes)
= 9/14 * 2/9 * 3/9 * 3/9 * 3/9
= 1/189
= 0.00529100529 -------------------------------- (I)

And, **P(No | x')** = P (PlayTennis = No) * P (Outlook=Sunny | PlayTennis=No) * P (Temperature=Cool | PlayTennis=No) * P (Humidity=High | PlayTennis=No) * P (Wind=Strong | PlayTennis=No)
= 5/14 * 3/5 * 1/5 * 4/5 * 3/5
= 18/875
= 0.02057142857 -------------------------------- (II)

From, (I) and (II), we see that, P(Yes | x') < P(No | x')
Hence, for the instance **x'** = {Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong} we conclude that not playing tennis is the maximum likelihood hypothesis.

Similarly, considering another instance
y' = {Outlook=Overcast, Temperature=Cool, Humidity=High, Wind=Strong}

Therefore, **P(Yes | y')** = P (PlayTennis = Yes) * P (Outlook= Overcast | PlayTennis=Yes) * P (Temperature=Cool | PlayTennis=Yes) * P (Humidity=High | PlayTennis=Yes) * P (Wind=Strong | PlayTennis=Yes)

= 9/14 * 4/9 * 3/9 * 3/9 * 3/9
= 2/189
= 0.01058201058 -------------------------------- (III)

And, **P(No | y')** = P (PlayTennis = No) * P (Outlook= Overcast | PlayTennis=No) * P (Temperature=Cool | PlayTennis=No) * P (Humidity=High | PlayTennis=No) * P (Wind=Strong | PlayTennis=No)

= 5/14 * 0/5 * 1/5 * 4/5 * 3/5
= 0 -------------------------------- (IV)

From, (III) and (IV), we see that, P(Yes | y') > P(No | y')
Hence, for the instance y' = {Outlook=Overcast, Temperature=Cool, Humidity=High, Wind=Strong} we conclude that playing tennis is the maximum likelihood hypothesis.
