

ETL Pipeline Portfolio Project

Overview

You have been contracted to design and implement a data pipeline to retrieve regional air quality data from the [Open Bristol Data Service](#), preprocess and transform the data in a required format and store it in both a PySpark DataFrame and a local PostgreSQL database.

The code should be clean, well written and well documented. Ensure functionality is encapsulated in functions so you can reuse them in other projects. Upload screenshots as proof of completion for all exercises.

Exercise 1

AO2.2

Using Python's request module, retrieve data from the services' api using their GeoJSON link. Consider implementing error handling techniques to mitigate issues with connecting to the API service.

Include screenshots below. What errors might you encounter when connecting to these kinds of services?

For this project as requested I am loading the GeoJason dataset with an Api connection from bristol.gov.uk.

The (**api_url**) link contains the monitor locations for the air quality controls in Bristol.

A key feature for this project is the “Geometry” data, which will map the locations of the monitoring devices across Bristol city.

At the top of this project I am importing all the necessary libraries to allow the extraction and transformation of the dataset:

- **requests** (used for fetching API's resources from the web)
- **json** (a commonly used format to manipulate APIs datasets)
- **pandas** (a powerful data manipulation tool very useful for datasets structures)
- **geopandas** (a pandas extension which is functional to geographic data)

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 1

AO2.2

```

Importing Libraries and Loading API DataFrame

import requests
import json
import pandas as pd
import geopandas as gpd

api_url = pd.DataFrame({
    'API_URL': [
        "https://maps2.bristol.gov.uk/server2/rest/services/ext/air_quality/MapServer/B/query?outFields=*&where=1X3D1&f=geojson"
    ]
})

# Reading the API
for i in api_url['API_URL']:
    response = requests.get(i)
    if response.status_code == 200:
        print(f"Source Data from Bristol.go.uk Website (i).")
        print(response.json())
    else:
        print(f"Failed to fetch from {i}")

[20] ✓ 0.0s Python
Source Data from Bristol.go.uk Website https://maps2.bristol.gov.uk/server2/rest/services/ext/air_quality/MapServer/B/query?outFields=*&where=1X3D1&f=geojson:
{'type': 'FeatureCollection', 'features': [{'type': 'Feature', 'id': 1, 'geometry': {'type': 'Point', 'coordinates': [-2.0277488832862448, 51.48745517999835]}, 'properties': {'OBJECTID': 1, 'siteNo': '1', 'location': 'Withywood School'}}]}

```

With the first *iteration* as(for **i** in **api_url**), I am checking the response status to be equal to **200**. This ensures the status response from the server is **OK**, which means that the api request was successful and the server is providing the data.

Other common *status codes* such as : 404, 500, 403 and 400 would mean:

- URL doesn't exist (404)
- Server Error (500)
- No permissions for access (403)
- something went wrong. (400)

In this case by verifying my server response to be equal to 200 I have effectively and ensured that the dataset is fully accessible and there isn't any issue for accessing it from the web source.

```

if response.status_code == 200:
    print("API load successful. Status code:", response.status_code)
else:
    print("API load failed. Status code:", response.status_code)

[2] ✓ 0.0s Python
API load successful. Status code: 200

```

In the next step I am simply defining my *dataframe_df* and saving a csv file copy to allow the next stages of the exercise.

```

defining my dataframe

dataframe_df = api_url

[15] ✓ 0.0s Python

saving dataframe in my folder to allow transformation

dataframe_df.to_csv("C:/Users/aldom/Documents/Data_Engineering/Final_ETL_Portfolio_Submission/dataframe_df.csv", index=False)

[17] ✓ 0.0s Python

```

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 1

AO2.2

Finally I am reading the file into the system, and, to do this I would need to verify the encoding of the dataset.

For this project I have tried to read the file using some of the most common encodings that support characters and digits. After several attempts and by applying try and error methodology the correct encoding for this database is: **ISO-8859-1**

With this step I am using a try and error technique using the most common encoding types to allow the next stages of the transformation

```
with open("C:/Users/aldom/Documents/Data_Engineering/Final_ETL_Portfolio_Submission/api_data.geojson", encoding="ISO-8859-1") as f:
    content = f.read()
    print(content)
```

```
{
  "type": "FeatureCollection",
  "name": "api_data",
  "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
  "features": [
    { "type": "Feature", "properties": { "OBJECTID": 1, "siteNo": "1", "location": "Withywood School", "SiteID": 1, "Easting": 356434, "Northing": 167823, "Current_": "No", "pollutants": "BTX NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 2, "siteNo": "B1", "location": "Colston Avenue", "SiteID": 2, "Easting": 358628, "Northing": 173011, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 3, "siteNo": "B10", "location": "Blackboy Hills", "SiteID": 3, "Easting": 357448, "Northing": 174658, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 4, "siteNo": "B11", "location": "Three Lamps", "SiteID": 4, "Easting": 359983, "Northing": 171856, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 5, "siteNo": "B12", "location": "Bedminster Parade", "SiteID": 5, "Easting": 358723, "Northing": 171784, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 6, "siteNo": "B13", "location": "Church Road", "SiteID": 6, "Easting": 361261, "Northing": 173413, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 7, "siteNo": "B14", "location": "St. Andrew's Rd", "SiteID": 7, "Easting": 351786, "Northing": 178250, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 8, "siteNo": "B15", "location": "Higham Street", "SiteID": 8, "Easting": 359836, "Northing": 171983, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 9, "siteNo": "B16", "location": "B.R.I.", "SiteID": 9, "Easting": 358729, "Northing": 173499, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 10, "siteNo": "B17", "location": "Bath Road", "SiteID": 10, "Easting": 361217, "Northing": 171429, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 11, "siteNo": "B18", "location": "Whitefriars", "SiteID": 11, "Easting": 358813, "Northing": 173342, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 12, "siteNo": "B19", "location": "Galleries", "SiteID": 12, "Easting": 359142, "Northing": 173211, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 13, "siteNo": "B2", "location": "Ferndown Close", "SiteID": 13, "Easting": 354493, "Northing": 177489, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 14, "siteNo": "B20", "location": "Red Lion Knole", "SiteID": 14, "Easting": 360877, "Northing": 178280, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 15, "siteNo": "B21", "location": "Horsefair", "SiteID": 15, "Easting": 359294, "Northing": 173485, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 16, "siteNo": "B3", "location": "Third Way", "SiteID": 16, "Easting": 352329, "Northing": 178698, "Current_": "Yes", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 17, "siteNo": "B4", "location": "Anglesea Place", "SiteID": 17, "Easting": 357273, "Northing": 174582, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 18, "siteNo": "B5", "location": "Hillcrest", "SiteID": 18, "Easting": 360691, "Northing": 178081, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 19, "siteNo": "B6", "location": "Conham Vale", "SiteID": 19, "Easting": 362921, "Northing": 172122, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    { "type": "Feature", "properties": { "OBJECTID": 20, "siteNo": "B7", "location": "Newfoundland Way", "SiteID": 20, "Easting": 359567, "Northing": 173638, "Current_": "No", "pollutants": "NO2", "InstrumentID": 1 },
    ...
  ]
}
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Exercise 2

AO2.2,
5.1

In anticipation of the loaded data eventually being used by Data Scientists and Analysts, the coordinates of each record need to be normalised to values between 0 and 1, this will mitigate the latitude values taking precedence over the longitude values in potential machine learning applications.

Create a function from first principles that normalises a given value between 0 and 1.

Write at least two assertion tests for this function to ensure it works as intended before you run the full application.

Consider writing a flow chart for the function before you implement it to ensure the logic is sound.

Write a flow chart for the function before you implement it to ensure the logic is sound. Include the flowchart and a screenshot of the function below.

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

In this step I am taking the `dataframe_df` file and extracting X, Y coordinates for the normalization of the data to be (0,1).

By applying Min-Max I have then transformed the values to a range between 0 and 1.

The min-max technique is very useful when working with “geo-spatial” data. This will normalize the coordinates of the dataset. Normalized values will be beneficial for:

- Machine Learning
- Visualization
- Feature Engineering

```
normalizing geometry with 0,1 vaules
```

```
from sklearn.preprocessing import MinMaxScaler

# Extracting the coordinates

dataframe_df['x'] = dataframe_df.geometry.x
dataframe_df['y'] = dataframe_df.geometry.y

# Normalizing

normalized_scale = MinMaxScaler()
dataframe_df[['x_normalized', 'y_normalized']] = normalized_scale.fit_transform(dataframe_df[['x', 'y']])

print(dataframe_df[['geometry', 'x_normalized', 'y_normalized']].head())
```

[27] ✓ 0.0s Python

	geometry	x_normalized	y_normalized
0	POINT (-2.62775 51.40775)	0.450763	0.090933
1	POINT (-2.59681 51.45456)	0.515873	0.293816
2	POINT (-2.61399 51.46921)	0.479713	0.357306
3	POINT (-2.57833 51.44421)	0.554769	0.248973
4	POINT (-2.59529 51.44281)	0.519071	0.242914

The following code verifies that the normalized x and y coordinate values are within the range 0 and 1. With this code I am checking the validity of (*x_normalized* and *y_normalized*) by creating a series of a Boolean expression to check if each element is (True) in the value between 0 and 1.

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 2

AO2.2,
5.1

```

# Check values

check_x = dataframe_df['x_normalized'].between(0, 1).all()
check_y = dataframe_df['y_normalized'].between(0, 1).all()

print(f"x is normalized: {check_x}")
print(f"y is normalized: {check_y}")

```

[31] ✓ 0.0s Python

```

... x is normalized: True
    y is normalized: True

```

Finally I am updating the “*geometry*” column with the **normalized data**. Before loading the data to a spark session I did some preparation (*transformation*), to ensure the relevant columns for the exercise (ID, Location, Location Type, Latitude (Normalised)) will be used with the ***normalized coordinate*** values.

The below code updated the “*geometry*” column in the “*dataframe_df*” by replacing it with the new object points created from the normalized coordinates. With this code I am:

- Creating a new “*point*” geometry by applying the lambda function to each row
- Apply the function to each row (axis=1) ensuring the transformation is applied to all rows

```

Updating the geometry column with the normalized values

from shapely.geometry import Point

# Column update

dataframe_df['geometry'] = dataframe_df.apply(lambda row: Point(row['x_normalized'], row['y_normalized']), axis=1)

print(dataframe_df[['geometry', 'x_normalized', 'y_normalized']].head())

```

[9] ✓ 0.0s Python

```

...
      geometry  x_normalized  y_normalized
0  POINT (0.45076 0.09093)    0.450763    0.090933
1  POINT (0.51587 0.29382)    0.515873    0.293816
2  POINT (0.47971 0.35731)    0.479713    0.357306
3  POINT (0.55477 0.24897)    0.554769    0.248973
4  POINT (0.51907 0.24291)    0.519071    0.242914

```

Lastly I am fitting the data in the Latitude (normalized) column. This ensures that the new ***normalized data*** is stored in a new column called Latitude (normalized).

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 2

AO2.2,
5.1

```
# fit the data in latitude (normalized)

dataframe_df['Latitude (normalized)'] = list(zip(dataframe_df['x_normalized'], dataframe_df['y_normalized']))
print(dataframe_df[['x_normalized', 'y_normalized', 'Latitude (normalized)']].head())
```

[11] ✓ 0.0s Python

	x_normalized	y_normalized	Latitude (normalized)
0	0.450763	0.090933	(0.45076298550013316, 0.090933038267851)
1	0.515873	0.293816	(0.5158727171676434, 0.2938159127984079)
2	0.479713	0.357306	(0.479712873484198, 0.357305528540806)
3	0.554769	0.248973	(0.5547692739533074, 0.2489726016785312)
4	0.519071	0.242914	(0.5190709212978044, 0.2429141946354605)

Below is the flowchart of the *ETL processing* I am applying to this project, from the source (Web) to the final loading destination (SQL database). This flowchart is made to show what steps I am following for the extraction, transformation and the loading of the file into SQL.

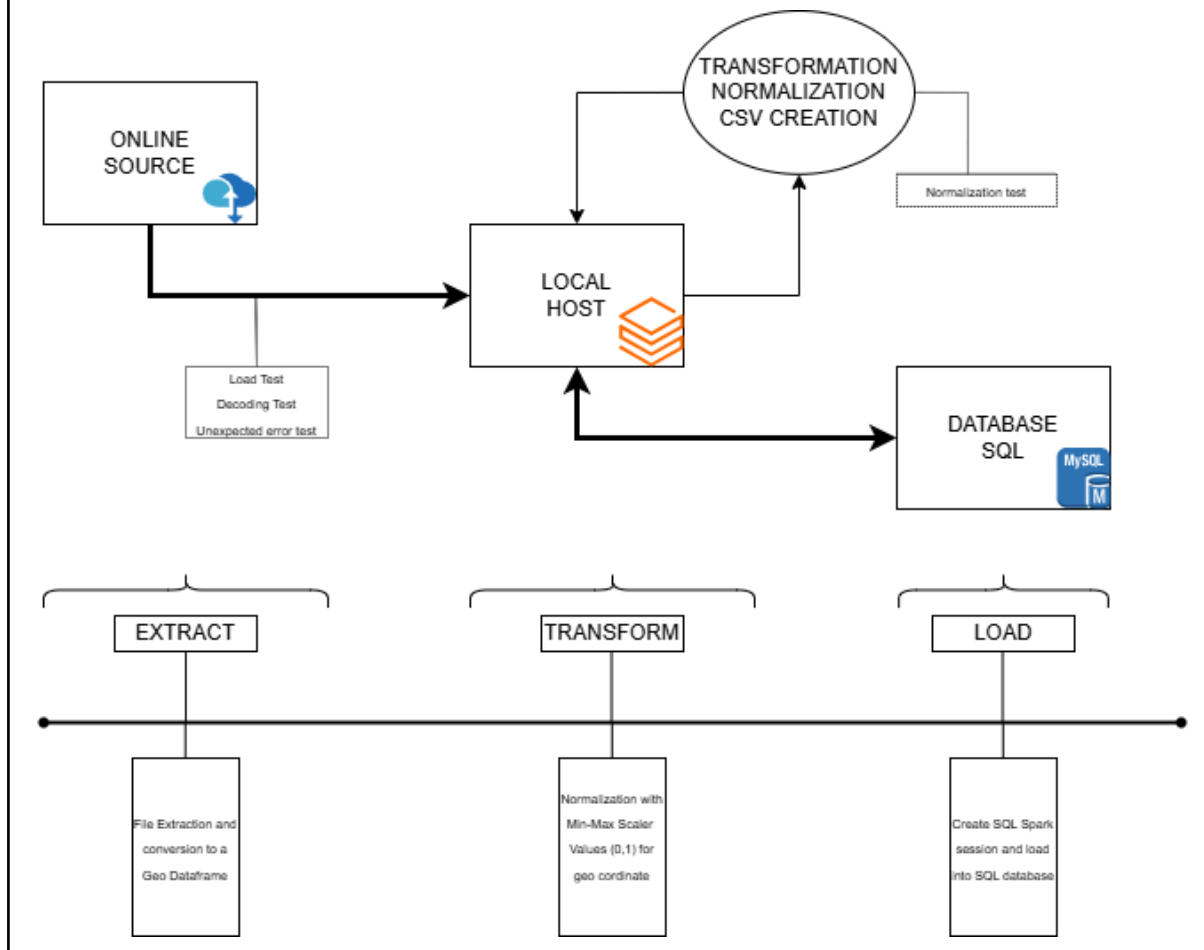


Figure 1: Required fields

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Field	Type
ID	bigint
Location	text
Location Type	text
Latitude (Normalised)	Double precision

Exercise 3

AO2.2,
5.1

Prepare the data for loading by ensuring it conforms to Figure 1.

Consider what data structures you should use in order to load the data into a data frame.

After pairing the new values in the new column on the exercise above, I am then selecting only the relevant columns for this exercise, updating the DataFrame to only include the relevant columns.

```
# select the relevant columns
dataframe_df = dataframe_df[['SiteID', 'location', 'siteNo', 'Latitude (normalized)']]
print(dataframe_df.head())
```

[9] ✓ 0.0s

	SiteID	location	siteNo	\
0	1	Withywood School	1	
1	2	Colston Avenue	B1	
2	3	Blackboy Hill	B10	
3	4	Three Lamps	B11	
4	5	Bedminster Parade	B12	

	Latitude (normalized)
0	(0.45076406043709216, 0.09093774496733431)
1	(0.5158721398968913, 0.29381988672233206)
2	(0.479709367377926, 0.3573084868393437)
3	(0.5547688327294686, 0.248978676129326)
4	(0.519070065069716, 0.24291883545097903)

With the following I am renaming the selected columns following the naming indications of the exercise.

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 3

AO2.2,
5.1

```
# rename the columns as asked in the exercise
dataframe_df.rename(columns={'SiteID': 'ID', 'siteNo': 'Location Type'}, inplace=True)
print(dataframe_df.head())
```

[10] ✓ 0.0s

```
...   ID      location Location Type \
0    1  Withywood School           1
1    2   Colston Avenue           B1
2    3   Blackboy Hill           B10
3    4     Three Lamps           B11
4    5  Bedminster Parade          B12

      Latitude (normalized)
0  (0.45076406043709216, 0.09093774496733431)
1  (0.5158721398968913, 0.29381988672233206)
2  (0.479709367377926, 0.3573084868393437)
3  (0.5547688327294686, 0.248978676129326)
4  (0.519070065069716, 0.24291883545097903)
```

```
print(dataframe_df.dtypes)
```

[11] ✓ 0.0s

```
...   ID      int64
location      object
Location Type  object
Latitude (normalized)  object
dtype: object
```

Now the Final Data is ready for importing into a Spark session.

```
final_data = dataframe_df.copy()
print(final_data.head())
```

[12] ✓ 0.0s

```
...   ID      location Location Type \
0    1  Withywood School           1
1    2   Colston Avenue           B1
2    3   Blackboy Hill           B10
3    4     Three Lamps           B11
4    5  Bedminster Parade          B12

      Latitude (normalized)
0  (0.45076406043709216, 0.09093774496733431)
1  (0.5158721398968913, 0.29381988672233206)
2  (0.479709367377926, 0.3573084868393437)
3  (0.5547688327294686, 0.248978676129326)
4  (0.519070065069716, 0.24291883545097903)
```

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 4

AO2.2

To ensure your transformation has worked, and that you have a local backup of your transformed data, load the data into a csv file in your project folder.

We suggest using the csv module.

Before importing this to a spark session, I am furthermore saving this final dataset (*final_data*), locally in my folder as a CSV format.

```
#save the final data to a CSV file
final_data.to_csv("C:\\Users\\alodom\\Documents\\Data_Engineering\\Submissions\\final_data.csv", index=False)
```

[13] ✓ 0.0s

Exercise 5

AO2.2

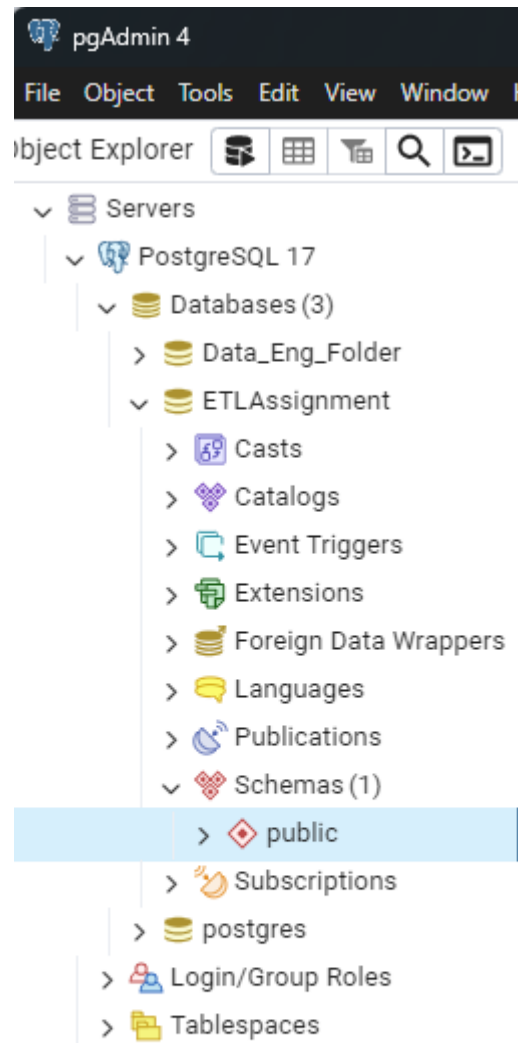
Create a Postgres server and use PgAdmin to set up a database called ETLAssignment.

In the picture below I have created a database called ETLAssignment.

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 5

AO2.2



Exercise 6

AO2.2

Import SparkSession and use the builder object to create a new spark session. Load your transformed data into a dataframe, and write it to your new database in a new table called AirQualityBristol.

When you write the data, ensure you use the overwrite mode parameter to prevent any

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 6

AO2.2

write issues.

Here I am Initiating the Spark environment and importing the session function and all other functions needed for the exercise.

Below I am creating the Spark Session called “AirQualityBristol”.

```
[14] import os
import sys

os.environ["JAVA_HOME"] = "JDK 8"
os.environ["PYSPARK_PYTHON"] = sys.executable
os.environ["PYSPARK_DRIVER_PYTHON"] = sys.executable
✓ 0.0s

[15] from pyspark.sql import SparkSession
from pyspark.sql.functions import *
✓ 0.2s

# Create a Spark session
Spark_Session = SparkSession.builder \
    .appName("AirQualityBristol") \
    .getOrCreate()

print(Spark_Session)
[16] ✓ 21.8s
... <pyspark.sql.session.SparkSession object at 0x00000216FDF78090>
```

Here I am loading the file (**final_data**) into the previously created Spark Session.

```
# Load the final_data.csv into a Spark DataFrame
Air_Quality_Bristol_df = Spark_Session.read.csv("C:\\Users\\alldom\\Documents\\Data_Engineering\\Submissions\\final_data.csv", header=True, inferSchema=True)

Air_Quality_Bristol_df.printSchema()
Air_Quality_Bristol_df.show(5)
[17] ✓ 11.0s

... root
 |-- ID: integer (nullable = true)
 |-- location: string (nullable = true)
 |-- Location Type: string (nullable = true)
 |-- Latitude (normalized): string (nullable = true)

+---+-----+-----+-----+
| ID|      location|Location Type|Latitude (normalized)|
+---+-----+-----+-----+
| 1| Withwood School|      1| (0.45076406043709...|
| 2| Colston Avenue|     B1| (0.51587213980689...|
| 3| Blackboy Hill|    B10| (0.47970936737792...|
| 4| Three Lamps|    B11| (0.55476883272946...|
| 5| Bedminster Parade|   B12| (0.51907006506971...|
+---+-----+-----+-----+
only showing top 5 rows
```

In the next step I am creating a new Sql_Spark session and loading the dataset in SQL.

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

Exercise 6

AO2.2

```
# Load the CSV file into a DataFrame in PostgreSQL

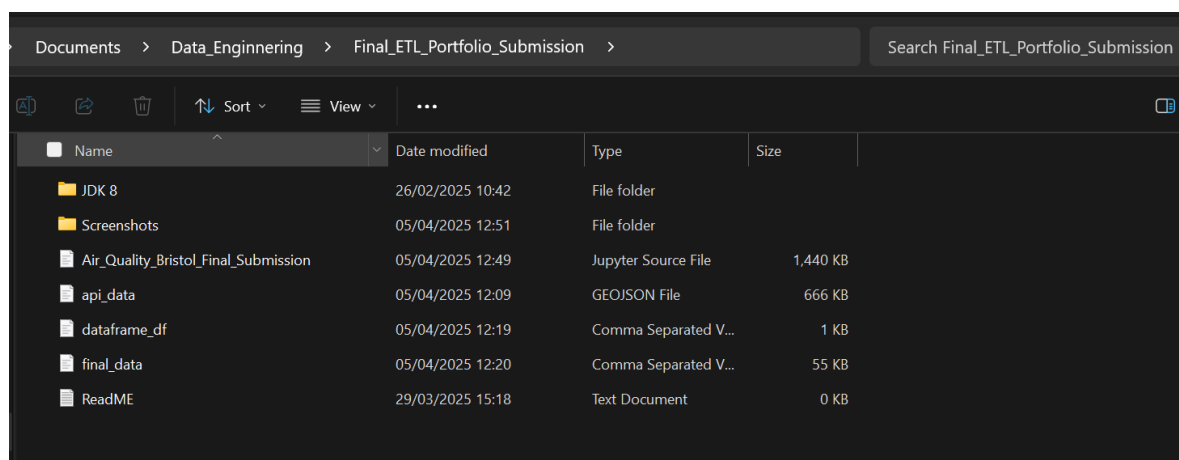
pg_url = "jdbc:postgresql://localhost:5432/ETLAssignment"
my_account = {
    "user": "postgres",
    "password": "This is my password",
    "driver": "org.postgresql.Driver"
}

# Write the transformed data to the database

Air_Quality_Bristol_df.write.jdbc(
    url=pg_url, \
    table="AirQualityBristol",
    mode="overwrite",
    properties=my_account
)
```

Exercise 7

In your project, create a README file that contains the following statement:
Contains public sector information licensed under the Open Government Licence v3.0.



Name	Date modified	Type	Size
JDK 8	26/02/2025 10:42	File folder	
Screenshots	05/04/2025 12:51	File folder	
Air_Quality_Bristol_Final_Submission	05/04/2025 12:49	Jupyter Source File	1,440 KB
api_data	05/04/2025 12:09	JSON File	666 KB
dataframe_df	05/04/2025 12:19	Comma Separated V...	1 KB
final_data	05/04/2025 12:20	Comma Separated V...	55 KB
README	29/03/2025 15:18	Text Document	0 KB

Exercise 8

Zip your project folder and upload it alongside this document.

The folder must be named in the following format: CRMID-Assignment

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY

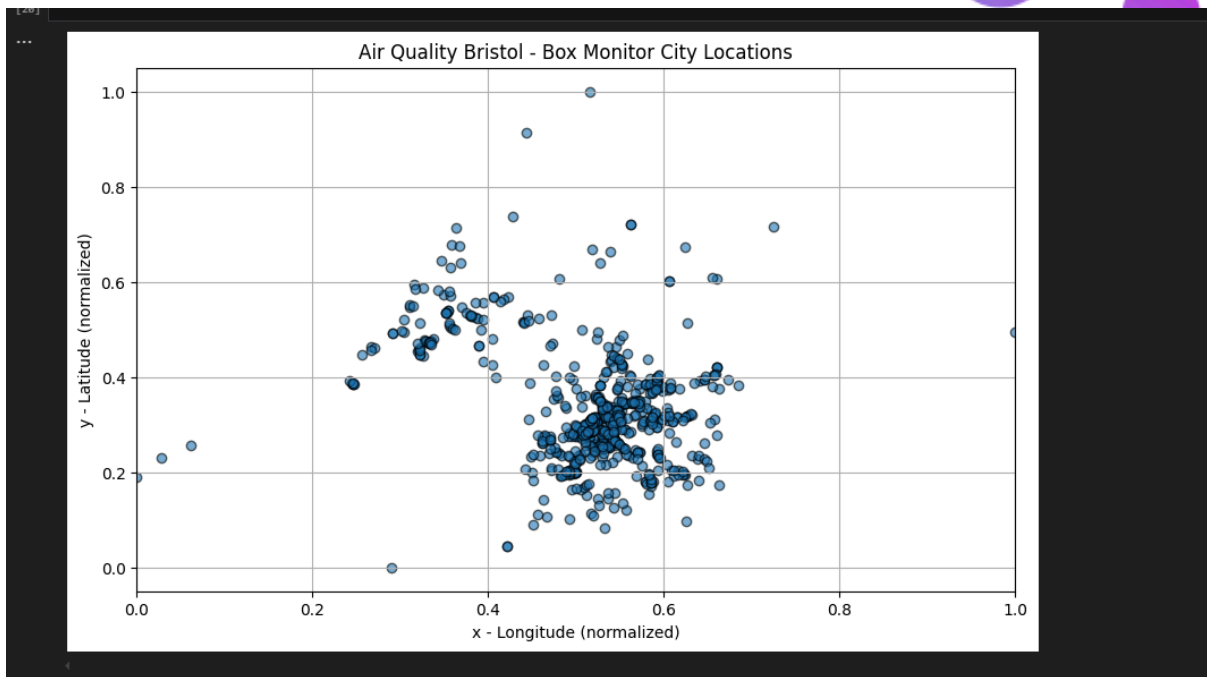
Exercise 8

✓ CRM0005331 - Assignment 02/04/2025 20:01 Compressed (zipped)... 101,469 KB

Extension activities.

1. Ensure your functions are well encapsulated, try to apply your ETL code to other open data sources on the service.
2. Use a visualisation library to explore the data
3. Load your transformed data from your database into PowerBI.

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY



Counting Total number of locations

```
total_locations = Air_Quality_Bristol_df.select("location").distinct().count()
print(f"The Total number of locations is: {total_locations}")
print("This is the number of Box Monitor Around Bristol City")
```

[24] ✓ 4.1s

Python

```
... The Total number of locations is: 674
This is the number of Box Monitor Around Bristol City
```

Ref:	TEM-XXXX	Doc:	Template Title	Rev:	X.X
Author:	First Name Surname	Class:	Public / Restricted / Confidential	Date:	DD MM YYYY