## **8LAB SHEET – 3**

## **AIR QUALITY ANALYSIS**

In this project we have to analyse data on air quality in one of the cities in Italy. We need to determine whether the concentration of air pollutants increase or decrease over time.

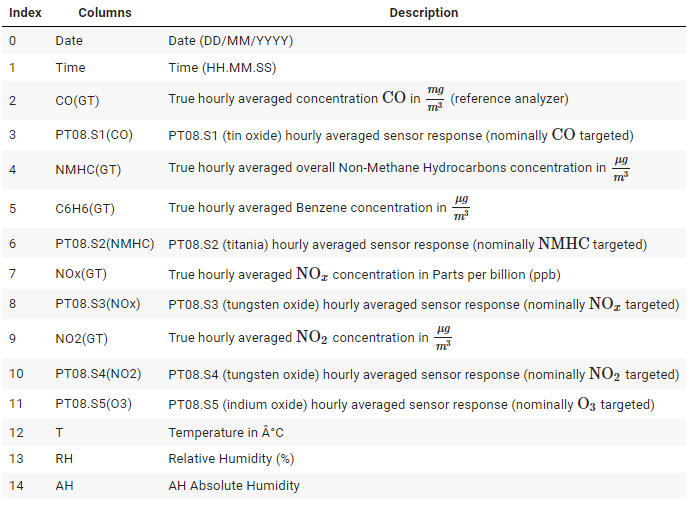
In the process, we will also learn how to work with time-series data, i.e, how to work with date and time values. Towards the end of the project, we need to create time-series plots such as line plots, bar plots and multivariate boxplots to observe the trend of air pollution concentration over time.

Now, skim through the data description to understand the kind of features and corresponding values we have in this data set on air quality.

**Data Description**

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device.

The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field-deployed air quality chemical sensor devices responses. A co-located reference certified analyzer provided **Ground Truth (GT)** hourly averaged concentrations for Carbon Monoxide (CO), Non-Methane Hydrocarbons (NMHC), Benzene (C6H6), total Nitrogen Oxides (NOx) & Nitrogen Dioxide (NO2) and Ozone (O3). These chemical compounds are commonly occurring air pollutants.

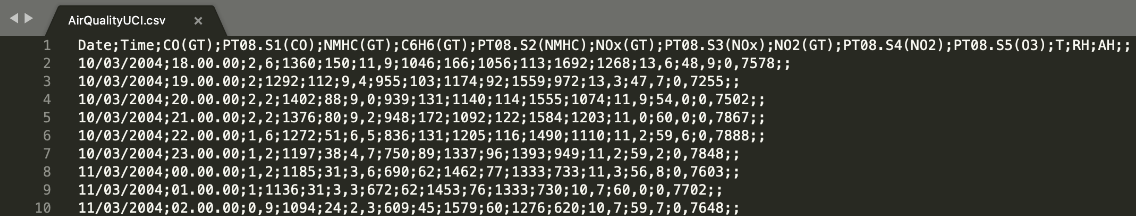


#### **Activity 1: Loading Data**

Let's begin with importing the required module and importing the time-series dataset on air quality. Here's the dataset link:

<https://student-datasets-bucket.s3.ap-south-1.amazonaws.com/whitehat-ds-datasets/air-quality/AirQualityUCI.csv>

The values in the dataset are separated by semi-columns (;) instead of commas. The image below shows the first 10 lines of the CSV file



To read the given CSV file, pass the sep = ';' parameter inside the read\_csv() function along with the file location.

**Q.1. Import the required modules and load the time-series dataset on air quality. Also, display the first five rows.**

The dataset contains two unnamed columns. They are Unnamed: 15 and Unnamed: 16. Both the columns contain NaN or null values. We need to drop them. Also, few of the columns contain numeric values separated by comma. We need to replace the commas with periods (or dots).

Let's first look at the complete information on the  Data Frame.

**Q.2. Apply the 'info()' function on the 'df' DataFrame.**

The values in the Date and Time columns need to be converted to the datetime values. Additionally, the values in the CO(GT), C6H6(GT), T, RH and AH columns need to be converted to float-point numbers.

#### **Activity 2: Missing Values**

We already know that the Unnamed: 15 and Unnamed: 16 columns contain the missing (or null) values. Let's find out if more columns also contain the null values.

**Q.3 Check for the missing values in the 'df' DataFrame.**

All the values contained in the Unnamed: 15 and Unnamed: 16 columns are the missing (or null) values. The other columns contain 114 null values. So, let's drop the last two columns using the drop() function.

It removes rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. In this case, we will specify the column names.

**Q.4 Drop the 'Unnamed: 15 and Unnamed: 16' columns from the 'df' DataFrame.**

The axis=1 parameter tells Python to drop columns. In this case, if you don't pass the axis=1 parameter, Python will drop the given columns because we have specified the column names to be dropped. Nonetheless, let's follow a good habit of specifying whether we want to remove a column or a row.

Now, let's verify whether the Unnamed: 15 and Unnamed: 16 columns are removed or not from the df DataFrame by printing all the column names.

**Q.5 Get the list of columns present in the 'df' DataFrame after removing the 'Unnamed: 15 and Unnamed: 16' columns.**

We know that the other columns have 114 missing values. Let's see which rows in the Date column are missing.

**Q.6 Print the rows missing in the 'Date' column.**

We can see that all the rows starting from index 9357 to 9470 contain missing values in all the columns.

**The dropna() Function**

Let's drop them all using the dropna() function. It removes the rows or columns containing the missing values. If no parameter is passed to the dropna() function, then it removes all the rows containing at least one null value.

**Q.7 Drop the rows containing at least one null value in the 'df' DataFrame using the 'dropna()' function.**

**Q.8 Check whether all the missing values are removed or not.**

**Q.9 How many rows and columns are in the DataFrame after removing the rows and columns containing the missing values?**

**Q.10 Print the last 5 rows of the 'df' DataFrame.**

#### **Activity 3: Parsing datetime Values - pd.to\_datetime() Function**

Now, we need to convert the values stored in the Date and Time columns to datetime values so that later we can sort the DataFrame in the chronological order. A datetime value is a combination of date and time.

Let's create a new Pandas series that is a concatenation of the Date and Time series. In other words, let's concatenate the values stored in the Date and Time columns and store them into a new Pandas series.

We will then convert these concatenated values to the datetime values using the pd.to\_datetime() function from the Pandas module. The input to the pd.to\_datetime() function is the Pandas series whose values are to be converted into datetime values.

**Syntax:** pd.to\_datetime(series)

where the series is the Pandas series whose values are to be converted into datetime values.

The default format of a datetime value is YYYY-MM-DD where YYYY denotes year, MM denotes month and DD denotes day.

But first we will have to replace the periods (or dots) with colons (:) in the values of the Time column because the time values are separated colons, i.e., HH:MM:SS where

* HH denotes hours
* MM denotes minutes
* SS denotes seconds

**Q.11 Concatenate the values stored in the 'Date' and 'Time' columns and store them into a new Pandas series.**

#### **Activity4:Parsing datetime Values- date(),strftime() & time() Function**

Let's learn how to extract date, day, month, year, time, hours, minutes and seconds from a datetime value.

**The date() Function**

In general, a datetime value is a Timestamp object. It holds both date and time.

**Q.12 Get the first 'datetime' value from the 'df\_series' and store it in the 'timestamp\_0' variable. Print its value.**

To get a date (i.e., datetime.date object) from a timestamp, apply the date() function.

**Q.13 Get date from the timestamp stored in the 'timestamp\_0' variable. Store the output in the 'date\_0' variable. Print the value and its data-type.**

**The day, month & year Attributes**

To get day, month and year from a datetime.date value apply the day, month and year attributes.

The output will be day, month and year as integer values.

**Q.14 Get the day, month and year attributes from the date stored in the 'date\_0' variable. Print the attributes and their data-types.**

**The strftime() Function**

To convert a datetime.date value into a string value, use the strftime() function. You can also format the date in different formats using the strftime() function.

Let's format the date stored in the date\_0 variable in the 03-Oct-2004 format. To do this, you will have to pass the following input to the strftime() function.

'%d-%b-%Y'

It is a combination of three format codes. They are:

1. %d format code denotes day of the month as a zero-padded decimal number. E.g., 01, 02, 03 etc.
2. %b format code denotes abbreviated month name of month in the datetime.date value. E.g., Jan Feb, March etc.
3. %Y format code denotes year with century as a decimal number. E.g, 2019, 2020, 2021 etc.

The hyphen - used here just to separate the day, month and year values. You can separate them with different characters.

To get a list of all the strftime() format codes, click on the link provided below and then go to the **strftime() and strptime() Format Codes** section.

<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>

**Q.15**

**Format the date stored in the 'date\_0' variable in the '03-Oct-2004' format. Print newly formatted date and also its data-type.**

**Q.16 Format the date stored in the 'date\_0' variable in the 'October 03, 2004' format. Print newly formatted date and also its data-type.**

**The time() Function**

To get time from a timestamp object, apply the time() function. It returns a datetime.time object.

**Q.17 Get the time from the timestamp stored in the 'timestamp\_0' variable. Store the output in the 'time\_0' variable**.

**The hour, minute & second Attributes**

To get hour, minute and second values from a datetime.time object, apply the hour, minute and second attributes.

The output will be hour, minute and second as integer values.

**Q.18**

**Get the hour, minute and second attributes from the time stored in the 'time\_0' variable. Print the attributes and their data-types.**

#### **Activity 5: Extract Year, Month, Day & Weekday Values**

Let's add four more columns to the DataFrame. The should contain the year, month, day and day-name values for each observation on the air pollutants, temperature, relative humidity and absolute humidity.

For this you can apply the following attributes/functions:

* series\_name.dt.year to get a Pandas series containing the year values as integers.
* series\_name.dt.month to get a Pandas series containing the month values as integers.
* series\_name.dt.day to get a Pandas series containing the day values as integers.
* series\_name.dt.day\_name() to get a Pandas series containing the days of a week, i.e., Monday, Tuesday, Wednesday etc.

**Q. 19 Get the Pandas series containing the year values as integers.**

**Q.20 Get the Pandas series containing the month values as integers.**

**Q.21 Get the Pandas series containing the day values as integers.**

**Q.22 Get the Pandas series containing the days of a week, i.e., Monday, Tuesday, Wednesday etc**.

We can add a column to a DataFrame by following the syntax given below.

**Syntax:** df\_name['column\_name'] = pandas\_series

where df\_name is the Pandas DataFrame in which the pandas\_series to added as a column with the column\_name as the desired name for the column.

**Note:** The indices of the items contained in the pandas\_series must be the same as the indices of the df\_name DataFrame.

**Q.23 Add the 'Year', 'Month', 'Day' and 'Day Name' columns to the DataFrame.**

**Q.24 Display the first five rows of the DataFrame after adding the new columns.**

Let's sort the DataFrame by the DateTime values in the ascending order by using the sort\_values() function. Inside the function, you need to pass the by = 'DateTime' parameter to sort the DataFrame by the DateTime values.

**Note:** By default, the sort\_values function sorts a DataFrame in the ascending order. To sort it in the descending order, pass ascending=False as the second parameter inside the sort\_values function.

**Q.25 Sort the DataFrame by the 'DateTime' values in the ascending order. Also, display the first 10 rows of the DataFrame.**

#### **Activity 6: Replacing Commas**

The values in the CO(GT), C6H6(GT), T, RH and AH columns contain the commas in them. Let's replace the commas with periods (or dots).

**Q.26 Display the first five rows of the DataFrame.**

Create a function which takes a Pandas series containing comma separated decimal values as an input and returns a new Pandas series containing period separated decimal values as an output.

**Q.27 Create a function to replace the commas with periods in a Pandas series.**

Let's apply the comma\_to\_period() function on the 'CO(GT), C6H6(GT), T, RH and AH columns.

**Q.28 Apply the 'comma\_to\_period()' function on the ''CO(GT)', 'C6H6(GT)', 'T', 'RH' and 'AH' columns.**

**Q.29 Display the first five rows of the DataFrame to see the effect of the 'comma\_to\_period()' function.**

Let's also print the information on the df DataFrame. Except for the first column and the last column, all other columns must have the numeric (float or int) data-type values.

**Q.30 Print the first information of the 'df' DataFrame.**

#### **Activity 7: Garbage Value Inspection**

The columns in the df DataFrame also contain -200 value. It is a garbage value or just a random number to represent the further missing (or null) values in the DataFrame. Let's replace it with the most appropriate values for each column.

**Q.31 Get the descriptive statistics for all the numeric data-type columns.**

Except for the Year, Month and Day columns all the other columns contain the -200 as its minimum value. Because of this their mean values are also affected. The median value never gets affected due to some garbage value or very high and very low values. So, we can replace -200 with the median value for each column.

But first, let's find out how many rows contain -200 in each column except for the DateTime, Year, Month and Day columns.

**Q.32 How many rows contain 200 in each column except for the 'DateTime', 'Year', 'Month' and 'Day' columns?**

**Q.33 Find out the percentage of rows containing 200 in each column except for the 'DateTime', 'Year', 'Month' and 'Day' columns.**

Using the same process as above, we calculate the percentage of rows containing -200 in each column except for the DateTime, Year, Month and Day columns.

Now, let's remove all the columns from the df DataFrame containing more than 10% garbage value.

**Q.34 Remove all the columns from the 'df' DataFrame containing more than 10% garbage value.**

**Q.35**

**Again, calculate the percentage of rows containing '200' in each column except for the 'DateTime', 'Year', 'Month' and 'Day' columns.**

Let's replace the -200 value with the median values in all the columns except for the DateTime, Year, Month and Day columns.

Before that we should split the entire DataFrame in two different DataFrames because it contains data for two different years, i.e., 2004 and 2005. Then we should calculate the median values for each column for 2004 and 2005 separately. However, we can also first find out whether the median values are actually different for the two years. If they are not, then we don't need to split the DataFrame into two DataFrames for 2004 and 2005 data points.

**Q.36 Calculate the median values for the columns having indices between 1 and -4 (excluding -4) for the year 2004.**

**Q.37**

**Calculate the median values for the columns having indices between 1 and -4 (excluding -4) for the year 2005.**

As we can see, the median values for 2004 and 2005 are different. Hence, we should split the df DataFrame into two different DataFrames. One for 2004 data points and another for 2005 data points.

#### **Activity 8: Garbage Value Replacement**

Before we separate the DataFrame, let's calculate the count of 2004 and 2005 data points.

**Q.38**

**Count the number of rows containing '2004' and '2005' year values.**

**Q.39 Create a new DataFrame containing records for the year 2004. Also, display the first five rows.**

**Q.40 Calculate the number of rows and columns in the DataFrame for the 2004 year records.**

**Q.41Create a new DataFrame containing records for the year 2005. Also, display the first five rows.**

**Q.42Calculate the number of rows and columns in the DataFrame for the 2004 year records.**

**Q.43Replace the -200 value with the median values for each column having indices between 1 and -4 (excluding -4) for the 2004 year DataFrame.**

**Q.44 Repeat the same exercise for the 2005 year DataFrame.**

**Q.45 Compute the number of rows containing '-200' in each column having indices between 1 and -4 (excluding -4) in the 2004 year DataFrame.**

**Q.46 Again, calculate the percentage of rows containing '-200' in each column except for the 'DateTime', 'Year', 'Month' and 'Day' columns.**

#### **Activity 9: Univariate Grouping**

Before we begin grouping and aggregation, let's first get the list of columns we are left with after applying all the previous operations.

**Q.47 Get the list of columns present in the DataFrame**.

**The groupby() Function**

Now, let's group all the values by months, i.e., club all the values together for January, February and so on...

To group the values of a DataFrame by a particular value, use the groupby() function. Inside the function, you have to pass the column name to by parameter to specify the column about which you want to group the values.

**Syntax:** dataframe.groupby(by='column')

You can additionally set the sort parameter either equal to True or False depending on whether you want to sort the values or not. By default, the groupby() function returns a new Pandas DataFrameGroupBy object containing the grouped sorted values.

**Q.48 Group the records for the 2004 DataFrame together by month.**

The group\_by() function returns a DataFrameGroupBy object with its physical memory location.

How you want to group the values depends on the problem statement at hand. Here, we would like to analyse data month-wise. Hence, we have grouped the DataFrame by the Month column.

**The get\_group() Function**

To get all the occurrences of a value from a group, use the get\_group() function.

**Syntax:** DataFrameGroupBy\_object.get\_group(item)

E.g., if you want to get all the records for the month of March from the group\_2004\_month DataFrameGroupBy object, pass 3 as input to the get\_group() function.

**Q.49 Get all the records for the month of March.**

**Q.50 Get all the records for the month of April.**

**Q.51 Group the records for the 2005 DataFrame together by month.**

#### **Activity 10: Aggregation on Univariate Grouping**

You have already learnt the aggregation functions such as count(), sum(), min(), max(), mean() etc. Let's apply a few of them on the DataFrameGroupBy objects that we have already created.

The pollution is most severe during the winter season in any part of the world. The winter season in Italy begins from the last week of October and lasts till the last week of March. So, let's get the descriptive statistics for the concentrations of the air pollutants, temperature, relative humidity and absolute humidity in March, November and December in 2004 and 2005.

**Q.52 Get the descriptive statistics for March 2004.**

**Q.53 Get the descriptive statistics for March 2005.**

If we keep on aggregating the values month-wise one-by-one, then it will become a cumbersome task. Let's get the descriptive statistics for all the months for both the years 2004 and 2005.

**Q.54 Get the descriptive statistics for all the months for the year 2004.**

**Q.55**

**Get the descriptive statistics for all the months for the year 2005.**

We still have quite large DataFrames containing the descriptive statistics for all the months and all the columns. Let's try to retrieve only the mean, standard deviation and median values for all the months.

#### **Activity 11: The agg() Function**

Instead of applying the aggregation functions on a DataFrameGroupBy object individually, you can apply them together using the agg() function. Inside the agg() function, you have to pass the names of the functions (as string values) as inputs.

**Syntax:** DataFrameGroupBy.agg(func=('func1', 'func2' ... 'funcN'))

where 'func1', 'func2' ... 'funcN' are the names of the aggregation functions to be applied. They are passed as a singular value (through tuple) to the func parameter.

Let's apply the mean(), std() and median() function on the group\_2004\_month DataFrameGroupBy object.

**Q.56 Get mean, standard deviation and median for all the months for the year 2004.**

**Q.57**

**Get mean, standard deviation and median for all the months for the year 2005.**

**Q.58**

**On 'group\_2004\_month', apply the 'mean', 'std' & 'median' functions on the 'T' column and 'max' function on the 'AH' column. Also apply the 'count' function on the 'Month' column.**

**Q.59**

**On the 'group\_2005\_month', apply the 'min' & 'max' functions on the 'AH' and 'RH' columns. Also apply the 'count' function on the 'Month' column.**

#### **Activity 12: Slicing DataFrameGroupBy Object**

We still want to know the mean, standard deviation and median values only for the winter season for 2004 and 2005.

So, we can use the loc[] function to get the rows only for the winter season.

**Q.60 Get mean, standard deviation and median values for the winter season of 2004.**

**Q.61**

**Get the column indices for the 'group\_2004\_month.agg(func=['mean', 'std', 'median'])' DataFrame columns.**

**Q.62**

**Get mean, standard deviation and median values for the winter season of 2004 without the 'Year' & 'Day' columns.**

**Q.63**

**Get mean, standard deviation and median values for the winter season of 2005 without the 'Year' & 'Day' columns.**

**Q.64**

**For winters of 2004, get the quartiles of all the air pollutants, temperature, relative humidity and absolute humidity.**

**Q.65**

**For winters of 2005, get the quartiles of all the air pollutants, temperature, relative humidity and absolute humidity.**

**Q.66 Group the DataFrame for the year 2004 by the 'Month' column without making it a default index column in the aggregated DataFrames.Apply the 'mean' function on the 'DataFrameGroupBy' object.**

#### **Activity 13: Multivariate Grouping & Aggregation**

We can also group multiple columns at once by passing a list of columns to be grouped.

Let's group the DataFrames about the Month & Day Name columns.

**Q.67**

**Group the DataFrame for the year 2004 about the 'Month' & 'Day Name' columns.**

**Q.68**

**Get the count of each day for each month in 2004.**

**Q.69**

**Get all the records for all Mondays in the month of March, 2004.**