INTRODUCTION

- What is Regression?
- Types of Regression
- What is Linear Regression?
- Linear Regression Terminologies
- Advantages And Disadvantages Of Linear Regression
- Linear Regression Use Cases
- Use case – Linear Regression Implementation

❖ What is regression?

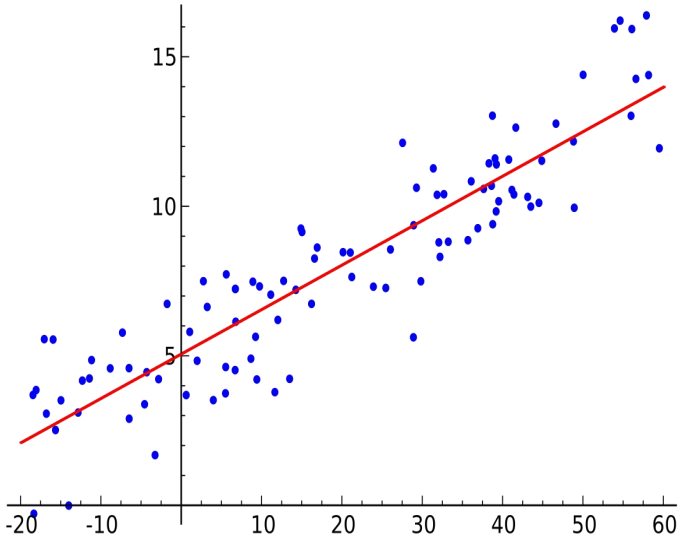
The main goal of regression is the construction of an efficient model to predict the dependent attributes from a bunch of attribute variables. A regression problem is when the output variable is either real or a continuous value i.e salary, weight, area, etc.

We can also define regression as a statistical means that is used in applications like housing, investing, etc. It is used to predict the relationship between a dependent variable and a bunch of independent variables. Let us take a look at various types of regression techniques.

❖ What is linear regression?

The main goal of regression is the construction of an efficient model to predict the dependent attributes from a bunch of attribute variables. A regression problem is when the output variable is either real or a continuous value i.e salary, weight, area, etc.

We can also define regression as a statistical means that is used in applications like housing, investing, etc. It is used to predict the relationship between a dependent variable and a bunch of independent variables. Let us take a look at various types of regression techniques.



Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable. The red line in the above graph is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best. The line can be modelled based on the linear equation shown below.

y = a_0 + a_1 * x ## Linear Equation

The motive of the linear regression algorithm is to find the best values for a_0 and a_1. Before moving on to the algorithm, let's have a look at two important concepts you must know to better understand linear regression.

❖ Cost Function

The cost function helps us to figure out the best possible values for a_0 and a_1 which would provide the best fit line for the data points. Since we want the best values for a_0 and a_1, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value.

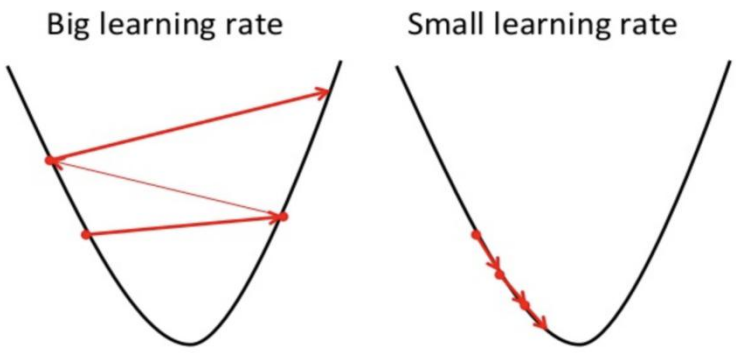
$$minimize \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

We choose the above function to minimize. The difference between the predicted values and ground truth measures the error difference. We square the error difference and sum over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points. Therefore, this cost function is also known as the Mean Squared Error(MSE) function. Now, using this MSE function we are going to change the values of a_0 and a_1 such that the MSE value settles at the minima.

❖ Gradient Descent

The next important concept needed to understand linear regression is gradient descent. Gradient descent is a method of updating a_0 and a_1 to reduce the cost function(MSE). The idea is that we start with some values for a_0 and a_1 and then we change these values iteratively to reduce the cost. Gradient descent helps us on how to change the values.



To draw an analogy, imagine a pit in the shape of U and you are standing at the topmost point in the pit and your objective is to reach the bottom of the pit. There is a catch, you can only take a discrete number of steps to reach the bottom. If you decide to take one step at a time you would eventually reach the bottom of the pit but this would take a longer time. If you choose to take longer steps each time, you would reach sooner but, there is a chance that you could overshoot the bottom of the pit and not exactly at the bottom. In the gradient

descent algorithm, the number of steps you take is the learning rate. This decides on how fast the algorithm converges to the minima.

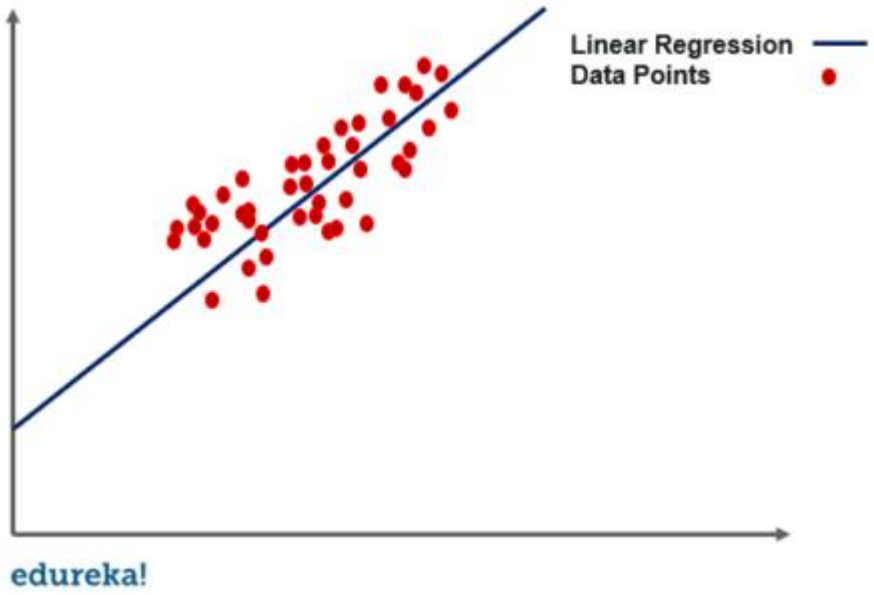
❖ Types Of Regression

The following are types of regression.

1. **Simple Linear Regression**
2. **Polynomial Regression**
3. **Support Vector Regression**
4. **Decision Tree Regression**
5. **Random Forest Regression**

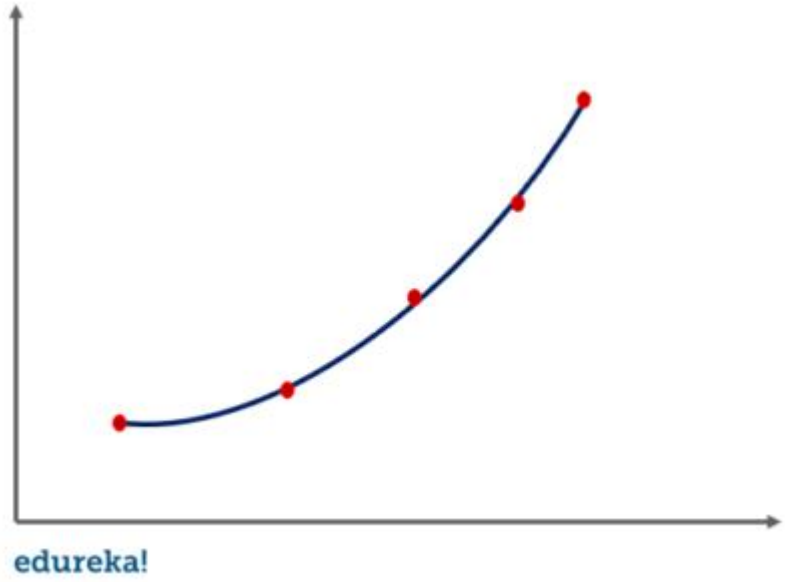
❖ Simple Linear Regression

One of the most interesting and common regression technique is simple linear regression. In this, we predict the outcome of a dependent variable based on the independent variables, the relationship between the variables is linear. Hence, the word linear regression.



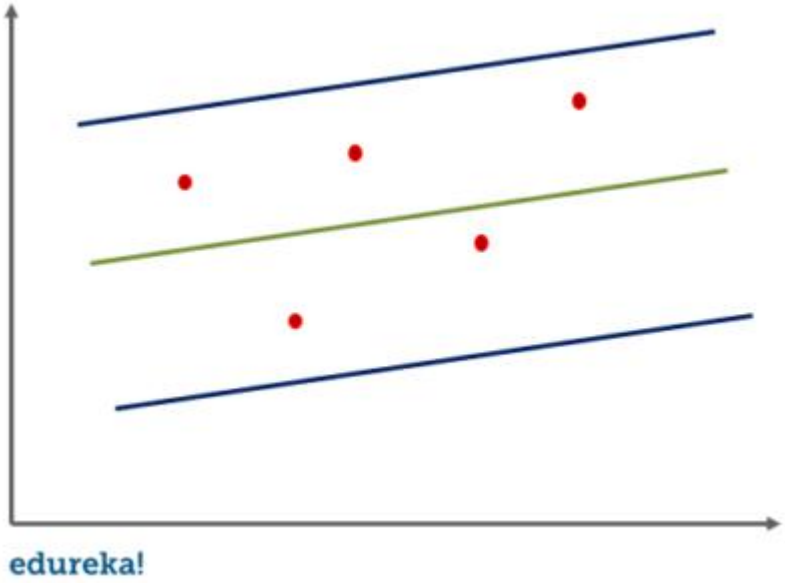
❖ Polynomial Regression

In this regression technique, we transform the original features into polynomial features of a given degree and then perform regression on it.



❖ Support Vector Regression

For support vector machine regression or SVR, we identify a hyperplane with maximum margin such that the maximum number of data points are within those margins. It is quite similar to the support vector machine classification algorithm.



❖ Decision Tree Regression

A decision tree can be used for both regression and classification. In the case of regression, we use the ID3 algorithm(Iterative Dichotomiser 3) to identify the splitting node by reducing the standard deviation.

❖ Random Forest Regression

In random forest regression, we ensemble the predictions of several decision tree regressions. Now that we know about different types of regression let us take a look at simple linear regression in detail.

❖ Advantages And Disadvantages

Advantages	Disadvantages
Linear regression performs exceptionally well for linearly separable data	The assumption of linearity between dependent and independent variables
Easier to implement, interpret and efficient to train	It is often quite prone to noise and overfitting
It handles overfitting pretty well using dimensionally reduction techniques, regularization, and cross-validation	Linear regression is quite sensitive to outliers
One more advantage is the extrapolation beyond a specific data set	It is prone to multicollinearity

Use Case – Implementing Linear Regression

The process takes place in the following steps:

1. Importing Libraries and Packages
2. Loading and Viewing Data Set
3. Plotting and Visualizing Data
4. Modeling and Predicting with sklearn

1. Importing Libraries and Packages

We will use these packages to help us manipulate the data and visualize the features/labels as well as measure how well our model performed. Numpy and Pandas are helpful for manipulating the dataframe and its columns and cells. We will use matplotlib along with Seaborn to visualize our data.

In [ ]:
```
###Import libraries
```

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
import time
import datetime
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
plt.style.use('ggplot')
```

2. Loading and Viewing Data Set

With Pandas, we can load both the training and testing set that we wil later use to train and test our model. Before we begin, we should take a look at our data table to see the values that we'll be working with. We can use the head and describe function to look at some sample data and statistics.

```
In [ ]: ###Read the data

In [2]: def date_2_time_stamp(string):
          return time.mktime(datetime.datetime.strptime(string,
                                                        "%Y-%m-%d").timetuple())

In [3]: data_read = pd.read_csv('C:/Users/hp/Downloads/new data.csv/NEW DATA.csv',sep=' ')

In [4]: data_read.head(2)
```

Out[4]:

	1:Date	2:Time	3:Temperature_Comedor_Sensor	4:Temperature_Habitacion_Sensor	5:Weather_Temperature	6:CO2_Comedor_Sensor	7:CO2_

```
In [5]: data_read.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2764 entries, 0 to 2763
Data columns (total 24 columns):
1:Date                          2764 non-null object
2:Time                          2764 non-null object
3:Temperature_Comedor_Sensor    2764 non-null float64
4:Temperature_Habitacion_Sensor 2764 non-null float64
5:Weather_Temperature           2764 non-null float64
6:CO2_Comedor_Sensor            2764 non-null float64
7:CO2_Habitacion_Sensor         2764 non-null float64
8:Humedad_Comedor_Sensor        2764 non-null float64
9:Humedad_Habitacion_Sensor     2764 non-null float64
10:Lighting_Comedor_Sensor      2764 non-null float64
11:Lighting_Habitacion_Sensor   2764 non-null float64
12:Precipitacion                2764 non-null float64
13:Meteo_Exterior_Crepusculo    2764 non-null float64
14:Meteo_Exterior_Viento        2764 non-null float64
15:Meteo_Exterior_Sol_Oest      2764 non-null float64
16:Meteo_Exterior_Sol_Est       2764 non-null float64
17:Meteo_Exterior_Sol_Sud       2764 non-null float64
18:Meteo_Exterior_Piranometro   2764 non-null float64
19:Exterior_Entalpic_1          2764 non-null int64
20:Exterior_Entalpic_2          2764 non-null int64
21:Exterior_Entalpic_turbo      2764 non-null int64
22:Temperature_Exterior_Sensor  2764 non-null float64
23:Humedad_Exterior_Sensor      2764 non-null float64
24:Day_Of_Week                  2764 non-null float64
dtypes: float64(19), int64(3), object(2)
memory usage: 518.4+ KB
```

```
In [9]: data_read.head(5)
```

Out[9]:

	1:Date	2:Time	3:Temperature_Comedor_Sensor	4:Temperature_Habitacion_Sensor	5:Weather_Temperature	6:CO2_Comedor_Sensor	7:CO2_Habitacior
0	13/03/2012	11:45	18.1875	17.8275	0.0	216.560	
1	13/03/2012	12:00	18.4633	18.1207	6.8	219.947	
2	13/03/2012	12:15	18.7673	18.4367	17.0	219.403	
3	13/03/2012	12:30	19.0727	18.7513	18.0	218.613	
4	13/03/2012	12:45	19.3721	19.0414	20.0	217.714	

5 rows × 24 columns

```
In [10]: data_read.tail()
```

Out[10]:

	1:Date	2:Time	3:Temperature_Comedor_Sensor	4:Temperature_Habitacion_Sensor	5:Weather_Temperature	6:CO2_Comedor_Sensor	7:CO2_Habita
2759	11/04/2012	05:30	21.1520	20.8187	13.0000	190.539	
2760	11/04/2012	05:45	21.0413	20.7053	12.1333	190.421	
2761	11/04/2012	06:00	20.9347	20.5827	12.0000	190.432	
2762	11/04/2012	06:15	20.8560	20.5200	12.0000	191.531	
2763	11/04/2012	06:30	20.7627	20.4400	12.1333	191.563	

5 rows × 24 columns

In [11]:
```
data_read.isna().sum()
```

Out[11]:
```
1:Date                          0
2:Time                          0
3:Temperature_Comedor_Sensor    0
4:Temperature_Habitacion_Sensor 0
5:Weather_Temperature           0
6:CO2_Comedor_Sensor            0
7:CO2_Habitacion_Sensor         0
8:Humedad_Comedor_Sensor        0
9:Humedad_Habitacion_Sensor     0
10:Lighting_Comedor_Sensor      0
11:Lighting_Habitacion_Sensor   0
12:Precipitacion                0
13:Meteo_Exterior_Crepusculo    0
14:Meteo_Exterior_Viento        0
15:Meteo_Exterior_Sol_Oest      0
16:Meteo_Exterior_Sol_Est       0
17:Meteo_Exterior_Sol_Sud       0
18:Meteo_Exterior_Piranometro   0
19:Exterior_Entalpic_1          0
20:Exterior_Entalpic_2          0
21:Exterior_Entalpic_turbo      0
22:Temperature_Exterior_Sensor  0
23:Humedad_Exterior_Sensor      0
24:Day_Of_Week                  0
dtype: int64
```

In [12]:
```
data_read.dtypes
```

Out[12]:
```
1:Date                          object
2:Time                          object
3:Temperature_Comedor_Sensor    float64
4:Temperature_Habitacion_Sensor float64
5:Weather_Temperature           float64
6:CO2_Comedor_Sensor            float64
7:CO2_Habitacion_Sensor         float64
8:Humedad_Comedor_Sensor        float64
9:Humedad_Habitacion_Sensor     float64
10:Lighting_Comedor_Sensor      float64
11:Lighting_Habitacion_Sensor   float64
12:Precipitacion                float64
13:Meteo_Exterior_Crepusculo    float64
14:Meteo_Exterior_Viento        float64
15:Meteo_Exterior_Sol_Oest      float64
16:Meteo_Exterior_Sol_Est       float64
17:Meteo_Exterior_Sol_Sud       float64
18:Meteo_Exterior_Piranometro   float64
19:Exterior_Entalpic_1            int64
20:Exterior_Entalpic_2            int64
21:Exterior_Entalpic_turbo        int64
22:Temperature_Exterior_Sensor  float64
23:Humedad_Exterior_Sensor      float64
24:Day_Of_Week                  float64
dtype: object
```

In [13]:
```
print(type(data_read))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [14]:
```
data_read.columns=['Date','Time','Temperature_Comedor_Sensor ','Temperature_Habitacion_Sensor','Weather_Temperature','CO2_Com
```

In [15]:
```
data_read.Day_Of_Week.value_counts()
```

Out[15]:
```
2.00000    429
3.00000    406
1.00000    380
7.00000    380
6.00000    380
5.00000    380
4.00000    380
2.06667      5
1.06667      4
3.06667      4
4.06667      4
6.60000      4
5.06667      4
6.06667      4
Name: Day_Of_Week, dtype: int64
```

```
In [16]: data_read.groupby('Day_Of_Week').mean()
```

Out[16]:

Day_Of_Week	Temperature_Comedor_Sensor	Temperature_Habitacion_Sensor	Weather_Temperature	CO2_Comedor_Sensor	CO2_Habitacion_Sensor	Humec
1.00000	20.035636	19.533464	14.329649	216.486705	217.141566	
1.06667	19.614825	19.108500	13.000000	203.541250	206.882750	
2.00000	19.089801	18.886498	14.911111	205.331795	207.361457	
2.06667	18.475600	18.276260	12.293340	205.657600	208.358400	
3.00000	18.377144	18.158860	13.185715	216.770165	222.821828	
3.06667	18.250525	17.898325	10.116675	208.413500	210.773250	
4.00000	18.715573	18.377350	13.024212	208.132495	211.258161	
4.06667	18.583150	18.047500	9.750000	206.290500	209.725250	
5.00000	18.746681	18.272584	13.234387	205.372358	207.298168	
5.06667	19.247675	18.731325	13.100000	206.701500	209.909250	
6.00000	19.352329	18.918054	14.279298	203.606966	205.981505	
6.06667	19.615325	19.129675	12.033342	207.653250	212.096000	
6.60000	20.506350	19.956675	12.500000	204.061500	207.184000	
7.00000	20.154353	19.672804	14.392106	203.678676	205.410579	

14 rows × 21 columns

```
In [19]: # AXIS 1 means column wise
         data_read = data_read.drop(['Date','Time','Exterior_Entalpic_1','Exterior_Entalpic_2','Exterior_Entalpic_turbo'], axis = 1)
```

```
In [20]: data_read
```

Out[20]:

	Temperature_Comedor_Sensor	Temperature_Habitacion_Sensor	Weather_Temperature	CO2_Comedor_Sensor	CO2_Habitacion_Sensor	Humedad_Come
0	18.1875	17.8275	0.0000	216.560	221.920	
1	18.4633	18.1207	6.8000	219.947	220.363	
2	18.7673	18.4367	17.0000	219.403	218.933	
3	19.0727	18.7513	18.0000	218.613	217.045	
4	19.3721	19.0414	20.0000	217.714	216.080	
...	...	...	...	...	...	
2759	21.1520	20.8187	13.0000	190.539	192.181	
2760	21.0413	20.7053	12.1333	190.421	193.067	
2761	20.9347	20.5827	12.0000	190.432	193.653	
2762	20.8560	20.5200	12.0000	191.531	193.387	
2763	20.7627	20.4400	12.1333	191.563	193.664	

2764 rows × 19 columns

```
In [21]: data_read.Day_Of_Week.unique()
```

```
Out[21]: array([2.     , 2.06667, 3.     , 3.06667, 4.     , 4.06667, 5.     ,
                5.06667, 6.     , 6.06667, 7.     , 6.6    , 1.     , 1.06667])
```

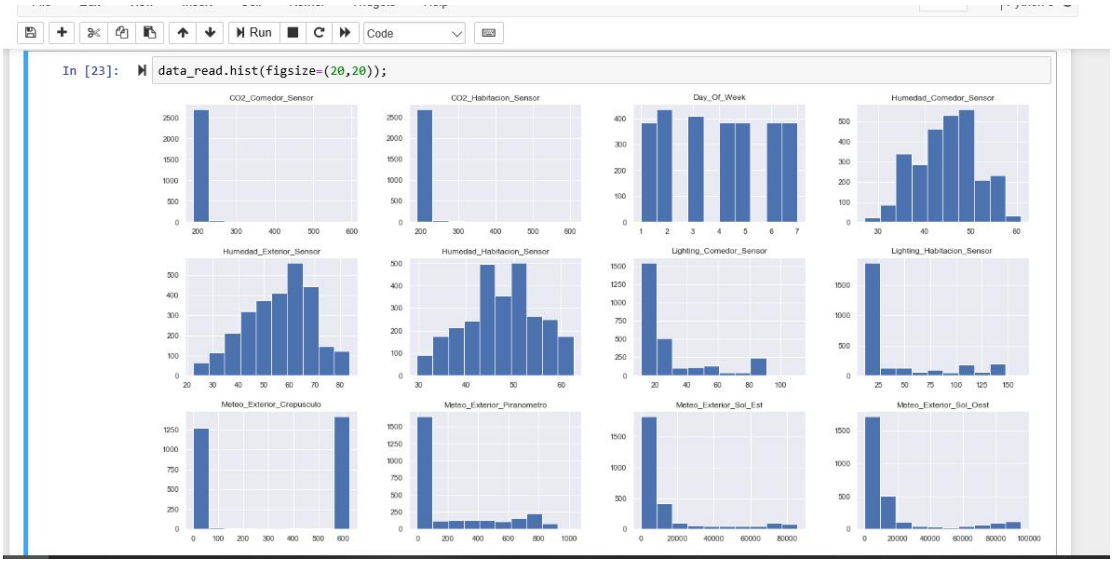
3. Plotting and Visualizing Data

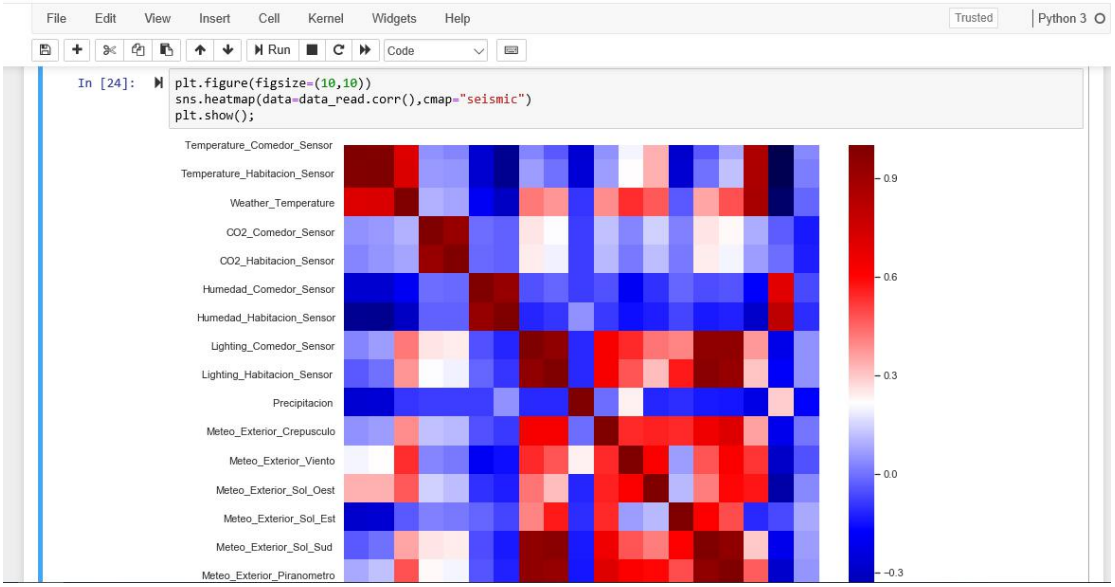
Plotting the data with matplotlib scatter

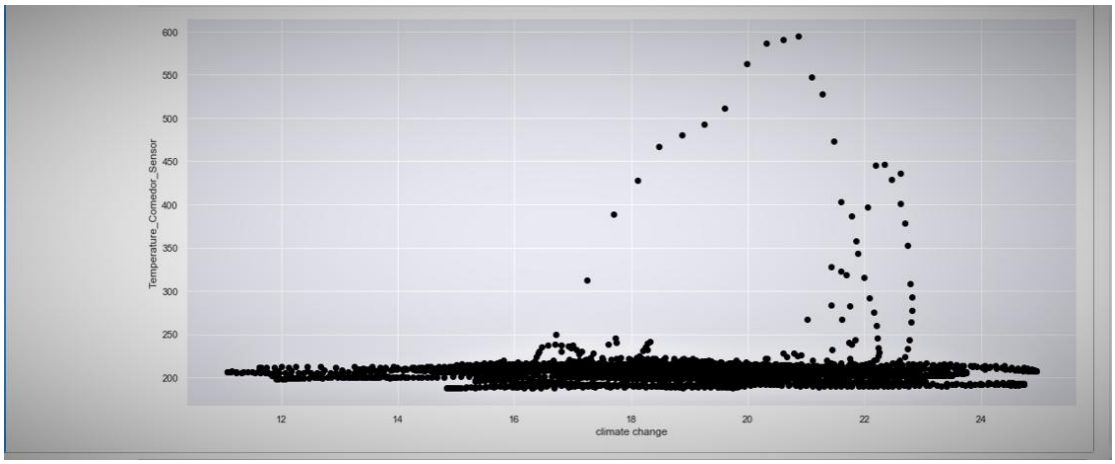
```
In [22]:
wordcloud = WordCloud().generate(data_read.to_string())

sns.set(rc={'figure.figsize':(12,8)})

# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```







4 Model Fitting, Optimizing, and Predicting

Now that our data has been processed and formmated properly, and that we understand the general data we're working with as well as the trends and associations, we can start to build our model. We can import different classifiers from sklearn.

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
In [26]: from sklearn.feature_selection import RFE

In [27]: data_copy = data_read.copy()

In [28]: X = data_copy.drop(['Weather_Temperature'], axis=1)

In [29]: X
```

Out[29]:

	Temperature_Comedor_Sensor	Temperature_Habitacion_Sensor	CO2_Comedor_Sensor	CO2_Habitacion_Sensor	Humedad_Comedor_Sensor	Humedad_
0	18.1875	17.8275	216.560	221.920	39.9125	
1	18.4633	18.1207	219.947	220.363	39.9267	
2	18.7673	18.4367	219.403	218.933	39.7720	
3	19.0727	18.7513	218.613	217.045	39.7760	
4	19.3721	19.0414	217.714	216.080	39.7757	
...	...	...	...	...	...	...
2759	21.1520	20.8187	190.539	192.181	41.3120	
2760	21.0413	20.7053	190.421	193.067	41.3173	
2761	20.9347	20.5827	190.432	193.653	41.3333	
2762	20.8560	20.5200	191.531	193.387	41.3093	
2763	20.7627	20.4400	191.563	193.664	41.2800	

2764 rows × 18 columns

```
In [30]: y = data_read['Day_Of_Week']

In [31]: X = X.values

In [32]: y = y.values

In [33]: y
Out[33]: array([2., 2., 2., ..., 3., 3., 3.])

In [34]: from sklearn import linear_model

In [35]: type(X)
Out[35]: numpy.ndarray

In [36]: X.shape
Out[36]: (2764, 18)

In [37]: y.shape
Out[37]: (2764,)
```

```
In [46]: X_train = X[:round_length_training]

In [47]: X_test = X[round_length_training:]

In [48]: X_test.shape
Out[48]: (553, 18)

In [49]: X_train.shape
Out[49]: (2211, 18)

In [50]: y_train = y[:round_length_training]

In [51]: y_train.shape
Out[51]: (2211,)

In [52]: y_test = y[round_length_training:]

In [53]: y_test.shape
Out[53]: (553,)

In [54]: regressor = LinearRegression()
```

```
In [56]: regressor.fit(X_train, y_train) #training the algorithm

Out[56]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [57]: y_train

Out[57]: array([2., 2., 2., ..., 4., 4., 4.])

In [58]: X_train

Out[58]: array([[ 18.1875,  17.8275, 216.56  , ...,  18.115 ,  48.375 ,   2.    ],
                [ 18.4633,  18.1207, 219.947 , ...,  18.4147,  47.808 ,   2.    ],
                [ 18.7673,  18.4367, 219.403 , ...,  18.8533,  47.432 ,   2.    ],
                ...,
                [ 20.5453,  20.584 , 202.261 , ...,  20.4467,  66.2   ,   4.    ],
                [ 20.7187,  20.7747, 202.763 , ...,  20.636 ,  65.4853,   4.    ],
                [ 20.912 ,  21.0107, 201.867 , ...,  20.7467,  64.7893,   4.    ]])

In [59]: print(X_train)

[[ 18.1875  17.8275 216.56   ...  18.115   48.375    2.    ]
 [ 18.4633  18.1207 219.947  ...  18.4147  47.808    2.    ]
 [ 18.7673  18.4367 219.403  ...  18.8533  47.432    2.    ]
 ...
 [ 20.5453  20.584  202.261  ...  20.4467  66.2      4.    ]
 [ 20.7187  20.7747 202.763  ...  20.636   65.4853   4.    ]
 [ 20.912   21.0107 201.867  ...  20.7467  64.7893   4.    ]]
```

```
In [60]: print(regressor.intercept_)

8.08242361927114e-14

In [61]: print(regressor.coef_)

[ 1.17476044e-15 -1.56377001e-15 -4.11942961e-16  5.31488715e-17
 -1.10505821e-16  2.10194677e-17  4.15683929e-17  6.23978578e-18
 -2.38941033e-15  2.02518936e-17  4.58783234e-16 -7.22633212e-20
 -2.61611289e-19  2.53321256e-20 -1.82462791e-17  1.77606356e-16
 -8.44425534e-18  1.00000000e+00]

In [62]: y_pred = regressor.predict(X_test)

In [63]: y_pred

Out[63]: array([4.     , 4.     , 4.     , 4.     , 4.     , 4.     , 4.     ,
                4.     , 4.     , 4.     , 4.     , 4.     , 4.     , 4.     ,
                4.     , 4.     , 4.     , 4.     , 4.     , 4.     , 4.     ,
                4.     , 4.     , 4.     , 4.     , 4.     , 4.     , 4.     ,
                4.     , 4.     , 4.     , 4.     , 4.     , 4.     , 4.     ,
                4.     , 4.     , 4.     , 4.     , 4.     , 4.     , 4.     ,
                4.     , 4.     , 4.     , 4.     , 4.06667, 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
                5.     , 5.     , 5.     , 5.     , 5.     , 5.     , 5.     ,
```

```
In [70]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

         Mean Absolute Error: 7.45315001196823e-15
         Mean Squared Error: 7.88159239836282e-29
         Root Mean Squared Error: 8.877833293300128e-15

In [71]: model=LinearRegression()

In [72]: model.fit(X_train,y_train)
         y_pred=model.predict(X_test)
         print("Training Accuracy:",model.score(X_train,y_train))
         print("Testing Accuracy:",model.score(X_test,y_test))

         Training Accuracy: 1.0
         Testing Accuracy: 1.0
```