```
pip install beautifulsoup4 #installare se necessario, utilizza la libreria nella cella successiva
```

```
    Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (4.11.2)
    Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4) (2.5)
```

il codice serve a estrarre e stampare i nomi dei file (collegamenti) presenti nella pagina web che iniziano con il prefisso specificato ('arpa'). Questa operazione serve ad estrarre informazioni specifiche da una pagina web, come i nomi dei file disponibili per il download.

```
import requests
from bs4 import BeautifulSoup

def get_names_with_prefix(soup, prefix):
    # Trova tutti i link nella pagina
    links = soup.find_all('a')

    # Inizializza una lista per memorizzare i nomi dei file
    file_names = []

    # Estrai e aggiungi i nomi dei file che iniziano con il prefisso alla lista
    for link in links:
        href = link.get('href')
        if href and href.startswith(prefix):
            file_names.append(href)

    # Restituisci la lista dei nomi dei file
    return file_names

url = 'http://www.pa.icar.cnr.it/storniolo/files/master/opendata'

# Effettua una richiesta GET per ottenere il contenuto HTML della pagina
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Chiama la funzione passando il prefisso desiderato
desired_prefix = 'arpa'
result_file_names = get_names_with_prefix(soup, desired_prefix)

# Stampa la lista dei nomi dei file
for file_name in result_file_names:
  print(file_name)
```

INSTALLAZIONE DELLA MACCHINA VIRTUALE, PREPARAZIONE AMBIENTE GOOGLE DRIVE, SPARK

per l'esecuzione di PySpark in Google Colab

```
#Inserire nella propria piattaforma ATLAS per MongoDB gli IP pubblici di Google Colab
'''
34.4.5.0/10
34.64.0.0/10
34.128.0.0/10
35.184.0.0/13
35.192.0.0/12
35.208.0.0/12
35.224.0.0/12
35.240.0.0/13
104.196.0.0/14
'''
#Visualizzazione indirizzo IP assegnato
!curl ipecho.net/plain
```

```
    34.106.152.3
```

```
'''
Il codice verifica se è in esecuzione in Google Colab (IN_COLAB).
Se è in esecuzione in Google Colab, monta Google Drive per accedere ai file presenti su Google Drive.

Configura un ambiente per l'esecuzione di PySpark, scaricando e configurando Apache Spark,
gestendo la connessione a Google Drive in Google Colab e installando le dipendenze necessarie
come findspark e pymongo
'''



import os
import sys

#Disable Warnings
import warnings
warnings.simplefilter(action='ignore', category=Warning)

IN_COLAB = "google.colab" in sys.modules

if IN_COLAB:
    from google.colab import drive

    drive.mount("/content/drive", force_remount=True)

    # Install JDK
    !apt-get install openjdk-19-jdk-headless -qq > /dev/null
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-19-openjdk-amd64"
    !echo; java --version; echo

    # download spark if not present on drive
    # and untar spark from drive
    ! GDRIVE_MASTER_DIRECTORY="/content/drive/MyDrive/Master/" \
    && SPARK_DOWNLOAD_URL="https://archive.apache.org/dist/spark/spark-3.4.1/spark-3.4.1-bin-hadoop3.tgz" \
    && SPARK_VM_SYMLINK="/content/spark" \
    && SPARK_TGZ_LOCAL_FILENAME=$(basename "${SPARK_DOWNLOAD_URL}") \
    && if [ ! -d ${SPARK_VM_SYMLINK} ]; then echo "${SPARK_VM_SYMLINK} link not exist!" \
    && cd ${GDRIVE_MASTER_DIRECTORY} && echo "change directory to ${GDRIVE_MASTER_DIRECTORY}" \
```

```
    && if [ ! -f "${SPARK_TGZ_LOCAL_FILENAME}" ]; then echo "${SPARK_TGZ_LOCAL_FILENAME} not exist!" \
    && curl -O ${SPARK_DOWNLOAD_URL} > /dev/null && echo "Save ${SPARK_TGZ_LOCAL_FILENAME}" \
    ; fi \
    && cd "/content" && echo "change directory to /content" \
    && tar -xf "${GDRIVE_MASTER_DIRECTORY}${SPARK_TGZ_LOCAL_FILENAME}" > /dev/null && echo "extract ${SPARK_TGZ_LOCAL_FI
    && ln -s "/content/$(basename -s '.tgz' ${SPARK_TGZ_LOCAL_FILENAME})" ${SPARK_VM_SYMLINK} > /dev/null && echo "creat
    ; fi

    os.environ["SPARK_HOME"] = "/content/spark"
else:
    print("Local Python execution environment!")
#-----------------------------------------------------------------------------

#Install findspark using pip to make pyspark importable as regular library
!pip -q install findspark
import findspark
findspark.init()

import pyspark
from pyspark.sql import SparkSession

# Install pymongo
!pip -q install pymongo
from pymongo import MongoClient
```

```
Mounted at /content/drive

openjdk 19.0.2 2023-01-17
OpenJDK Runtime Environment (build 19.0.2+7-Ubuntu-0ubuntu322.04)
OpenJDK 64-Bit Server VM (build 19.0.2+7-Ubuntu-0ubuntu322.04, mixed mode, sharing)
```

Configurazione di una connessione tra PySpark e MongoDB Atlas

Stabilisce una connessione tra PySpark e MongoDB Atlas, configurando le informazioni di connessione e testando la connessione attraverso una query di ping

```
mongoURL = "mongodb+srv://root:root@cluster0.k4vmon8.mongodb.net/"
mongoParam = "?authSource=admin&replicaSet=atlas-wjju6m-shard-0&readPreference=primary"
mongoDB = "hadoop"

URI = mongoURL + mongoDB + mongoParam

spark = SparkSession.builder \
    .master("local") \
    .appName("Spark_MongoDB") \
    .config("spark.mongodb.input.uri", URI) \
    .config("spark.mongodb.output.uri", URI) \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:3.0.2") \
    .enableHiveSupport() \
    .getOrCreate()

sc = spark.sparkContext
sql = spark.sql

spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
spark.conf.set("spark.sql.execution.arrow.pyspark.fallback.enabled", "true")
spark.conf.set("spark.sql.execution.arrow.pyspark.selfDestruct.enabled","true")

print("SPARK version: "+spark.version)
!find /root/ -type f -name "mongo-spark-*"

# Test di connessione
try:
    # Query di ping per confermare la connessione
    ping_result = spark.read.format("mongo").option("uri", URI).option("collection", "test").load()
    print("Connessione a MongoDB Atlas riuscita!")
except Exception as e:
    print(f"Errore di connessione a MongoDB Atlas: {e}")
```

```
SPARK version: 3.4.1
/root/.ivy2/cache/org.mongodb.spark/mongo-spark-connector_2.12/jars/mongo-spark-connector_2.12-3.0.2.jar
Connessione a MongoDB Atlas riuscita!
```

Definizione della funzione load_dati

Questa funzione scarica un file da una specifica URL, lo salva e lo scompatta se è un file zip, legge i dati da un file CSV risultante, e infine salva i dati in una collezione di MongoDB. I file temporanei creati durante il processo vengono rimossi alla fine.

```
def load_data(url, collection, mode):
    print(url)
    file_name = url.split('/')[-1]
    r = requests.get(url, allow_redirects=True)
    open(file_name, "wb").write(r.content)
    print(file_name)
    os.system("unzip " + file_name)
    input_file = file_name.split('.')[0] + '.csv'
    print(input_file)
    #Create DataFrame
    print('Reading from < ' + input_file)
    data = spark.read.csv(input_file, sep=";", header=True, inferSchema=True)
    print('Saving to > ' + collection, '\t-\tMode: ' + mode)
    #Save dataframe to MongoDB (!)
    data.write.format("com.mongodb.spark.sql.DefaultSource").option("collection", collection).mode(mode).save()
    #Remove temporary files
    os.system("rm -f " + file_name)
    os.system("rm -f " + input_file)
```

Creazione collezioni in MongoDB

Si procede con la creazione di due collezioni in MongoDB: una denominata 'stazioni' e l'altra dedicata agli 'inquinanti'. Tale operazione verrà eseguita utilizzando i file dell'anagrafica come fonte di dati primaria

```
'''
La funzione generate_final_urls crea una lista di URL completi combinando una porzione di URL di base
con i nomi dei file forniti, solo per i file che contengono una determinata parola chiave.
La lista risultante viene poi restituita.
'''

def generate_final_urls(file_names, base_url, keyword):
    final_urls = []

    for file_name in file_names:
        if keyword in file_name:
            final_url = base_url + file_name
            final_urls.append(final_url)

    return final_urls


# Utilizzo per l'anafrafica delle stazioni
base_url_portion = 'http://www.pa.icar.cnr.it/storniolo/files/master/opendata/'
keyword_to_include = 'stazioni'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

# Stampa dei risultati
print(result_final_urls)
```

Caricamento dei dati dell'anagrafica stazioni nella collezione MongoDB 'stazioni'

```
for url in result_final_urls:
    try:
```

```
        load_data(url, "stazioni", "overwrite")
    except requests.exceptions.ConnectTimeout as e:
        print(f"Timeout during the request to {url}. Please check your internet connection and try again.")


'''
legge i dati dalla collezione MongoDB "stazioni" utilizzando PySpark,
rimuove il campo "_id", stampa lo schema
e mostra i primi 60 record ordinati per "stazione_id".
'''

df_stazioni = spark.read.format("mongo").option("collection","stazioni").load().drop('_id')
df_stazioni.printSchema()
df_stazioni.sort(df_stazioni.stazione_id).show(60,truncate=False)
```

```
|14        |38.174341        |15.546512        |ME - Villa Dante                    |IT1913 |
|17        |37.065105        |14.261254        |Gela - Tribunale                    |IT1914 |
|23        |38.16029         |15.231209        |Barcellona Pozzo di Gotto           |IT1914 |
|44        |37.096276        |15.131752        |Solarino                            |IT1914 |
|45        |37.306413        |13.589904        |AG - Centro                         |IT1915 |
|48        |35.502802        |12.597921        |Lampedusa                           |IT1915 |
|49        |37.489848        |14.05111         |Caltanissetta                       |IT1915 |
|52        |37.924522        |14.662009        |Cesarò Port. Femmina Morta Calacuderi|IT1915 |
|53        |37.882476        |12.71802         |Salemi diga Rubino                  |IT1915 |
|102       |0.0              |0.0              |Gela Pontile                        |IT1914 |
|110       |37.221026        |15.169058        |Augusta Contrada Marcellino         |IT1914 |
|112       |0.0              |0.0              |Augusta Villa Augusta               |IT1914 |
|1908101   |38.012365        |12.546894        |Trapani                             |IT1915 |
```

```
|1908367    |38.198256          |15.552955          |ME - Boccetta                       |IT1913 |
|1908369    |38.190608          |15.249112          |Milazzo - Termica                   |IT1914 |
|1908371    |38.231469          |15.247567          |a2a - Milazzo                       |IT1914 |
|1908487    |37.29924           |13.551601          |AG - Monserrato                     |IT1915 |
|1908497    |37.291894          |13.532537          |Porto Empedocle                     |IT1914 |
|1908498    |37.307047          |13.593963          |AG - ASP                            |IT1915 |
|1908501    |37.062217          |14.284218          |Gela - Enimed                       |IT1914 |
|1908512    |37.145943          |14.395552          |Niscemi                             |IT1914 |
|1908513    |37.070349          |14.253618          |Gela - Via Venezia                  |IT1914 |
|1908517    |37.05961           |14.26508           |Parcheggio AGIP                     |IT1914 |
|1908519    |37.055867          |14.297144          |Gela - Ex Autoparco                 |IT1914 |
|1908520    |37.022486          |14.344965          |Gela - Biviere                      |IT1914 |
|1908521    |37.075693          |14.223844          |Gela - Capo Soprano                 |IT1914 |
|1908601    |37.564125          |14.281463          |Enna                                |IT1915 |
|1908701    |37.515714          |15.010788          |Misterbianco                        |IT1912 |
|1908702    |37.529257          |15.081122          |CT - Parco Gioieni                  |IT1912 |
|1908712    |37.511594          |15.075233          |CT - Ospedale Garibaldi             |IT1912 |
|1908795    |37.515808          |15.097211          |CT - Viale Vittorio Veneto          |IT1912 |
|1908801    |36.917119          |14.734022          |RG - Campo Atletica                 |IT1914 |
|1908802    |36.926331          |14.714509          |RG - Villa Archiemede               |IT1914 |
|1908805    |36.729474          |14.838651          |Pozzallo                            |IT1914 |
|1908901    |37.2184            |15.2205            |Augusta                             |IT1914 |
|1908902    |37.093973          |15.208712          |SR - Belvedere                      |IT1914 |
|1908910    |37.098447          |15.262506          |SR - Via Gela                       |IT1914 |
|1908962    |37.182374          |15.128831          |Melilli                             |IT1914 |
|1908963    |37.156119          |15.190867          |Priolo                              |IT1914 |
|1908964    |37.085751          |15.268014          |SR - ASP Pizzuta                    |IT1914 |
|1908965    |37.067768          |15.285331          |SR - Pantheon                       |IT1914 |
|1908966    |37.091295          |15.285297          |SR - Verga                          |IT1914 |
|1908967    |37.075831          |15.278581          |SR - Teracati                       |IT1914 |
|1908971    |37.1946            |15.1829            |Augusta Megara                      |IT1914 |
+-----------+-------------------+-------------------+------------------------------------+-------+
```

Caricamento dei dati dell'anagrafica inquinanti nella collezione MongoDB 'inquinanti'

```
keyword_to_include = 'inquinanti'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

for url in result_final_urls:
    try:
        load_data(url, "inquinanti", "overwrite")
    except requests.exceptions.ConnectTimeout as e:
        print(f"Timeout during the request to {url}. Please check your internet connection and try again.")
```

```
http://www.pa.icar.cnr.it/storniolo/files/master/opendata/arpa-qualita-aria-anagrafica-inquinanti_csv.zip
arpa-qualita-aria-anagrafica-inquinanti_csv.zip
arpa-qualita-aria-anagrafica-inquinanti_csv.csv
Reading from < arpa-qualita-aria-anagrafica-inquinanti_csv.csv
Saving to > inquinanti  -       Mode: overwrite
```

```
df_inquinanti = spark.read.format("mongo").option("collection","inquinanti").load().drop('_id')
df_inquinanti.printSchema()
df_inquinanti.show(truncate=False)
```

```
root
 |-- condizioneStandardTemperatura_descrizione: string (nullable = true)
 |-- condizioneStandardTemperatura_unitaMisura: string (nullable = true)
 |-- condizioneStandardTemperatura_valore: integer (nullable = true)
 |-- inquinante_descrizione: string (nullable = true)
 |-- inquinante_id: integer (nullable = true)
 |-- inquinante_simbolo: string (nullable = true)
 |-- tipoMisura: string (nullable = true)
 |-- unitaMisura_descrizione: string (nullable = true)
 |-- unitaMisura_id: string (nullable = true)
 |-- unitaMisura_simbolo: string (nullable = true)

+-----------------------------------------------------------+-----------------------------------------+----------
|condizioneStandardTemperatura_descrizione                  |condizioneStandardTemperatura_unitaMisura|condizione
+-----------------------------------------------------------+-----------------------------------------+----------
|standardizzazione del volume di aria alla temperatura di 293 K|K                                        |293
|standardizzazione del volume di aria alla temperatura di 293 K|K                                        |293
|standardizzazione del volume di aria alla temperatura di 293 K|K                                        |293
```

```
|standardizzazione del volume di aria alla temperatura di 293 K|K                                         |293
|standardizzazione del volume di aria alla temperatura di 293 K|K                                         |293
|standardizzazione del volume di aria alla temperatura di 293 K|K                                         |293
|standardizzazione del volume di aria alla temperatura di 293 K|K                                         |293
|standardizzazione del volume di aria alla temperatura di 293 K|K                                         |293
|null                                                          |null                                      |null
|null                                                          |null                                      |null
|standardizzazione del volume di aria alla temperatura di 293 K|K                                         |293
+--------------------------------------------------------------+------------------------------------------+----------
```

1. Caricamento dei dati della qualità dell'aria del 2018 e del 2019

2. Nelle collezioni MongoDB 'aria-2018' e 'aria-2019'

```
def process_urls_and_load_data(urls, collection, mode):
    for url in urls:
        try:
            load_data(url, collection, mode)
        except requests.exceptions.ConnectTimeout as e:
            print(f"Timeout during the request to {url}. Please check your internet connection and try again.")


keyword_to_include = '2018'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

process_urls_and_load_data(result_final_urls, "aria_2018", "append")
```

```
keyword_to_include = '2019-6001'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

process_urls_and_load_data(result_final_urls, "aria_2019", "append")
```

```
    http://www.pa.icar.cnr.it/storniolo/files/master/opendata/arpa-qualita-aria-2019-6001_csv.zip
    arpa-qualita-aria-2019-6001_csv.zip
    arpa-qualita-aria-2019-6001_csv.csv
    Reading from < arpa-qualita-aria-2019-6001_csv.csv
    Saving to > aria_2019   -       Mode: append
```

```
keyword_to_include = '2019-5'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

process_urls_and_load_data(result_final_urls, "aria_2019", "append")
```

```
keyword_to_include = '2020-5'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

process_urls_and_load_data(result_final_urls, "aria_2020", "append")
```

```
keyword_to_include = '2020-6001'

# Chiamata alla funzione con la lista di nomi di file, la porzione di URL e la stringa desiderata
result_final_urls = generate_final_urls(result_file_names, base_url_portion, keyword_to_include)

process_urls_and_load_data(result_final_urls, "aria_2020", "append")
```

Carichiamo i dati sulla qualità dell'aria del 2018

```
df_aria = spark.read.format("mongo").option("collection","aria_2018").load().drop('_id')
df_aria.printSchema()
df_aria.show(50)
```

```
root
 |-- inquinante_id: integer (nullable = true)
 |-- misura_anno: integer (nullable = true)
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)
 |-- periodo_media: string (nullable = true)
 |-- stazione_id: integer (nullable = true)

+-------------+-----------+-------------------+-------------+-------------+-----------+
|inquinante_id|misura_anno|     misura_dataora|misura_valore|periodo_media|stazione_id|
+-------------+-----------+-------------------+-------------+-------------+-----------+
|            1|       2018|2018-01-01 00:00:00|          1.7|            h|    1908366|
|            1|       2018|2018-01-01 01:00:00|          1.3|            h|    1908366|
|            1|       2018|2018-01-01 02:00:00|          0.9|            h|    1908366|
|            1|       2018|2018-01-01 03:00:00|          0.7|            h|    1908366|
|            1|       2018|2018-01-01 04:00:00|          0.7|            h|    1908366|
|            1|       2018|2018-01-01 05:00:00|          0.8|            h|    1908366|
|            1|       2018|2018-01-01 06:00:00|          0.8|            h|    1908366|
|            1|       2018|2018-01-01 07:00:00|          0.7|            h|    1908366|
|            1|       2018|2018-01-01 08:00:00|          0.6|            h|    1908366|
|            1|       2018|2018-01-01 09:00:00|          0.9|            h|    1908366|
|            1|       2018|2018-01-01 10:00:00|          1.3|            h|    1908366|
|            1|       2018|2018-01-01 11:00:00|          2.9|            h|    1908366|
|            1|       2018|2018-01-01 12:00:00|          9.7|            h|    1908366|
|            1|       2018|2018-01-01 13:00:00|          1.6|            h|    1908366|
|            1|       2018|2018-01-01 14:00:00|          0.9|            h|    1908366|
|            1|       2018|2018-01-01 15:00:00|          0.8|            h|    1908366|
|            1|       2018|2018-01-01 16:00:00|          0.8|            h|    1908366|
|            1|       2018|2018-01-01 17:00:00|          0.6|            h|    1908366|
|            1|       2018|2018-01-01 18:00:00|          0.6|            h|    1908366|
|            1|       2018|2018-01-01 19:00:00|          0.6|            h|    1908366|
|            1|       2018|2018-01-01 20:00:00|          0.5|            h|    1908366|
```

```
|             1|       2018|2018-01-01 21:00:00|          0.6|             h|    1908366|
|             1|       2018|2018-01-01 22:00:00|          0.7|             h|    1908366|
|             1|       2018|2018-01-01 23:00:00|          0.5|             h|    1908366|
|             1|       2018|2018-01-02 00:00:00|          0.6|             h|    1908366|
|             1|       2018|2018-01-02 01:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 02:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 03:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 04:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 05:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 06:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 07:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 08:00:00|          1.6|             h|    1908366|
|             1|       2018|2018-01-02 09:00:00|          0.5|             h|    1908366|
|             1|       2018|2018-01-02 10:00:00|          0.9|             h|    1908366|
|             1|       2018|2018-01-02 11:00:00|          0.2|             h|    1908366|
|             1|       2018|2018-01-02 12:00:00|          2.7|             h|    1908366|
|             1|       2018|2018-01-02 13:00:00|          3.2|             h|    1908366|
|             1|       2018|2018-01-02 14:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 15:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 16:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 17:00:00|          0.0|             h|    1908366|
|             1|       2018|2018-01-02 18:00:00|          0.1|             h|    1908366|
|             1|       2018|2018-01-02 19:00:00|          0.6|             h|    1908366|
|             1|       2018|2018-01-02 20:00:00|          0.6|             h|    1908366|
|             1|       2018|2018-01-02 21:00:00|          3.2|             h|    1908366|
|             1|       2018|2018-01-02 22:00:00|          7.3|             h|    1908366|
```

3. Selezione delle stazioni meteo comprese nel quadrante scelto

Le stazione del secondo quadrante ricadono nelle soglie seguenti di latitudine e longitudine:

latitudine > 37.3 longitudine > 14

La funzione filter_dataframe_by_coordinates filtra un DataFrame Spark delle stazioni di rilevamento in base alle coordinate specificate (lat_threshold e lon_threshold). Vengono mantenute solo le stazioni con una latitudine superiore a 37.30 e una longitudine superiore a 14.0.

Viene restituito un nuovo DataFrame Spark contenente solo le righe che soddisfano i criteri del filtro applicato. Infine, viene stampato il DataFrame risultante e il numero di stazioni nel secondo quadrante.

```
from pyspark.sql.functions import col

def filter_dataframe_by_coordinates(df, lat_threshold, lon_threshold):
    """
    Filtra il DataFrame in base alle coordinate specificate.

    Parameters:
    - df: DataFrame Spark
    - lat_threshold: soglia per la latitudine
    - lon_threshold: soglia per la longitudine

    Returns:
    - DataFrame Spark filtrato
    """
    filtered_df = df.filter((col("stazione_latitudine") > lat_threshold) & (col("stazione_longitudine") > lon_thresho
    return filtered_df

# Chiamata alla funzione per ottenere un DataFrame filtrato
df_quadrante = filter_dataframe_by_coordinates(df_stazioni, lat_threshold=37.30, lon_threshold=14.0)

# Visualizza il risultato
df_quadrante.show()
print(f'Le stazioni del secondo quadrante sono:{df_quadrante.count()}')
```

```
+-----------+------------------+-------------------+--------------------+-------+
|stazione_id|stazione_latitudine|stazione_longitudine|       stazione_nome|zone_id|
+-----------+------------------+-------------------+--------------------+-------+
|         14|         38.174341|          15.546512|    ME - Villa Dante| IT1913|
|    1908367|         38.198256|          15.552935|       ME - Boccetta| IT1913|
|    1908313|         38.160495|          15.275031|Santa Lucia del Mela| IT1914|
|    1908366|         38.205556|          15.310064|       Pace del Mela| IT1914|
|         23|          38.16029|          15.231209|Barcellona Pozzo ...| IT1914|
|    1908369|         38.190608|          15.249112|   Milazzo - Termica| IT1914|
|         49|         37.489848|           14.05111|       Caltanissetta| IT1915|
```

```
|    1908712|          37.511594|           15.075233|CT - Ospedale Gar...| IT1912|
|    1908795|          37.515808|           15.097211|CT - Viale Vittor...| IT1912|
|    1908702|          37.529257|           15.081122|  CT - Parco Gioieni| IT1912|
|         11|          37.578669|           15.101062|San Giovanni La P...| IT1912|
|    1908701|          37.515714|           15.010788|        Misterbianco| IT1912|
|    1908601|          37.564125|           14.281463|               Enna| IT1915|
|         52|          37.924522|           14.662009|Cesarò Port. Femm...| IT1915|
|    1908371|          38.231469|           15.247567|       a2a - Milazzo| IT1914|
|    1908310|          38.183005|           15.301496| a2a - Pace del mela| IT1914|
|    1908312|          38.174624|           15.271443|a2a - S.Filippo d...| IT1914|
+-----------+-------------------+--------------------+--------------------+-------+

Le stazioni del secondo quadrante sono:17
```

```
#Estrazione della lista degli id delle stazioni del secondo quadrante

lista_stazioni_id = df_quadrante.select("stazione_id").rdd.flatMap(lambda x: x).collect()

print(lista_stazioni_id)
```

```
[14, 1908367, 1908313, 1908366, 23, 1908369, 49, 1908712, 1908795, 1908702, 11, 1908701, 1908601, 52, 1908371, 1908
```

3.1 Selezione Stazione con la più alta concentrazione di particolato PM10 nel 2018

La funzione filter_and_aggregate filtra il DataFrame Spark sulla qualità dell'aria del 2018 in base alla lista delle stazioni specificate e il tipo di inquinante. Successivamente, aggrega i dati per stazione calcolando la media dei valori di misurazione per l'inquinante specificato. Questo perchè essendo le rilevazioni giornaliere ed effettuate ogni ora abbiamo deciso di considerare la stazione con la più alta concentrazione di particolato PM10 quella che presenta il valore medio annuale più alto (lo stesso ragionamento verrà ripetuto per il particolato PM2.5). Viene poi aggiunta la colonna del nome della stazione e rinominato il DataFrame risultante con colonne significative. Infine, il DataFrame viene ordinato in base ai valori di PM10 in ordine decrescente, e le informazioni sulla prima stazione vengono estratte per essere restituite insieme al DataFrame.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, translate

def filter_and_aggregate(df, stazione_ids, misura_anno, inquinante_id):
    # Filtra il DataFrame in base ai criteri specificati
    filtered_df = df.filter((col("stazione_id").isin(stazione_ids)) &
                            (col("misura_anno") == misura_anno) &
                            (col("inquinante_id") == inquinante_id))

    # Accesso a df_inquinanti per ottenere inquinante_simbolo
    inquinante_simbolo = str(df_inquinanti.filter(col("inquinante_id") == inquinante_id).select("inquinante_simbolo").fi

    if "." in inquinante_simbolo:
      inquinante_simbolo = inquinante_simbolo.replace(".", " ")

    # Rinomina la colonna risultante con inquinante_simbolo
    result_df = filtered_df.groupBy("stazione_id").agg(avg("misura_valore").alias(inquinante_simbolo))

    # Aggiungi la colonna stazione_nome basata su df_stazioni
    result_df = result_df.join(df_stazioni.select("stazione_id", "stazione_nome"), "stazione_id", "left_outer")

    # Riordina le colonne e rinomina come desiderato
    result_df = result_df.select("stazione_id", "stazione_nome", inquinante_simbolo).withColumnRenamed("stazione_id", "I

    # Ordina il DataFrame in base a PM10 in ordine decrescente
    result_df = result_df.orderBy(col(inquinante_simbolo).desc())
    first_id = result_df.select("Id Stazione").first()[0]
    first_nome = result_df.select("Nome Stazione").first()[0]

    return result_df, first_id, first_nome
```

```
#Questa funzione restituisce il simbolo dell'inquinante dato in input

def retrieve_inquinante_simbolo(df_inquinanti, inquinante_id):
    # Carica il DataFrame dal MongoDB

    # Filtra il DataFrame in base all'inquinante_id fornito
    result_df = df_inquinanti.filter((col("inquinante_id") == inquinante_id))

    # Verifica se ci sono risultati
    if result_df.count() > 0:
        # Recupera il valore di inquinante_simbolo dalla prima riga (supponendo che ci sia al massimo una corrispondenza
        inquinante_simbolo = result_df.select("inquinante_simbolo").first()["inquinante_simbolo"]
        print(f"Inquinante Simbolo per id:{inquinante_id} è {inquinante_simbolo}")
        return inquinante_simbolo
    else:
        print(f"Nessun risultato trovato per inquinante_id {inquinante_id}")
        return None


misura_anno = 2018  # Specifica l'anno di misura desiderato
inquinante_id_pm10 = 5  # Specifica l'inquinante_id desiderato PM10
df_high_pm10, pm10_id_stazione, pm10_nome_stazione  = filter_and_aggregate(df_aria, lista_stazioni_id, misura_anno, inqu
simbolo_pm10 = retrieve_inquinante_simbolo(df_inquinanti, inquinante_id_pm10)
# Visualizza il risultato
df_high_pm10.show()
print(f'La stazione con la più alta concentrazione nel {misura_anno} di particolato {simbolo_pm10} è: {pm10_nome_stazion
```

```
Inquinante Simbolo per id:5 è PM10
+-----------+------------------+-----------------+
|Id Stazione|      Nome Stazione|             PM10|
+-----------+------------------+-----------------+
|    1908795|CT - Viale Vittor...|27.069017075019598|
|    1908371|       a2a - Milazzo|25.010050656228497|
|    1908701|        Misterbianco|23.451274032909907|
|         14|    ME - Villa Dante|23.332698812310202|
|    1908702|  CT - Parco Gioieni| 21.85377187420584|
|    1908312|a2a - S.Filippo d...|21.627203196347132|
|    1908367|       ME - Boccetta|21.516768358371998|
|    1908369|   Milazzo - Termica|20.914164716805587|
```

```
|    1908366|       Pace del Mela| 19.58898150262621|
|    1908601|               Enna|14.743821852020027|
+-----------+--------------------+------------------+

La stazione con la più alta concentrazione nel 2018 di particolato PM10 è: CT - Viale Vittorio Veneto
```

3.1 Selezione Stazione con la più alta concentrazione di particolato PM2.5 nel 2018

Applichiamo la funzione per recuperare la stazione con la più alta concentrazione di particolato PM2.5 nel 2018

```
# Specifica i valori desiderati per la funzione
inquinante_id_pm25 = 6001  # Specifica l'inquinante_id desiderato PM2.5

df_high_pm25, pm25_id_stazione, pm25_nome_stazione  = filter_and_aggregate(df_aria, lista_stazioni_id, misura_anno, inqu
simbolo_pm25 = retrieve_inquinante_simbolo(df_inquinanti, inquinante_id_pm25)
# Visualizza il risultato
df_high_pm25.show()
print(f'La stazione con la più alta concentrazione nel {misura_anno} di particolato {simbolo_pm25} è: {pm25_nome_stazion
```

```
Inquinante Simbolo per id:6001 è PM2.5
+-----------+--------------------+------------------+
|Id Stazione|       Nome Stazione|            PM2 5|
+-----------+--------------------+------------------+
|    1908701|        Misterbianco| 12.41637914383442|
|    1908312|a2a - S.Filippo d...|11.845582191780762|
|    1908366|       Pace del Mela|11.840888838112683|
|    1908371|       a2a - Milazzo| 11.15884185027115|
|    1908601|               Enna| 7.928248463941201|
+-----------+--------------------+------------------+

La stazione con la più alta concentrazione nel 2018 di particolato PM2.5 è: Misterbianco
```

3.2.1 Grafici sull'andamento delle stazioni durante l'anno 2018

Rilevazioni nel 2018 di particolato *PM10 e PM2.5*

La funzione load_and_filter_aria_data carica i dati relativi alla qualità dell'aria. Successivamente, filtra il DataFrame in base all'identificativo dell'inquinante e della stazione specificati, eseguendo la selezione delle colonne "misura_dataora" e "misura_valore". Infine, ordina il DataFrame in base alla data e restituisce il DataFrame risultante. La funzione fornisce anche informazioni sullo schema del DataFrame, il conteggio delle righe e visualizza le prime 5 righe del DataFrame filtrato.

```
from pyspark.sql.functions import col

def load_and_filter_aria_data(collection_name, inquinante_id, stazione_id):
    spark = SparkSession.builder.appName("LoadAndFilterAriaData").getOrCreate()

    df_aria_filtered = (spark.read.format("mongo")
        .option("collection", collection_name)
        .load()
        .drop('_id', 'periodo_media')  # Drop delle colonne '_id' e 'periodo_media'
        .filter((col("inquinante_id") == inquinante_id) & (col("stazione_id") == stazione_id))
        .select("misura_dataora", "misura_valore")
        .orderBy("misura_dataora"))

    df_aria_filtered.printSchema()
    print(f"Count del DataFrame: {df_aria_filtered.count()}")
    df_aria_filtered.show(5)

    return df_aria_filtered


#Particolato PM 10
collection_name = "aria_2018"

aria2018_df_pm10 = load_and_filter_aria_data(collection_name, inquinante_id_pm10, pm10_id_stazione)
```

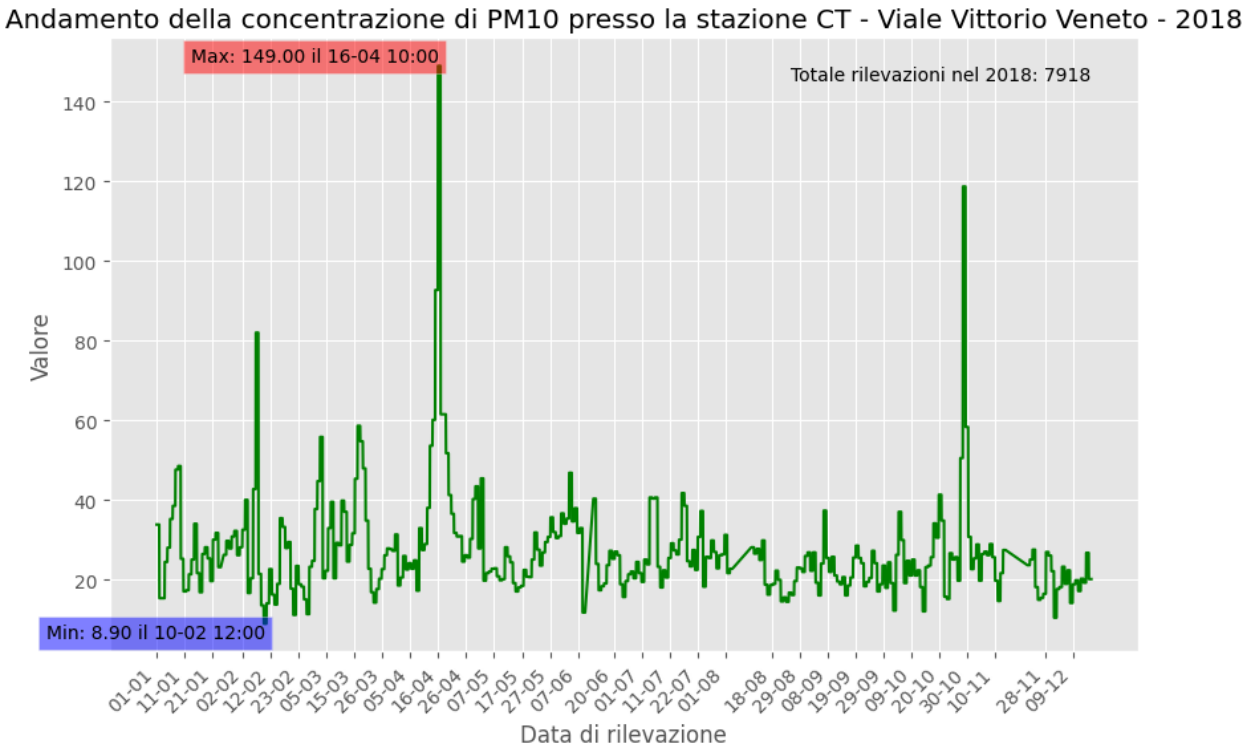
```
 root
  |-- misura_dataora: timestamp (nullable = true)
  |-- misura_valore: double (nullable = true)
```

```
Count del DataFrame: 7918
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2018-01-01 01:00:00|     33.79999|
|2018-01-01 02:00:00|     33.79999|
|2018-01-01 03:00:00|     33.79999|
|2018-01-01 04:00:00|     33.79999|
|2018-01-01 05:00:00|     33.79999|
+-------------------+-------------+
only showing top 5 rows
```

La funzione plot_valore genera un grafico a linea per visualizzare l'andamento della concentrazione di un determinato inquinante presso una stazione specifica in un dato anno. Il grafico include etichette per i valori massimi e minimi con le rispettive date, oltre a informazioni sul totale delle rilevazioni nel periodo considerato.

```
from pyspark.sql.functions import col, unix_timestamp, date_format
import matplotlib.pyplot as plt

def plot_valore(df, anno, stazione, inquinante, step=250):
    data = df.toPandas()

    # Trova i valori massimi e minimi con le rispettive date
    max_value_row = df.orderBy(col("misura_valore").desc()).first()
    min_value_row = df.orderBy("misura_valore").first()

    # Imposta il tema a ggplot
    plt.style.use('ggplot')

    # Crea il grafico a linea senza marker
    plt.figure(figsize=(10, 6))
    plt.plot(data["misura_dataora"], data["misura_valore"], linestyle='-', color='green')
    plt.title(f'Andamento della concentrazione di {inquinante} presso la stazione {stazione} - {anno}')
    plt.xlabel('Data di rilevazione')
    plt.ylabel('Valore')

    # Aggiungi etichette per i valori massimi e minimi
    max_formatted_date = max_value_row["misura_dataora"].strftime('%d-%m %H:%M')
    min_formatted_date = min_value_row["misura_dataora"].strftime('%d-%m %H:%M')

    plt.text(max_value_row["misura_dataora"], max_value_row["misura_valore"],
            f'Max: {max_value_row["misura_valore"]:.2f} il {max_formatted_date}',
            bbox=dict(facecolor='red', alpha=0.5), verticalalignment='bottom', horizontalalignment='right')

    plt.text(min_value_row["misura_dataora"], min_value_row["misura_valore"],
            f'Min: {min_value_row["misura_valore"]:.2f} il {min_formatted_date}',
            bbox=dict(facecolor='blue', alpha=0.5), verticalalignment='top', horizontalalignment='right')


    # Imposta il passo per le etichette delle date (ogni 10 giorni in questo caso)
    step = step
    formatted_dates = data["misura_dataora"][::step].dt.strftime('%d-%m')
```

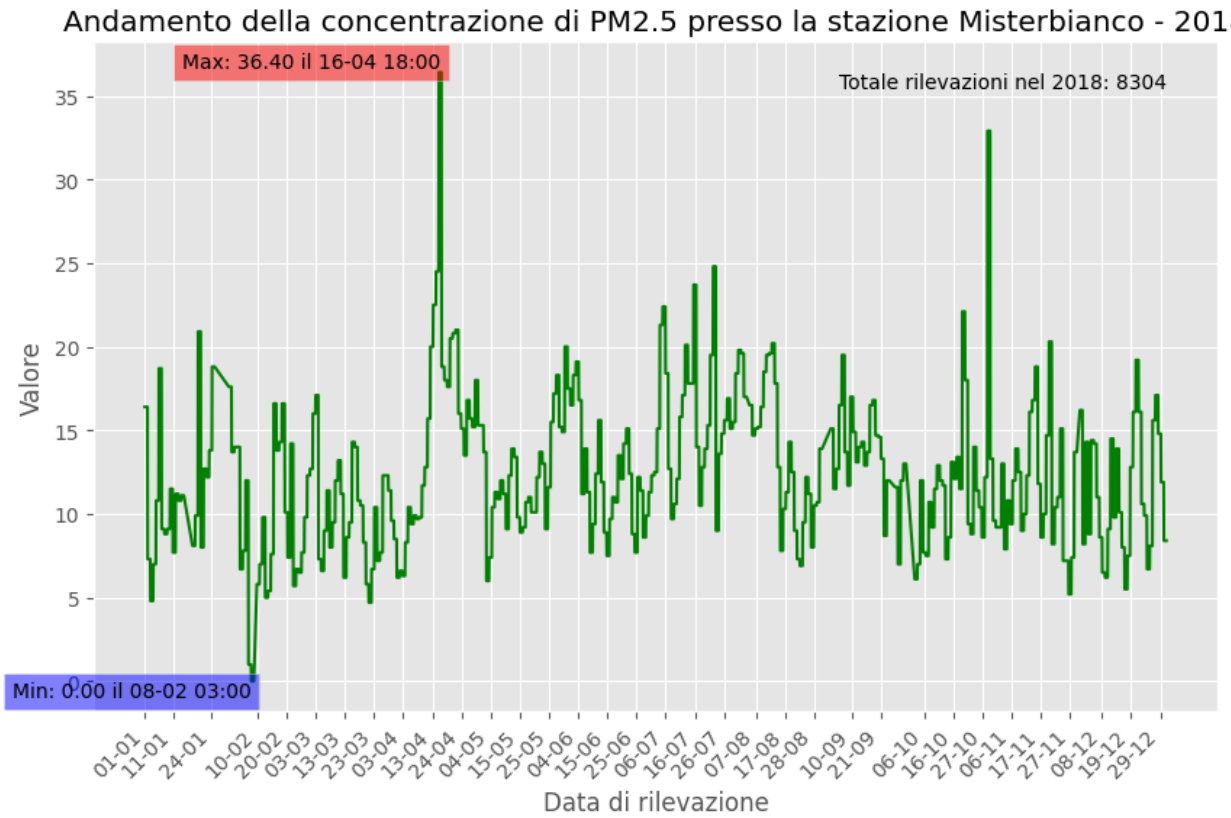
```
    plt.xticks(data["misura_dataora"][::step], formatted_dates, rotation=45, ha='right')


    # Aggiungi il numero totale di date di rilevazione
    plt.text(data["misura_dataora"].iloc[-1], max(data["misura_valore"]),
             f'Totale rilevazioni nel {anno}: {len(data["misura_dataora"])}', verticalalignment='top', horizontalalignme

    plt.show()


plot_valore(aria2018_df_pm10, '2018', pm10_nome_stazione, simbolo_pm10)
```




```
#Particolato PM 2.5
inquinante_id_pm25 = "6001"

aria2018_df_pm25 = load_and_filter_aria_data(collection_name, inquinante_id_pm25, pm25_id_stazione)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 8304
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2018-01-01 01:00:00|  16.39999962|
|2018-01-01 02:00:00|  16.39999962|
|2018-01-01 03:00:00|  16.39999962|
|2018-01-01 04:00:00|  16.39999962|
|2018-01-01 05:00:00|  16.39999962|
+-------------------+-------------+
only showing top 5 rows
```

```
plot_valore(aria2018_df_pm25, '2018', pm25_nome_stazione, simbolo_pm25)
```



3.2.2 Creare un modello MLP

Preparazione dataframe di training e di test per l'inquinante PM10. Utilizzata stazione di Misterbianco perchè nel 2019 non abbiamo rilevazioni per le prime due stazioni con alta concentrazione nel 2018

- Id Stazione Nome Stazione
- 1908795: CT - Viale Vittorio Veneto
- 1908371: a2a - Milazzo
- 1908701: Misterbianco

```
'''
Dataframe:
aria2018_df_pm10
aria2018_df_pm25
'''

collection_name = "aria_2019"
aria2019_df_pm10 = load_and_filter_aria_data(collection_name, inquinante_id_pm10, pm10_id_stazione)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 0
+--------------+-------------+
|misura_dataora|misura_valore|
+--------------+-------------+
+--------------+-------------+
```

```
#Ricerca dati per id stazione:1908371 per la Stazione di:a2a - Milazzo
pm10_id_stazione=1908371

aria2019_df_pm10 = load_and_filter_aria_data(collection_name, inquinante_id_pm10, pm10_id_stazione)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
```

```
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 0
+--------------+-------------+
|misura_dataora|misura_valore|
+--------------+-------------+
+--------------+-------------+
```

```
#Ricerca dati per id stazione:1908701 per la Stazione di: Misterbianco
pm10_id_stazione = pm25_id_stazione #La Stazione di Misterbianco era già stata selezionata per l'inquinante PM2.5
pm10_nome_stazione = pm25_nome_stazione

aria2019_df_pm10 = load_and_filter_aria_data(collection_name, inquinante_id_pm10, pm10_id_stazione)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 8208
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2019-01-01 00:00:00|         11.5|
|2019-01-01 01:00:00|         19.4|
|2019-01-01 02:00:00|         19.4|
|2019-01-01 03:00:00|         19.4|
|2019-01-01 04:00:00|         19.4|
+-------------------+-------------+
only showing top 5 rows
```

```
#creiamo il dataframe per l'anno 2018 con i dati della stazione di Misterbianco
collection_name = "aria_2018"
aria2018_df_pm10 = load_and_filter_aria_data(collection_name, inquinante_id_pm10, pm10_id_stazione)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
```

```
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 8424
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2018-01-01 01:00:00|  23.89999962|
|2018-01-01 02:00:00|  23.89999962|
|2018-01-01 03:00:00|  23.89999962|
|2018-01-01 04:00:00|  23.89999962|
|2018-01-01 05:00:00|  23.89999962|
+-------------------+-------------+
only showing top 5 rows



def filter_rows_by_value_zero(dataframe):
    # Filtra le righe dove misura_valore è zero
    result_dataframe = dataframe.filter(col("misura_valore") == 0)

    return result_dataframe


result_df = filter_rows_by_value_zero(aria2018_df_pm10)

# Visualizza il risultato
print("Righe nel dataframe aria2018_df_pm10 con misura_valore 0 = " + str(result_df.count()))
result_df.show(24)
result_df = filter_rows_by_value_zero(aria2019_df_pm10)

# Visualizza il risultato
print("Righe nel dataframe aria2019_df_pm10 con misura_valore 0 = " + str(result_df.count()))
result_df.show(24)

Righe nel dataframe aria2018_df_pm10 con misura_valore 0 = 24
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2018-02-08 01:00:00|          0.0|
|2018-02-08 02:00:00|          0.0|
```

```
|2018-02-08 03:00:00|          0.0|
|2018-02-08 04:00:00|          0.0|
|2018-02-08 05:00:00|          0.0|
|2018-02-08 06:00:00|          0.0|
|2018-02-08 07:00:00|          0.0|
|2018-02-08 08:00:00|          0.0|
|2018-02-08 09:00:00|          0.0|
|2018-02-08 10:00:00|          0.0|
|2018-02-08 11:00:00|          0.0|
|2018-02-08 12:00:00|          0.0|
|2018-02-08 13:00:00|          0.0|
|2018-02-08 14:00:00|          0.0|
|2018-02-08 15:00:00|          0.0|
|2018-02-08 16:00:00|          0.0|
|2018-02-08 17:00:00|          0.0|
|2018-02-08 18:00:00|          0.0|
|2018-02-08 19:00:00|          0.0|
|2018-02-08 20:00:00|          0.0|
|2018-02-08 21:00:00|          0.0|
|2018-02-08 22:00:00|          0.0|
|2018-02-08 23:00:00|          0.0|
|2018-02-09 00:00:00|          0.0|
+-------------------+-------------+

Righe nel dataframe aria2019_df_pm10 con misura_valore 0 = 24
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2019-09-14 01:00:00|          0.0|
|2019-09-14 02:00:00|          0.0|
|2019-09-14 03:00:00|          0.0|
|2019-09-14 04:00:00|          0.0|
|2019-09-14 05:00:00|          0.0|
|2019-09-14 06:00:00|          0.0|
|2019-09-14 07:00:00|          0.0|
|2019-09-14 08:00:00|          0.0|
|2019-09-14 09:00:00|          0.0|
|2019-09-14 10:00:00|          0.0|
|2019-09-14 11:00:00|          0.0|
|2019-09-14 12:00:00|          0.0|
|2019-09-14 13:00:00|          0.0|
```

```
|2019-09-14 14:00:00|          0.0|
|2019-09-14 15:00:00|          0.0|
|2019-09-14 16:00:00|          0.0|
|2019-09-14 17:00:00|          0.0|
|2019-09-14 18:00:00|          0.0|
|2019-09-14 19:00:00|          0.0|
|2019-09-14 20:00:00|          0.0|
|2019-09-14 21:00:00|          0.0|
|2019-09-14 22:00:00|          0.0|
|2019-09-14 23:00:00|          0.0|
|2019-09-15 00:00:00|          0.0|
```

Il dataframe del 2018 presenta 24 rilevazioni pari a zero nella giornata del 08/02/2018 e il dataframe del 2019 presenta 24 rilevazioni pari a zero nella giornata del 14/09/2019. Probabilmente per quelle giornate si è verificato un anomalia della strumentazione della stazione oppure il livello di particolato nell'aria era prossimo alla zero. Essendo poche rilevazioni non andranno ad inficiare i risultati dei modelli, di conseguenza abbiamo deciso di non eseguire modifiche ai dataframe.

Fai doppio clic (o premi Invio) per modificare

```
#funzione per contare i valori mancanti

def count_null_values(dataframe):
    # Conta il numero di valori mancanti per ciascuna colonna
    null_counts = [dataframe.where(col(column).isNull()).count() for column in dataframe.columns]

    # Calcola il totale dei valori mancanti nel DataFrame
    total_nulls = sum(null_counts)

    if total_nulls == 0:
      print("Il Dataframe non contiene valori mancanti")
    else:
      print(f"Il DataFrame contiene {total_nulls} valori mancanti")


count_null_values(aria2018_df_pm10)
```

```
    Il Dataframe non contiene valori mancanti
```

```
count_null_values(aria2019_df_pm10)
```

```
    Il Dataframe non contiene valori mancanti
```

La funzione df_in_Pandas converte un DataFrame Spark in un DataFrame pandas e crea una colonna aggiuntiva chiamata 'target', che contiene i valori di 'misura_valore' spostati in avanti di un numero specificato di righe (dato da look_back). Successivamente, elimina le righe che contengono valori Null nel DataFrame risultante.

Per i nostri modelli utilizzeremo la colonna misura_valore come features e la colonna target generata rappresenta la "label". Non conosciamo il futuro della serie e in questo modo creiamo la sequenza temporale.

```
def df_in_Pandas(df, look_back):
    pandas_df = df.toPandas()
    pandas_df['target'] = pandas_df['misura_valore'].shift(-look_back)
    # Elimina le righe con valori Null
    pandas_df = pandas_df.dropna()
    return pandas_df
```

```
#sperimentiamo il modello con 3 diversi valori di look_back

lb_1 = 1
lb_12 = 12
lb_24 = 24

train_df_pm10_lb_1 = df_in_Pandas(aria2018_df_pm10, lb_1)
test_df_pm10_lb_1 = df_in_Pandas(aria2019_df_pm10, lb_1)

train_df_pm10_lb_12 = df_in_Pandas(aria2018_df_pm10, lb_12)
test_df_pm10_lb_12 = df_in_Pandas(aria2019_df_pm10, lb_12)

train_df_pm10_lb_24 = df_in_Pandas(aria2018_df_pm10, lb_24)
test_df_pm10_lb_24 = df_in_Pandas(aria2019_df_pm10, lb_24)
```

Questa funzione confronta la forma e le colonne di un DataFrame Spark e di un DataFrame pandas. Stampa le differenze, se presenti.

```
def compare_spark_and_pandas(df_spark, df_pandas):
    # Stampa la forma dei DataFrame
    spark_shape = (df_spark.count(), len(df_spark.columns))
    pandas_shape = df_pandas.shape
    print(f"Shape (Spark): {spark_shape}")
    print(f"Shape (Pandas): {pandas_shape}")

    # Confronta le colonne
    columns_match = set(df_spark.columns) == set(df_pandas.columns)

    # Restituisci le differenze
    if columns_match:
        result =  "Le colonne sono identiche."
    else:
        spark_columns = set(df_spark.columns)
        pandas_columns = set(df_pandas.columns)
        column_diff = {
            "Spark Columns": spark_columns - pandas_columns,
            "Pandas Columns": pandas_columns - spark_columns
        }
        result =  f"Le colonne sono diverse:\n{column_diff}"
    print(result)
```

```
# Chiama la funzione per confrontare i DataFrame
print(f"Look_back: {lb_1}\n")
print("aria2018_df_pm10 e train_df_pm10_lb_1\n")
compare_spark_and_pandas(aria2018_df_pm10, train_df_pm10_lb_1)
print('\n2019_df_pm10, test_df_pm10_lb_1\n')
compare_spark_and_pandas(aria2019_df_pm10, test_df_pm10_lb_1)

print(f"\nLook_back: {lb_12}\n")
print("aria2018_df_pm10 e train_df_pm10_lb_12\n")
compare_spark_and_pandas(aria2018_df_pm10, train_df_pm10_lb_12)
print('\n2019_df_pm10, test_df_pm10_lb_12\n')
compare_spark_and_pandas(aria2019_df_pm10, test_df_pm10_lb_12)

print(f"\nLook_back: {lb_24}\n")
print("aria2018_df_pm10 e train_df_pm10_lb_12\n")
compare_spark_and_pandas(aria2018_df_pm10, train_df_pm10_lb_24)
print('\n2019_df_pm10, test_df_pm10_lb_12\n')
compare_spark_and_pandas(aria2019_df_pm10, test_df_pm10_lb_24)
```

```
Look_back: 1

aria2018_df_pm10 e train_df_pm10_lb_1

Shape (Spark): (8424, 2)
Shape (Pandas): (8423, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

2019_df_pm10, test_df_pm10_lb_1

Shape (Spark): (8208, 2)
Shape (Pandas): (8207, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

Look_back: 12

aria2018_df_pm10 e train_df_pm10_lb_12
```

```
Shape (Spark): (8424, 2)
Shape (Pandas): (8412, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

2019_df_pm10, test_df_pm10_lb_12

Shape (Spark): (8208, 2)
Shape (Pandas): (8196, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

Look_back: 24

aria2018_df_pm10 e train_df_pm10_lb_12

Shape (Spark): (8424, 2)
Shape (Pandas): (8400, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

2019_df_pm10, test_df_pm10_lb_12

Shape (Spark): (8208, 2)
Shape (Pandas): (8184, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}
```

Questa funzione utilizza uno scaler Min-Max di scikit-learn per normalizzare le colonne numeriche di un DataFrame pandas in un intervallo specificato (tra 0 e 1). Restituisce un nuovo DataFrame con i dati normalizzati.

La normalizzazione dei dati è un processo essenziale nell'analisi delle rilevazioni. Questa procedura mira a rendere omogenei i dati, consentendo una comparabilità più accurata e mitigando gli effetti derivanti dalle differenze di scala.

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

scaler = MinMaxScaler(feature_range = (0, 1))

def scale_dataframe(df, scaler):
    # Assicuriamoci di eliminare eventuali colonne non numeriche prima dello scaling
    numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
    df_numeric = df[numerical_columns]

    # Applica lo scaler alle colonne numeriche
    scaled_data = scaler.fit_transform(df_numeric)

    # Crea un nuovo DataFrame con i dati scalati
    df_scaled = pd.DataFrame(scaled_data, columns=numerical_columns)

    return df_scaled


#funzione per trovare il valore massimo e il valore minimo di un dataframe
#Ci serve come test sulla normalizzazione

import numpy as np

def trova_max_min(dataframe):
    max_value = dataframe.values.max()
    min_value = dataframe.values.min()
    return max_value, min_value
```

```
#look back = 1
train_scaler_pm10_lb1 = scale_dataframe(train_df_pm10_lb_1, scaler)
test_scaler_pm10_lb1 = scale_dataframe(test_df_pm10_lb_1, scaler)

print(train_scaler_pm10_lb1.shape)
train_max, train_min = trova_max_min(train_scaler_pm10_lb1)

print(f"Train - Valore Massimo: {train_max}, Valore Minimo: {train_min}")

print(f'\n{test_scaler_pm10_lb1.shape}')
test_max, test_min = trova_max_min(test_scaler_pm10_lb1)
print(f"Test - Valore Massimo: {test_max}, Valore Minimo: {test_min}")

#look back = 12
train_scaler_pm10_lb12 = scale_dataframe(train_df_pm10_lb_12, scaler)
test_scaler_pm10_lb12 = scale_dataframe(test_df_pm10_lb_12, scaler)

print('\n')
print(train_scaler_pm10_lb12.shape)
train_max, train_min = trova_max_min(train_scaler_pm10_lb12)

print(f"Train - Valore Massimo: {train_max}, Valore Minimo: {train_min}")

print(f'\n{test_scaler_pm10_lb12.shape}')
test_max, test_min = trova_max_min(test_scaler_pm10_lb12)
print(f"Test - Valore Massimo: {test_max}, Valore Minimo: {test_min}")

#look back = 24
train_scaler_pm10_lb24 = scale_dataframe(train_df_pm10_lb_24, scaler)
test_scaler_pm10_lb24 = scale_dataframe(test_df_pm10_lb_24, scaler)
print('\n')
print(train_scaler_pm10_lb24.shape)
train_max, train_min = trova_max_min(train_scaler_pm10_lb24)

print(f"Train - Valore Massimo: {train_max}, Valore Minimo: {train_min}")

print(f'\n{test_scaler_pm10_lb24.shape}')
```

```
test_max, test_min = trova_max_min(test_scaler_pm10_lb24)
print(f"Test - Valore Massimo: {test_max}, Valore Minimo: {test_min}")
```

```
(8423, 2)
Train - Valore Massimo: 1.0, Valore Minimo: 0.0

(8207, 2)
Test - Valore Massimo: 1.0, Valore Minimo: 0.0


(8412, 2)
Train - Valore Massimo: 1.0, Valore Minimo: 0.0

(8196, 2)
Test - Valore Massimo: 1.0, Valore Minimo: 0.0


(8400, 2)
Train - Valore Massimo: 1.0, Valore Minimo: 0.0

(8184, 2)
Test - Valore Massimo: 1.0, Valore Minimo: 0.0
```

La funzione prepare_dataframes estrae colonne specifiche dai dataframes di addestramento e test, stampa le dimensioni delle colonne selezionate, e restituisce quattro array corrispondenti a tali colonne.

```
def prepare_dataframes(train_df, test_df):
    X_train_lb = train_df.iloc[:, 0:1].values
    y_train_lb = train_df.iloc[:, -1:].values
    X_test_lb = test_df.iloc[:, 0:1].values
    y_test_lb = test_df.iloc[:, -1:].values

    print(X_train_lb.shape)
    print(y_train_lb.shape)
    print(X_test_lb.shape)
    print(y_test_lb.shape)

    return X_train_lb, y_train_lb, X_test_lb, y_test_lb


X_train_pm10_lb1, y_train_pm10_lb1, X_test_pm10_lb1, y_test_pm10_lb1 = prepare_dataframes(train_scaler_pm10_lb1, test_sc
```

```
(8423, 1)
(8423, 1)
(8207, 1)
(8207, 1)
```

```
X_train_pm10_lb12, y_train_pm10_lb12, X_test_pm10_lb12, y_test_pm10_lb12 = prepare_dataframes(train_scaler_pm10_lb12, te
```

```
(8412, 1)
(8412, 1)
(8196, 1)
(8196, 1)
```

```
X_train_pm10_lb24, y_train_pm10_lb24, X_test_pm10_lb24, y_test_pm10_lb24 = prepare_dataframes(train_scaler_pm10_lb24, te
```

```
(8400, 1)
(8400, 1)
(8184, 1)
(8184, 1)
```

```
#pip install tqdm

#Questa libreria viene utilizzata per visualizzare delle barre di avanzamento durante il training del modello
```

La funzione train_and_evaluate_model implementa l'addestramento e la valutazione di un modello di rete neurale multistrato (MLP) utilizzando PyTorch. Il modello viene definito con un numero specifico di strati e dimensioni, la funzione di perdita viene impostata come l'errore quadratico medio (MSE), e l'ottimizzatore è Adam. Durante l'addestramento, vengono registrati i valori di loss e, alla fine, il modello viene valutato sui dati di test, stampando il Mean Squared Error (MSE) e visualizzando il grafico della loss nel corso delle epoche. Infine, il modello addestrato, la funzione di perdita, e l'ottimizzatore vengono restituiti insieme alle predizioni del modello.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from tqdm import tqdm
from pyspark.sql.functions import hour, dayofyear
from pyspark.ml.feature import VectorAssembler

def train_and_evaluate_model(scaler, X_train, y_train, X_test, y_test, input_size, hidden_size, output_size, num_epochs=
    # Definisci il modello MLP in PyTorch
    class MLPModel(nn.Module):
        def __init__(self, input_size, hidden_size, output_size):
            super(MLPModel, self).__init__()
            self.fc1 = nn.Linear(input_size, hidden_size)
            self.relu = nn.ReLU()
            self.fc2 = nn.Linear(hidden_size, output_size)

        def forward(self, x):
            out = self.fc1(x)
            out = self.relu(out)
            out = self.fc2(out)
            return out

    model = MLPModel(input_size, hidden_size, output_size)

    # Definisci la funzione di perdita e l'ottimizzatore
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Liste per memorizzare i valori della loss
    train_loss_list = []

    # Addestra il modello
    for epoch in tqdm(range(num_epochs), desc="Training Progress"):
        # Forward pass
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
```

```
        # Backward pass e ottimizzazione
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Aggiungi il valore della loss alla lista
        train_loss_list.append(loss.item())

        # Stampa la perdita ogni 100 epoche
        if (epoch + 1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

    # Valuta il modello sul DataFrame di test
    with torch.no_grad():
        y_predicted = model(X_test)
        y_predicted = np.array(y_predicted)
        y_test = np.array(y_test)
        predicted_model = [i[0] for i in y_predicted]
        data_test = [i[0] for i in y_test]
        df_predicted = pd.DataFrame({
            'Osservati': data_test,
            'Previsione': predicted_model
        })
        df_predicted = pd.DataFrame(scaler.inverse_transform(df_predicted), columns=['Osservati', 'Previsione'])
        y_predicted = df_predicted['Previsione']
        y_test = df_predicted['Osservati']
        y_predicted = torch.tensor(y_predicted, dtype=torch.float32)
        y_test = torch.tensor(y_test, dtype=torch.float32)
        mse = criterion(y_predicted, y_test)
        print(f'Mean Squared Error (MSE) on test data rescaled: {mse.item()}')

    # Stampare il grafico della loss
    plt.plot(train_loss_list, label='Training Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training Loss Over Epochs')
```

```
    plt.legend()
    plt.show()

    return y_predicted, y_test, model, criterion, optimizer, mse.item()
```

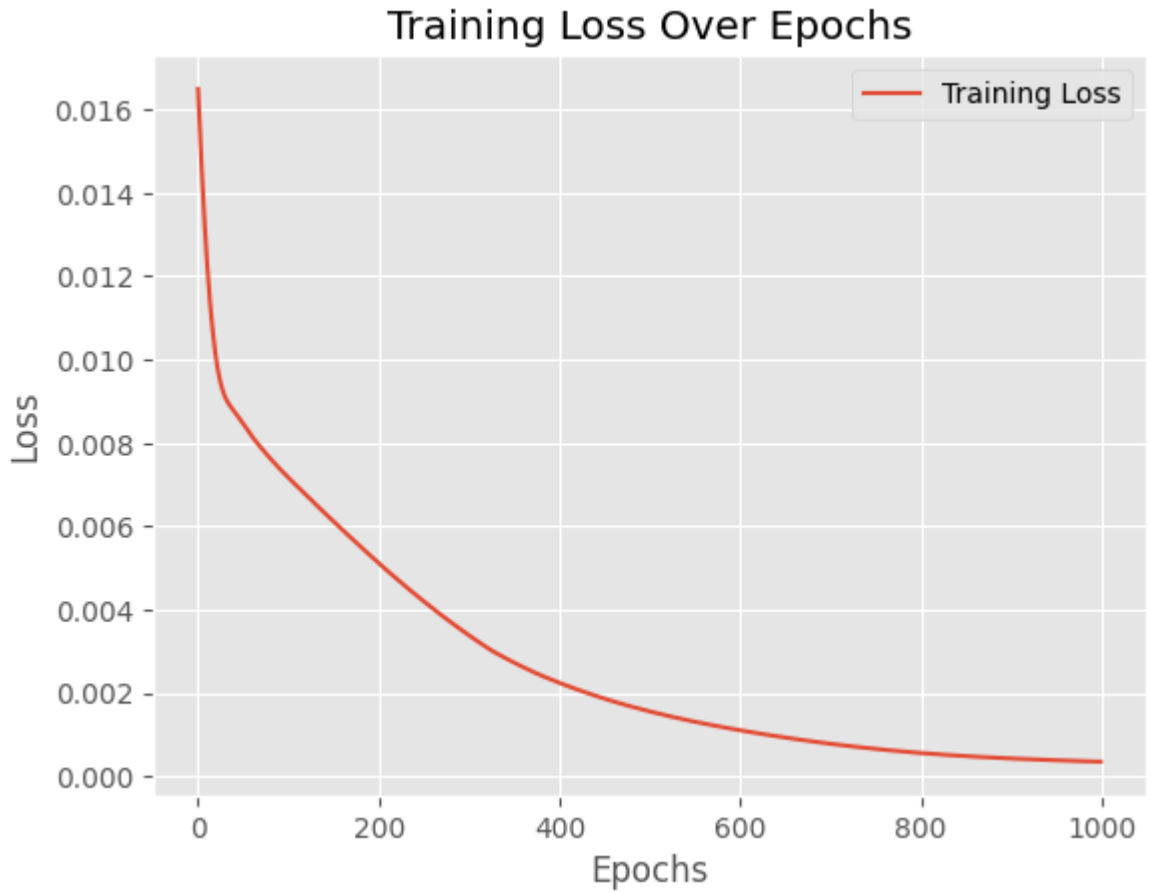
Convertiamo le colonne in tensori PyTorch

```
# Converte le colonne in tensori PyTorch
X_train_pm10_tensor_lb_1 = torch.tensor(X_train_pm10_lb1, dtype=torch.float32)
y_train_pm10_tensor_lb_1 = torch.tensor(y_train_pm10_lb1, dtype=torch.float32)
X_test_pm10_tensor_lb_1 = torch.tensor(X_test_pm10_lb1, dtype=torch.float32)
y_test_pm10_tensor_lb_1 = torch.tensor(y_test_pm10_lb1, dtype=torch.float32)

X_train_pm10_tensor_lb_12 = torch.tensor(X_train_pm10_lb12, dtype=torch.float32)
y_train_pm10_tensor_lb_12 = torch.tensor(y_train_pm10_lb12, dtype=torch.float32)
X_test_pm10_tensor_lb_12 = torch.tensor(X_test_pm10_lb12, dtype=torch.float32)
y_test_pm10_tensor_lb_12 = torch.tensor(y_test_pm10_lb12, dtype=torch.float32)

X_train_pm10_tensor_lb_24 = torch.tensor(X_train_pm10_lb24, dtype=torch.float32)
y_train_pm10_tensor_lb_24 = torch.tensor(y_train_pm10_lb24, dtype=torch.float32)
X_test_pm10_tensor_lb_24 = torch.tensor(X_test_pm10_lb24, dtype=torch.float32)
y_test_pm10_tensor_lb_24 = torch.tensor(y_test_pm10_lb24, dtype=torch.float32)
```

Applichiamo la funzione per il training dei modelli MLP

```
#lb=1
y_predicted_MLP_pm10_lb1, y_test_pm10_lb1_rescaled, model_MLP_pm10_lb1, criterion_MLP_pm10_lb1, optimizer_MLP_pm10_lb1,
```

```
Training Progress:  15%|█          | 153/1000 [00:00<00:01, 507.29it/s]Epoch
Epoch [200/1000], Loss: 0.0051
Training Progress:  37%|████       | 373/1000 [00:00<00:01, 544.51it/s]Epoch
Epoch [400/1000], Loss: 0.0023
Training Progress:  60%|██████     | 605/1000 [00:01<00:00, 567.65it/s]Epoch
Epoch [600/1000], Loss: 0.0011
Training Progress:  78%|████████   | 776/1000 [00:01<00:00, 551.78it/s]Epoch
Training Progress:  83%|████████   | 832/1000 [00:01<00:00, 370.59it/s]Epoch
Training Progress:  91%|█████████  | 914/1000 [00:02<00:00, 245.73it/s]Epoch
Training Progress: 100%|██████████| 1000/1000 [00:02<00:00, 404.08it/s]
Epoch [1000/1000], Loss: 0.0004
Mean Squared Error (MSE) on test data rescaled: 14.768537521362305
```

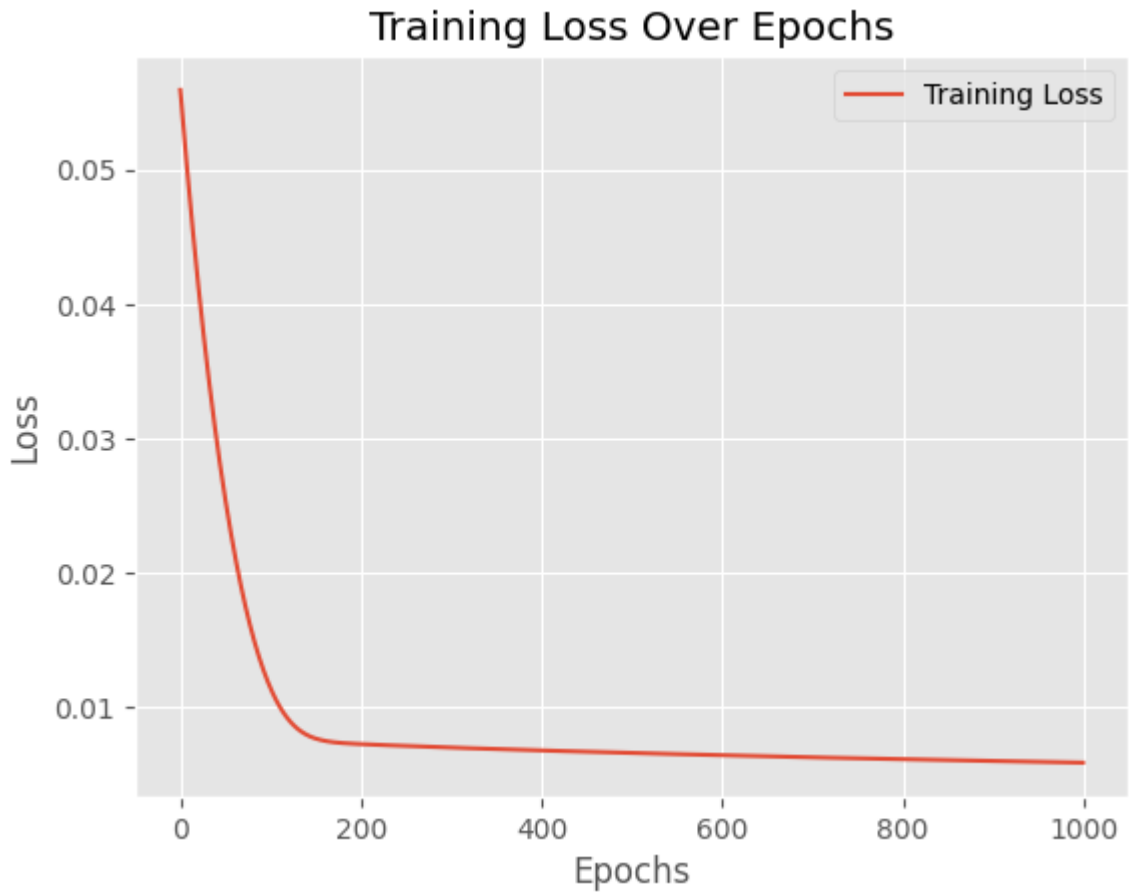


```
#lb=12
y_predicted_MLP_pm10_lb12, y_test_pm10_lb12_rescaled, model_MLP_pm10_lb12, criterion_MLP_pm10_lb12, optimizer_MLP_pm10_l
```

```
Training Progress:  22%|██        | 225/1000 [00:00<00:01, 748.30it/s]Epoch
Epoch [200/1000], Loss: 0.0051
Training Progress:  37%|███       | 374/1000 [00:00<00:00, 705.57it/s]Epoch
Epoch [400/1000], Loss: 0.0043
Training Progress:  59%|█████     | 590/1000 [00:00<00:00, 697.40it/s]Epoch
Epoch [600/1000], Loss: 0.0036
Training Progress:  82%|███████   | 815/1000 [00:01<00:00, 731.89it/s]Epoch
Epoch [800/1000], Loss: 0.0031
Training Progress: 100%|██████████| 1000/1000 [00:01<00:00, 721.61it/s]
Epoch [900/1000], Loss: 0.0030
Epoch [1000/1000], Loss: 0.0029
Mean Squared Error (MSE) on test data rescaled: 119.25532531738281
```

```
#lb=24
y_predicted_MLP_pm10_lb24, y_test_pm10_lb24_rescaled, model_MLP_pm10_lb24, criterion_MLP_pm10_lb24, optimizer_MLP_pm10_l
```

```
Training Progress:  23%|██        | 232/1000 [00:00<00:01, 732.58it/s]Epoch
Epoch [200/1000], Loss: 0.0072
Training Progress:  38%|███       | 380/1000 [00:00<00:00, 724.18it/s]Epoch
Epoch [400/1000], Loss: 0.0068
Training Progress:  60%|██████    | 604/1000 [00:00<00:00, 739.10it/s]Epoch
Epoch [600/1000], Loss: 0.0064
Training Progress:  83%|████████  | 828/1000 [00:01<00:00, 738.30it/s]Epoch
Epoch [800/1000], Loss: 0.0061
Training Progress: 100%|██████████| 1000/1000 [00:01<00:00, 729.13it/s]Epoc
Epoch [1000/1000], Loss: 0.0059
Mean Squared Error (MSE) on test data rescaled: 207.33798217773438
```



La funzione crea_e_stampadf_predicted converte le previsioni del modello (contenute in un tensore PyTorch) in un DataFrame Pandas, crea un DataFrame combinando le date con le colonne osservate e previste.

```
import numpy as np

def crea_e_stampadf_predicted(y_predicted, y_test, df):
    # Converte i tensori PyTorch in un array NumPy
    y_predicted = np.array(y_predicted)
    y_test = np.array(y_test)
    # Crea un DataFrame
    df_predicted = pd.DataFrame({
        #recupera la colonna delle date
        'misura_dataora': df[['misura_dataora']].values.flatten(),
        'Osservati': y_test,
        'Previsione': y_predicted
    })
    # Restituisci il DataFrame risultante
    return df_predicted


#lb=1
predicted_mlp_pm10_lb1 = crea_e_stampadf_predicted(y_predicted_MLP_pm10_lb1, y_test_pm10_lb1_rescaled, test_df_pm10_lb_1
print(predicted_mlp_pm10_lb1)
```

```
            misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00  19.400000   11.350191
1    2019-01-01 01:00:00  19.400000   19.350836
2    2019-01-01 02:00:00  19.400000   19.350836
3    2019-01-01 03:00:00  19.400000   19.350836
4    2019-01-01 04:00:00  19.400000   19.350836
...                  ...        ...         ...
8202 2019-12-31 18:00:00  21.700001   21.680155
8203 2019-12-31 19:00:00  21.700001   21.680155
8204 2019-12-31 20:00:00  21.700001   21.680155
8205 2019-12-31 21:00:00  21.700001   21.680155
8206 2019-12-31 22:00:00  21.700001   21.680155
```

```
[8207 rows x 3 columns]
```

```
#lb=12
predicted_mlp_pm10_lb12 = crea_e_stampadf_predicted(y_predicted_MLP_pm10_lb12, y_test_pm10_lb12_rescaled, test_df_pm10_l
print(predicted_mlp_pm10_lb12)
```

```
            misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00  19.400000   17.384174
1    2019-01-01 01:00:00  19.400000   23.074646
2    2019-01-01 02:00:00  19.400000   23.074646
3    2019-01-01 03:00:00  19.400000   23.074646
4    2019-01-01 04:00:00  19.400000   23.074646
...                  ...        ...         ...
8191 2019-12-31 07:00:00  21.700001   24.731371
8192 2019-12-31 08:00:00  21.700001   24.731371
8193 2019-12-31 09:00:00  21.700001   24.731371
8194 2019-12-31 10:00:00  21.700001   24.731371
8195 2019-12-31 11:00:00  21.700001   24.731371

[8196 rows x 3 columns]
```

```
#lb=24
predicted_mlp_pm10_lb24 = crea_e_stampadf_predicted(y_predicted_MLP_pm10_lb24, y_test_pm10_lb24_rescaled, test_df_pm10_l
print(predicted_mlp_pm10_lb24)
```

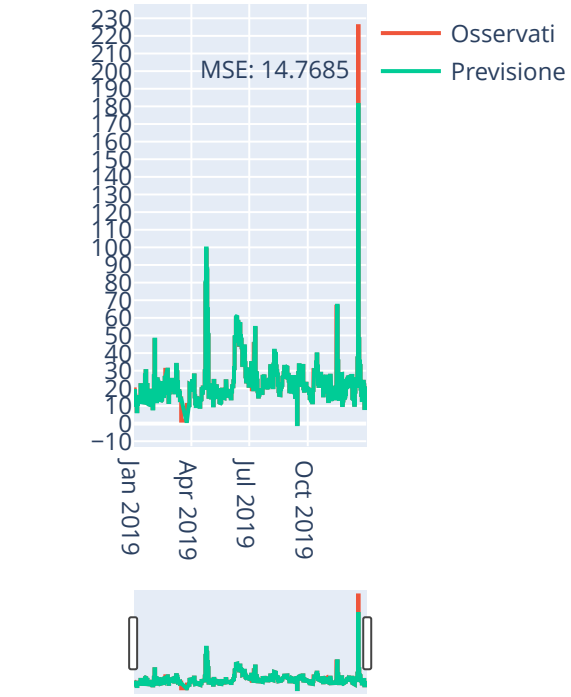
```
            misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00  19.400000   23.427660
1    2019-01-01 01:00:00  15.900000   26.613943
2    2019-01-01 02:00:00  15.900000   26.613943
3    2019-01-01 03:00:00  15.900000   26.613943
4    2019-01-01 04:00:00  15.900000   26.613943
...                  ...        ...         ...
8179 2019-12-30 19:00:00  21.700001   24.032648
8180 2019-12-30 20:00:00  21.700001   24.032648
8181 2019-12-30 21:00:00  21.700001   24.032648
8182 2019-12-30 22:00:00  21.700001   24.032648
8183 2019-12-30 23:00:00  21.700001   24.032648
```

```
[8184 rows x 3 columns]
```

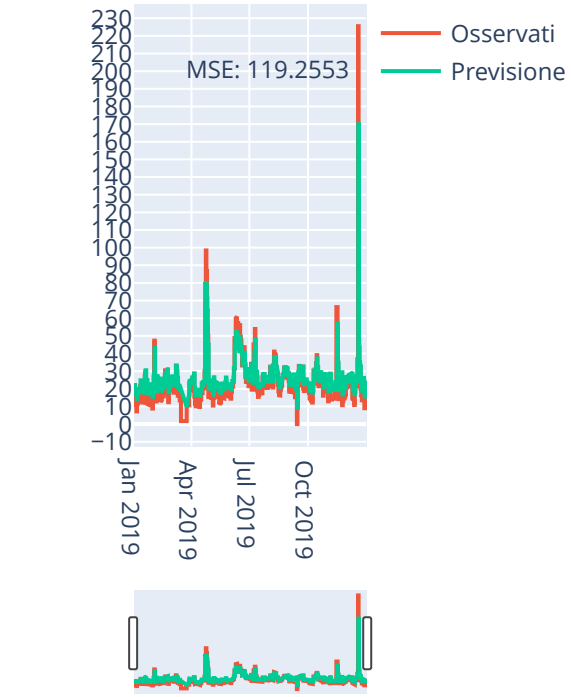
La funzione grafico_interattivo crea un grafico con Plotly Express,imposta le opzioni di zoom per entrambi gli assi e mostra il grafico.

```
import plotly.express as px
import plotly.graph_objects as go

def grafico_interattivo(df, titolo, mse=None):
    grafico = px.line(title=titolo)

    for i in df.columns[1:]:
        grafico.add_scatter(x=df["misura_dataora"], y=df[i], name=i)

    # Aggiungi la possibilità di fare zoom in entrambi gli assi
    grafico.update_layout(
        xaxis=dict(
            rangeslider=dict(
                visible=True,
            ),
            type='date',  # Assicurati che l'asse x sia di tipo 'date'
        ),
        yaxis=dict(
            dtick=10,  # Intervallo tra gli step sull'asse y
        ),
    )

    # Aggiungi la label per il valore MSE
    if mse is not None:
        label_mse = f'MSE: {mse:.4f}'
        grafico.add_annotation(
            text=label_mse,
            xref='paper',
            yref='paper',
            x=0.95,
            y=0.9,
            showarrow=False,
            font=dict(size=12),
        )

    grafico.show()
```

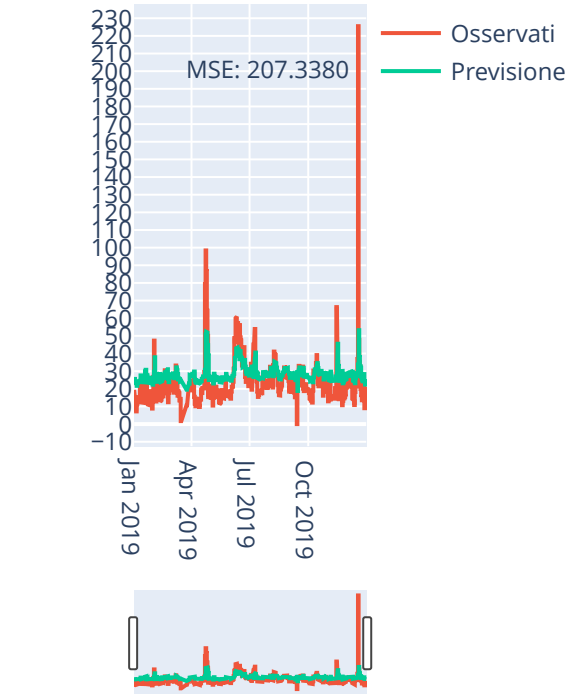
```
grafico_interattivo(predicted_mlp_pm10_lb1, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simbo
```




```
grafico_interattivo(predicted_mlp_pm10_lb12, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simb
```




```
grafico_interattivo(predicted_mlp_pm10_lb24, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simb
```



Scegliamo il modello con l'MSE pari a 14.76

```
print(f'LB=1  - MSE = {mse_mlp_pm10_lb1}')
print(f'LB=12 - MSE = {mse_mlp_pm10_lb12}')
print(f'LB=24 - MSE = {mse_mlp_pm10_lb24}')
```

```
LB=1  - MSE = 14.768537521362305
LB=12 - MSE = 119.25532531738281
LB=24 - MSE = 207.33798217773438
```

```
model_MLP_pm10 = model_MLP_pm10_lb1
criterion_MLP_pm10 = criterion_MLP_pm10_lb1
optimizer_MLP_pm10 = optimizer_MLP_pm10_lb1

X_train_pm10_tensor = X_train_pm10_tensor_lb_1
y_train_pm10_tensor = y_train_pm10_tensor_lb_1
X_test_pm10_tensor = X_test_pm10_tensor_lb_1
y_test_pm10_tensor = y_test_pm10_tensor_lb_1

test_df_pm10 = test_df_pm10_lb_1
test_scaler_pm10 = test_scaler_pm10_lb1
mse_model = {}
mse_model['mlp_pm10'] = mse_mlp_pm10_lb1
```

3.2.3 Predizione per una settimana del 2020 per PM10

inquinante PM 10

Addestriamo il modello MLP dell'inquinante PM10 - model_MLP_pm10 - con i dati del 2019 per prevedere una settimana del 2020

I dati del 2019 attualmente sono memorizzati nei tensori:

- X_test_pm10_tensor
- y_test_pm10_tensor

Oggetti del modello:

- criterion_MLP_pm10 = Criterio funzione di perdita
- optimizer_MLP_pm10 = Ottimizzatore

```
#memorizziamo i tensori per il training e test
X_2019_pm10 = X_test_pm10_tensor
y_2019_pm10 = y_test_pm10_tensor
```

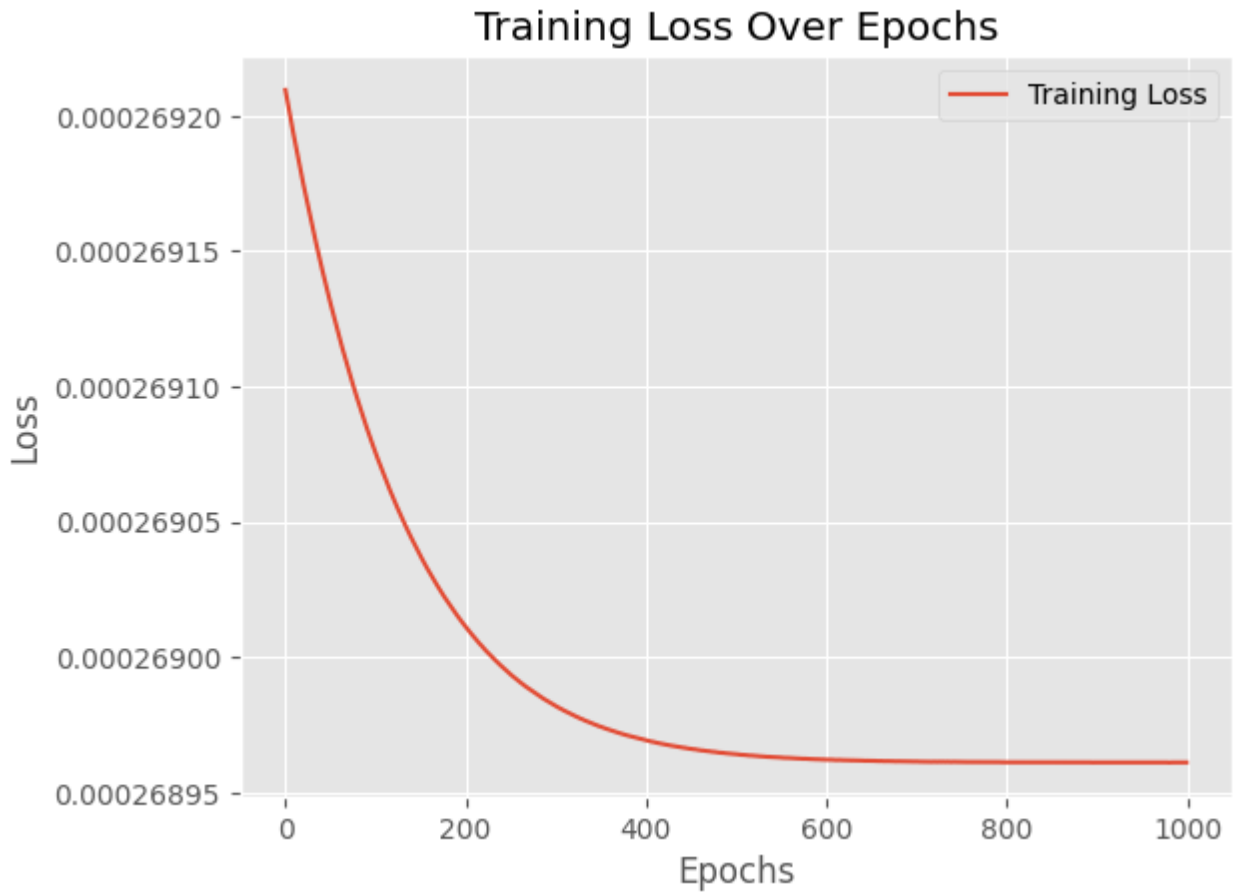
Split del dataset in training e test, con percentuali 70% e 30%

```
from sklearn.model_selection import train_test_split

# Converti i tensori in array NumPy
X_np = X_2019_pm10.numpy()
y_np = y_2019_pm10.numpy()

# Suddividi i dati in set di addestramento e test
X_train, X_test, y_train, y_test = train_test_split(X_np, y_np, test_size=0.3, random_state=42)

# Converti i dati divisi nuovamente in tensori PyTorch
X_train_tensor_2019_pm10 = torch.from_numpy(X_train)
X_test_tensor_2019_pm10 = torch.from_numpy(X_test)
y_train_tensor_2019_pm10 = torch.from_numpy(y_train)
y_test_tensor_2019_pm10 = torch.from_numpy(y_test)
```

```
#La funzione prende in input un modello e altri parametri ed esegue l'addestramento con i dati di train

def new_train_and_evaluate_model(model, scaler, criterion, optimizer, X_train, y_train, X_test, y_test, num_epochs=1000)
    # Liste per memorizzare i valori della loss
    train_loss_list = []
    # Forward pass
    # Addestra il modello
    for epoch in tqdm(range(num_epochs), desc="Training Progress"):
        # Forward pass
        outputs = model(X_train)
        loss = criterion(outputs, y_train)

        # Backward pass e ottimizzazione
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Aggiungi il valore della loss alla lista
        train_loss_list.append(loss.item())

        # Stampa la perdita ogni 100 epoche
        if (epoch + 1) % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

    # Valuta il modello sul DataFrame di test
    with torch.no_grad():
        y_predicted = model(X_test)
        y_predicted = np.array(y_predicted)
        y_test = np.array(y_test)
        predicted_model = [i[0] for i in y_predicted]
        data_test = [i[0] for i in y_test]
        df_predicted = pd.DataFrame({
            'Osservati': data_test,
            'Previsione': predicted_model
        })
        df_predicted = pd.DataFrame(scaler.inverse_transform(df_predicted), columns=['Osservati', 'Previsione'])
        y_predicted = df_predicted['Previsione']
```

```
            y_test = df_predicted['Osservati']
            y_predicted = torch.tensor(y_predicted, dtype=torch.float32)
            y_test = torch.tensor(y_test, dtype=torch.float32)
            mse = criterion(y_predicted, y_test)
            print(f'Mean Squared Error (MSE) on test data rescaled: {mse.item()}')

        # Stampare il grafico della loss
        plt.plot(train_loss_list, label='Training Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.title('Training Loss Over Epochs')
        plt.legend()
        plt.show()

        return model, y_predicted, mse.item()


model_MLP_pm10_train2019, y_predicted_mlp_pm10_2020, mse_mlp_train2019_pm10 = new_train_and_evaluate_model(model_MLP_pm1
```

```
Training Progress:  21%|██        | 212/1000 [00:00<00:01, 614.71it/s]Epoch
Epoch [200/1000], Loss: 0.0003
Training Progress:  40%|████      | 398/1000 [00:00<00:01, 600.23it/s]Epoch
Epoch [400/1000], Loss: 0.0003
Training Progress:  59%|██████    | 589/1000 [00:00<00:00, 621.38it/s]Epoch
Epoch [600/1000], Loss: 0.0003
Training Progress:  78%|███████   | 779/1000 [00:01<00:00, 616.00it/s]Epoch
Epoch [800/1000], Loss: 0.0003
Training Progress: 100%|██████████| 1000/1000 [00:01<00:00, 611.59it/s]Epoc
Epoch [1000/1000], Loss: 0.0003
Mean Squared Error (MSE) on test data rescaled: 0.11914420872926712
```



```
#La funzione prende in input il modello addestrato, la sequenza temporale X di interesse e lo scaler e restituisce la pr

def predict_and_transform(model, X, scaler):
    with torch.no_grad():
        y_predicted = model(X)
        y_predicted = np.array(y_predicted)
        predicted_model = [i[0] for i in y_predicted]

        # Appiattisci y se è bidimensionale
        if y_predicted.ndim > 1:
            y_predicted = y_predicted.flatten()

        df_predicted = pd.DataFrame({
            'Osservati': y_predicted,
            'Previsione': predicted_model
        })
        df_predicted = pd.DataFrame(scaler.inverse_transform(df_predicted), columns=['Osservati', 'Previsione'])

        y_predicted_tensor = torch.tensor(df_predicted['Previsione'].values, dtype=torch.float32)
        y_original_tensor = torch.tensor(df_predicted['Osservati'].values, dtype=torch.float32)

    return y_predicted_tensor, y_original_tensor

# Utilizzo della funzione con i tuoi dati
y_predicted_mlp_pm10_2020, y_2019_pm10 = predict_and_transform(model_MLP_pm10_train2019, X_2019_pm10, scaler)
```

La funzione crea_e_stampadf_predicted_2020 modifica il DataFrame creato da crea_e_stampadf_predicted, incrementando le date di un anno e rinominando la colonna "Osservati" in "Previsione".

```
import pandas as pd

def crea_e_stampadf_predicted_2020(y_predicted, y_test, df):
    # Chiama la tua funzione originale per ottenere il DataFrame base
    df_predicted_base = crea_e_stampadf_predicted(y_predicted, y_test, df)

    # Effettua le modifiche richieste al DataFrame base
    df_predicted_base['misura_dataora'] = df_predicted_base['misura_dataora'] + pd.DateOffset(years=1)
    df_predicted_base = df_predicted_base.drop('Osservati', axis=1)
    df_predicted_base = df_predicted_base.rename(columns={'Osservati': 'Previsione'})

    # Restituisci il DataFrame risultante
    return df_predicted_base


predicted_mlp_2020_pm10 = crea_e_stampadf_predicted_2020(y_predicted_mlp_pm10_2020, y_2019_pm10, test_df_pm10)


#carichiamo i dati del 2020

collection_name = "aria_2020"
aria2020_df_pm10 = load_and_filter_aria_data(collection_name, inquinante_id_pm10, pm10_id_stazione)
count_null_values(aria2020_df_pm10)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 8303
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2020-01-01 01:00:00|  37.59999084|
|2020-01-01 02:00:00|  37.59999084|
|2020-01-01 03:00:00|  37.59999084|
|2020-01-01 04:00:00|  37.59999084|
|2020-01-01 05:00:00|  37.59999084|
+-------------------+-------------+
only showing top 5 rows
```

```
Il Dataframe non contiene valori mancanti
```

La funzione filter_dataframe_by_week_spark filtra un DataFrame Spark per selezionare le righe con date comprese tra reference_date e i sette giorni successivi (per considerare una settimana). Il DataFrame filtrato viene restituito.

```
from pyspark.sql.functions import col, to_timestamp
from datetime import datetime, timedelta  # Importa la classe timedelta

def filter_dataframe_by_week_spark(df, reference_date):
    one_week_later = reference_date + timedelta(days=7)
    return df.filter(
        (col("misura_dataora") >= reference_date) &
        (col("misura_dataora") < one_week_later)
    )


reference_date = datetime.strptime("2020-03-29", "%Y-%m-%d").date()
week_df_pm10 = filter_dataframe_by_week_spark(aria2020_df_pm10, reference_date)
week_df_pm10.printSchema()
print(f"Il DataFrame contiene: {week_df_pm10.count()} rilevazioni")
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Il DataFrame contiene: 168 rilevazioni
```

La funzione filter_dataframe_by_week_pandas opera come la precedente ma su un dataframe Pandas

```
def filter_dataframe_by_week_pandas(df, reference_date):
    # Converte la colonna "misura_dataora" in formato datetime, se non è già in quel formato
    df['misura_dataora2'] = pd.to_datetime(df['misura_dataora'])

    # Converte la data di riferimento in formato datetime
    reference_date = pd.to_datetime(reference_date)

    # Calcola la data una settimana dopo la data di riferimento
    one_week_later = reference_date + pd.DateOffset(weeks=1)

    # Crea una maschera booleana per selezionare le righe nell'intervallo desiderato
    mask = (df['misura_dataora2'] >= reference_date) & (df['misura_dataora2'] < one_week_later)

    # Applica la maschera per ottenere il DataFrame filtrato
    filtered_df = df[mask]

    return filtered_df



df_filtered = filter_dataframe_by_week_pandas(predicted_mlp_2020_pm10, reference_date)
print(f"Il DataFrame filtrato contiene: {len(df_filtered)} righe.")
```

```
 Il DataFrame filtrato contiene: 168 righe.
```

La funzione merge_spark_and_pandas prende in input i due dataframe creati in precedenza e restituisce un nuovo dataframe Pandas combinando i dataframe presi in input

```
def merge_spark_and_pandas(spark_df, pandas_df):
    # Converte il DataFrame Spark in un DataFrame Pandas
    spark_df_pandas = spark_df.toPandas()

    # Converte la colonna "misura_dataora" di df_filtered in formato datetime
    pandas_df['misura_dataora'] = pd.to_datetime(pandas_df['misura_dataora'])

    # Unisce i due DataFrame Pandas sulla colonna "misura_dataora"
    merged_df = pd.merge(spark_df_pandas, pandas_df, on="misura_dataora", how="inner")

    return merged_df


plot_mlp_pm10_predict_week2020 = merge_spark_and_pandas(week_df_pm10, df_filtered)


#Eliminiamo la colonna "misura_dataora2" in quanto duplicata
plot_mlp_pm10_predict_week2020.drop('misura_dataora2', axis=1, inplace=True)
```

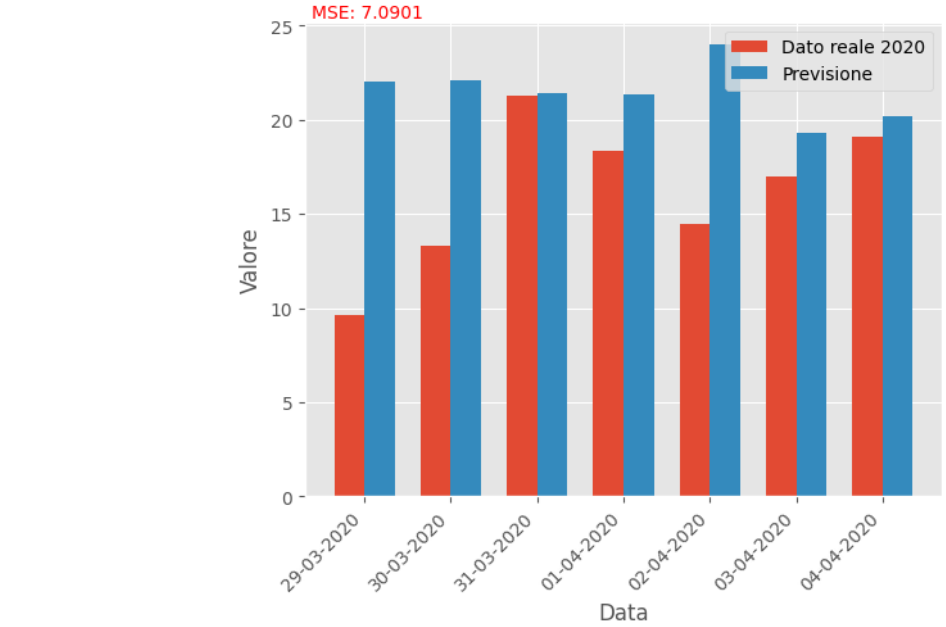
La funzione group_by_date_and_average prende in input il dataframe creato in precedenza, elimina il riferimento orario alle date. Esegue un group by per data facendo la media delle osservazioni orarie.

```
def group_by_date_and_average(df):
    # Copia il DataFrame originale per evitare di modificarlo direttamente
    df_copy = df.copy()
    df_copy.rename(columns={'misura_valore': 'Dato reale 2020'}, inplace=True)

    # Converte la colonna "misura_dataora" in formato datetime
    df_copy['misura_dataora'] = pd.to_datetime(df_copy['misura_dataora'])

    # Estrae la data nel formato "yyyy-mm-dd"
    df_copy['Data'] = df_copy['misura_dataora'].dt.strftime('%Y-%m-%d')

    # Effettua il group by sulla colonna "Data" e calcola la media di "Dato reale 2020" e "Previsione"
    grouped_df = df_copy.groupby('Data').agg({
        'Dato reale 2020': 'mean',
        'Previsione': 'mean'
    }).reset_index()

    return grouped_df


plot_mlp_pm10_predict_week2020_groupby = group_by_date_and_average(plot_mlp_pm10_predict_week2020)


print(plot_mlp_pm10_predict_week2020_groupby)
```

```
           Data  Dato reale 2020  Previsione
0  2020-03-29         9.600000   22.018248
1  2020-03-30        13.337500   22.130013
2  2020-03-31        21.262500   21.446922
3  2020-04-01        18.341667   21.322710
4  2020-04-02        14.462500   23.984728
5  2020-04-03        16.983334   19.339655
6  2020-04-04        19.112501   20.180090
```

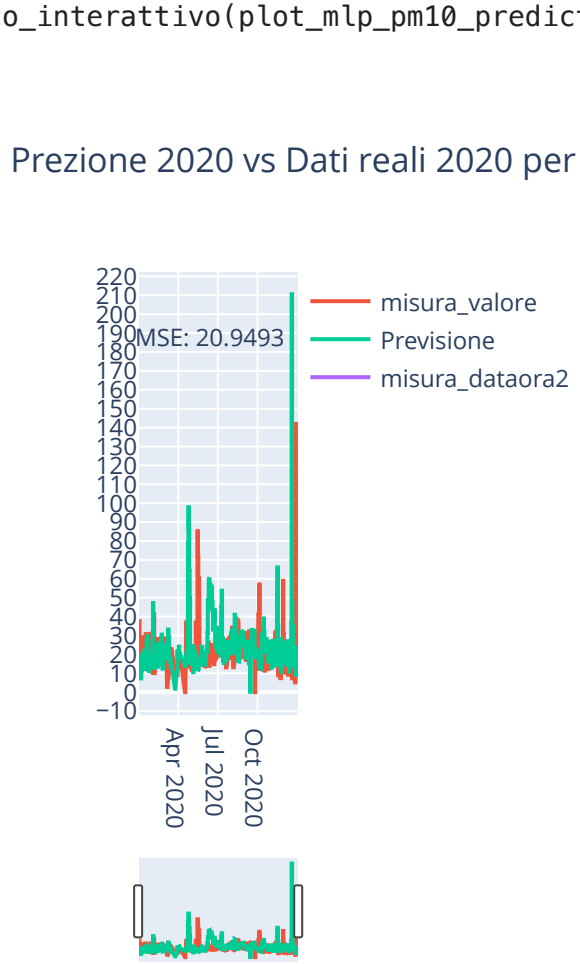
La funzione plot_histogram crea un istogramma per confrontare i dati reali del 2020 con la previsione relativi alla settimana di interesse

```
def plot_histogram(df, inquinante, stazione, modello, mse=None):
    df['Data'] = pd.to_datetime(df['Data']).dt.strftime('%d-%m-%Y')
    # Seleziona le colonne di interesse
    data_col = df['Data']
    real_col = df['Dato reale 2020']
    pred_col = df['Previsione']

    # Imposta la larghezza delle barre e la posizione degli indici
    bar_width = 0.35
    index = np.arange(len(data_col))

    # Crea un istogramma comparativo tra Dato reale 2020 e Previsione
    plt.bar(index, real_col, bar_width, label='Dato reale 2020')
    plt.bar(index + bar_width, pred_col, bar_width, label='Previsione')

    # Aggiungi legenda e titoli
    plt.xlabel('Data')
    plt.ylabel('Valore')
    titolo = 'Confronto tra Dato reale 2020 e Previsione di ' + str(inquinante) + ' per la stazione di ' + stazione +
    plt.title(titolo, y=1.06)
    plt.xticks(index + bar_width / 2, data_col, rotation=45, ha='right')  # Aggiungi etichette delle date

    # Aggiungi label MSE se è stato fornito
    if mse is not None:
        plt.text(0.01, 1.02, f'MSE: {mse:.4f}', transform=plt.gca().transAxes, ha='left', va='center', fontsize=10, colo

    plt.legend()
    plt.show()


from sklearn.metrics import mean_squared_error
import math

mse_mlp_pm10_week2020 = math.sqrt(mean_squared_error(plot_mlp_pm10_predict_week2020['misura_valore'], plot_mlp_pm10_pred


plot_histogram(plot_mlp_pm10_predict_week2020_groupby, simbolo_pm10, pm10_nome_stazione, 'MLP', mse_mlp_pm10_week2020)
```




```
plot_mlp_pm10_predict2020_year = merge_spark_and_pandas(aria2020_df_pm10, predicted_mlp_2020_pm10)
```

```
from sklearn.metrics import mean_squared_error
import math

mse_mlp_pm10_year2020 = math.sqrt(mean_squared_error(plot_mlp_pm10_predict2020_year['misura_valore'], plot_mlp_pm10_pred

titolo = "Prezione 2020 vs Dati reali 2020 per inquinante " + simbolo_pm10 + " presso la stazione di " + pm10_nome_stazi
grafico_interattivo(plot_mlp_pm10_predict2020_year, titolo, mse_mlp_pm10_year2020)
```



3.2.2 Calcolo modello MLP per l'inquinante PM 2.5

```
#Carichiamo i dati del 2019 per l'inquinante PM2.5 per la Stazione con la più alta concentrazione nel 2018

collection_name = "aria_2019"
aria2019_df_pm25 = load_and_filter_aria_data(collection_name, inquinante_id_pm25, pm25_id_stazione)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 8040
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2019-01-01 00:00:00|          8.4|
|2019-01-01 01:00:00|         13.2|
|2019-01-01 02:00:00|         13.2|
|2019-01-01 03:00:00|         13.2|
|2019-01-01 04:00:00|         13.2|
+-------------------+-------------+
only showing top 5 rows
```

```
#controlliamo i valori con rilevazione zero

result_df = filter_rows_by_value_zero(aria2018_df_pm25)
print("Righe nel dataframe aria2018_df_pm25 con misura_valore 0 = " + str(result_df.count()))
result_df.show()

result_df = filter_rows_by_value_zero(aria2019_df_pm25)

print("Righe nel dataframe aria2019_df_pm25 con misura_valore 0 = " + str(result_df.count()))
result_df.show()
```

```
Righe nel dataframe aria2018_df_pm25 con misura_valore 0 = 24
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2018-02-08 01:00:00|          0.0|
|2018-02-08 02:00:00|          0.0|
|2018-02-08 03:00:00|          0.0|
|2018-02-08 04:00:00|          0.0|
|2018-02-08 05:00:00|          0.0|
|2018-02-08 06:00:00|          0.0|
|2018-02-08 07:00:00|          0.0|
|2018-02-08 08:00:00|          0.0|
|2018-02-08 09:00:00|          0.0|
|2018-02-08 10:00:00|          0.0|
|2018-02-08 11:00:00|          0.0|
|2018-02-08 12:00:00|          0.0|
|2018-02-08 13:00:00|          0.0|
|2018-02-08 14:00:00|          0.0|
|2018-02-08 15:00:00|          0.0|
|2018-02-08 16:00:00|          0.0|
|2018-02-08 17:00:00|          0.0|
|2018-02-08 18:00:00|          0.0|
|2018-02-08 19:00:00|          0.0|
|2018-02-08 20:00:00|          0.0|
+-------------------+-------------+
only showing top 20 rows

Righe nel dataframe aria2019_df_pm25 con misura_valore 0 = 144
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2019-03-24 01:00:00|          0.0|
|2019-03-24 02:00:00|          0.0|
|2019-03-24 03:00:00|          0.0|
|2019-03-24 04:00:00|          0.0|
|2019-03-24 05:00:00|          0.0|
|2019-03-24 06:00:00|          0.0|
|2019-03-24 07:00:00|          0.0|
|2019-03-24 08:00:00|          0.0|
|2019-03-24 09:00:00|          0.0|
|2019-03-24 10:00:00|          0.0|
```

```
|2019-03-24 11:00:00|          0.0|
|2019-03-24 12:00:00|          0.0|
|2019-03-24 13:00:00|          0.0|
|2019-03-24 14:00:00|          0.0|
|2019-03-24 15:00:00|          0.0|
|2019-03-24 16:00:00|          0.0|
|2019-03-24 17:00:00|          0.0|
|2019-03-24 18:00:00|          0.0|
|2019-03-24 19:00:00|          0.0|
|2019-03-24 20:00:00|          0.0|
+-------------------+-------------+
only showing top 20 rows
```

Essendo nel 2019 presenti 144 rilevazioni con valore zero. Utilizziamo lo stesso approccio seguito per l'inquinante PM10.

Proseguiamo con un check sui valori mancanti nei dataframe

```
count_null_values(aria2018_df_pm25)
```

```
Il Dataframe non contiene valori mancanti
```

```
count_null_values(aria2019_df_pm25)
```

```
Il Dataframe non contiene valori mancanti
```

```
#sperimentiamo il modello con 3 diversi valori di look_back

lb_1 = 1
lb_12 = 12
lb_24 = 24

train_df_pm25_lb_1 = df_in_Pandas(aria2018_df_pm25, lb_1)
test_df_pm25_lb_1 = df_in_Pandas(aria2019_df_pm25, lb_1)

train_df_pm25_lb_12 = df_in_Pandas(aria2018_df_pm25, lb_12)
test_df_pm25_lb_12 = df_in_Pandas(aria2019_df_pm25, lb_12)

train_df_pm25_lb_24 = df_in_Pandas(aria2018_df_pm25, lb_24)
test_df_pm25_lb_24 = df_in_Pandas(aria2019_df_pm25, lb_24)


# Chiama la funzione per confrontare i DataFrame
print(f"Look_back: {lb_1}\n")
print("aria2018_df_pm25 e train_df_pm25_lb_1\n")
compare_spark_and_pandas(aria2018_df_pm25, train_df_pm25_lb_1)
print('\n2019_df_pm25, test_df_pm25_lb_1\n')
compare_spark_and_pandas(aria2019_df_pm25, test_df_pm25_lb_1)

print(f"\nLook_back: {lb_12}\n")
print("aria2018_df_pm25 e train_df_pm25_lb_12\n")
compare_spark_and_pandas(aria2018_df_pm25, train_df_pm25_lb_12)
print('\n2019_df_pm25, test_df_pm25_lb_12\n')
compare_spark_and_pandas(aria2019_df_pm25, test_df_pm25_lb_12)

print(f"\nLook_back: {lb_24}\n")
print("aria2018_df_pm25 e train_df_pm25_lb_12\n")
compare_spark_and_pandas(aria2018_df_pm25, train_df_pm25_lb_24)
print('\n2019_df_pm25, test_df_pm25_lb_12\n')
compare_spark_and_pandas(aria2019_df_pm25, test_df_pm25_lb_24)
```

```
    Look_back: 1

    aria2018_df_pm25 e train_df_pm25_lb_1
```

```
Shape (Spark): (8304, 2)
Shape (Pandas): (8303, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

2019_df_pm25, test_df_pm25_lb_1

Shape (Spark): (8040, 2)
Shape (Pandas): (8039, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

Look_back: 12

aria2018_df_pm25 e train_df_pm25_lb_12

Shape (Spark): (8304, 2)
Shape (Pandas): (8292, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

2019_df_pm25, test_df_pm25_lb_12

Shape (Spark): (8040, 2)
Shape (Pandas): (8028, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

Look_back: 24

aria2018_df_pm25 e train_df_pm25_lb_12

Shape (Spark): (8304, 2)
Shape (Pandas): (8280, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}

2019_df_pm25, test_df_pm25_lb_12

Shape (Spark): (8040, 2)
```

```
Shape (Pandas): (8016, 3)
Le colonne sono diverse:
{'Spark Columns': set(), 'Pandas Columns': {'target'}}
```

Normalizziamo i dati

```
#look back = 1
train_scaler_pm25_lb1 = scale_dataframe(train_df_pm25_lb_1, scaler)
test_scaler_pm25_lb1 = scale_dataframe(test_df_pm25_lb_1, scaler)

print(train_scaler_pm25_lb1.shape)
train_max, train_min = trova_max_min(train_scaler_pm25_lb1)

print(f"Train - Valore Massimo: {train_max}, Valore Minimo: {train_min}")

print(f'\n{test_scaler_pm25_lb1.shape}')
test_max, test_min = trova_max_min(test_scaler_pm25_lb1)
print(f"Test - Valore Massimo: {test_max}, Valore Minimo: {test_min}")

#look back = 12
train_scaler_pm25_lb12 = scale_dataframe(train_df_pm25_lb_12, scaler)
test_scaler_pm25_lb12 = scale_dataframe(test_df_pm25_lb_12, scaler)

print('\n')
print(train_scaler_pm25_lb12.shape)
train_max, train_min = trova_max_min(train_scaler_pm25_lb12)

print(f"Train - Valore Massimo: {train_max}, Valore Minimo: {train_min}")

print(f'\n{test_scaler_pm25_lb12.shape}')
test_max, test_min = trova_max_min(test_scaler_pm25_lb12)
print(f"Test - Valore Massimo: {test_max}, Valore Minimo: {test_min}")

#look back = 24
train_scaler_pm25_lb24 = scale_dataframe(train_df_pm25_lb_24, scaler)
test_scaler_pm25_lb24 = scale_dataframe(test_df_pm25_lb_24, scaler)
print('\n')
print(train_scaler_pm25_lb24.shape)
train_max, train_min = trova_max_min(train_scaler_pm25_lb24)

print(f"Train - Valore Massimo: {train_max}, Valore Minimo: {train_min}")

print(f'\n{test_scaler_pm25_lb24.shape}')
```

```
test_max, test_min = trova_max_min(test_scaler_pm25_lb24)
print(f"Test - Valore Massimo: {test_max}, Valore Minimo: {test_min}")
```

```
(8303, 2)
Train - Valore Massimo: 1.0, Valore Minimo: 0.0

(8039, 2)
Test - Valore Massimo: 1.0, Valore Minimo: 0.0


(8292, 2)
Train - Valore Massimo: 1.0, Valore Minimo: 0.0

(8028, 2)
Test - Valore Massimo: 1.0, Valore Minimo: 0.0


(8280, 2)
Train - Valore Massimo: 1.0, Valore Minimo: 0.0

(8016, 2)
Test - Valore Massimo: 1.0, Valore Minimo: 0.0
```

```
X_train_pm25_lb1, y_train_pm25_lb1, X_test_pm25_lb1, y_test_pm25_lb1 = prepare_dataframes(train_scaler_pm25_lb1, test_sc
```

```
(8303, 1)
(8303, 1)
(8039, 1)
(8039, 1)
```

```
X_train_pm25_lb12, y_train_pm25_lb12, X_test_pm25_lb12, y_test_pm25_lb12 = prepare_dataframes(train_scaler_pm25_lb12, te
```

```
(8292, 1)
(8292, 1)
(8028, 1)
(8028, 1)
```

```
X_train_pm25_lb24, y_train_pm25_lb24, X_test_pm25_lb24, y_test_pm25_lb24 = prepare_dataframes(train_scaler_pm25_lb24, te
```

```
(8280, 1)
(8280, 1)
(8016, 1)
(8016, 1)
```

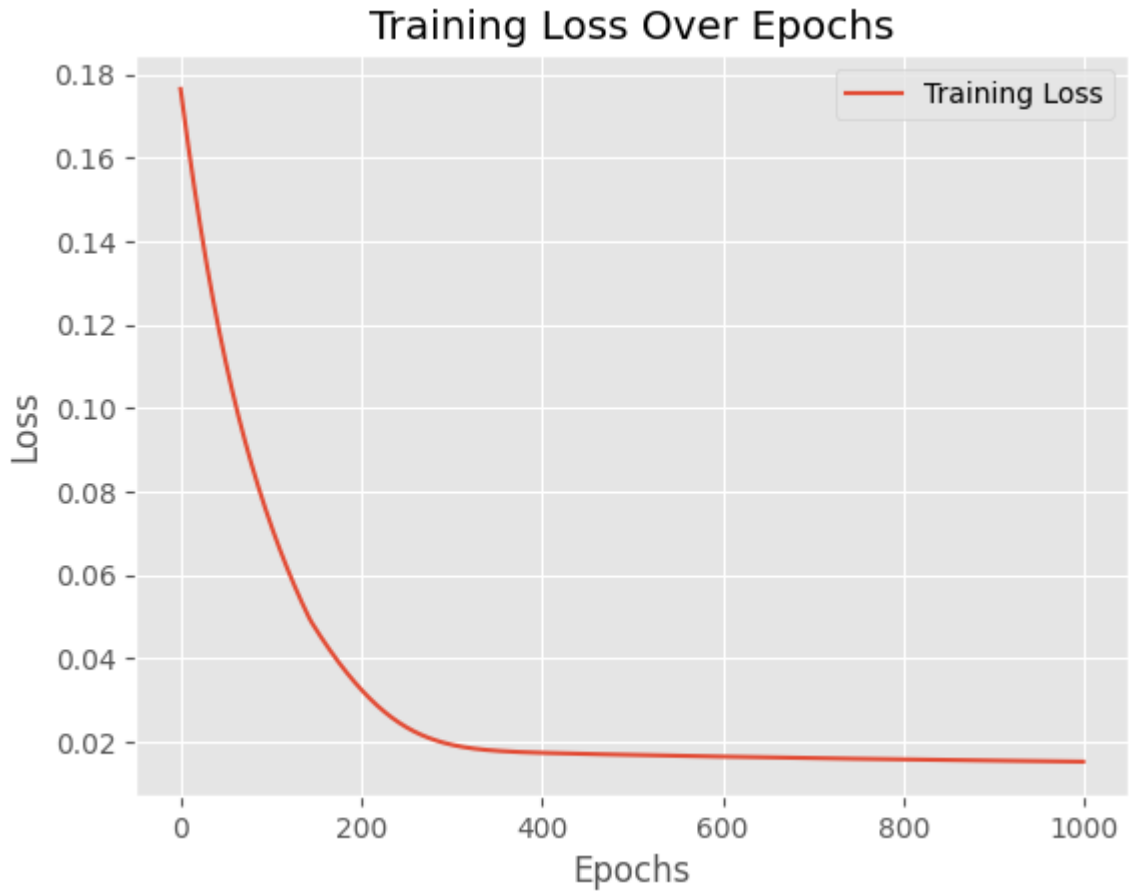
Convertiamo le colonne in tensori PyTorch

```
X_train_pm25_tensor_lb_1 = torch.tensor(X_train_pm25_lb1, dtype=torch.float32)
y_train_pm25_tensor_lb_1 = torch.tensor(y_train_pm25_lb1, dtype=torch.float32)
X_test_pm25_tensor_lb_1 = torch.tensor(X_test_pm25_lb1, dtype=torch.float32)
y_test_pm25_tensor_lb_1 = torch.tensor(y_test_pm25_lb1, dtype=torch.float32)

X_train_pm25_tensor_lb_12 = torch.tensor(X_train_pm25_lb12, dtype=torch.float32)
y_train_pm25_tensor_lb_12 = torch.tensor(y_train_pm25_lb12, dtype=torch.float32)
X_test_pm25_tensor_lb_12 = torch.tensor(X_test_pm25_lb12, dtype=torch.float32)
y_test_pm25_tensor_lb_12 = torch.tensor(y_test_pm25_lb12, dtype=torch.float32)

X_train_pm25_tensor_lb_24 = torch.tensor(X_train_pm25_lb24, dtype=torch.float32)
y_train_pm25_tensor_lb_24 = torch.tensor(y_train_pm25_lb24, dtype=torch.float32)
X_test_pm25_tensor_lb_24 = torch.tensor(X_test_pm25_lb24, dtype=torch.float32)
y_test_pm25_tensor_lb_24 = torch.tensor(y_test_pm25_lb24, dtype=torch.float32)
```

Applichiamo la funzione per il training dei modelli MLP

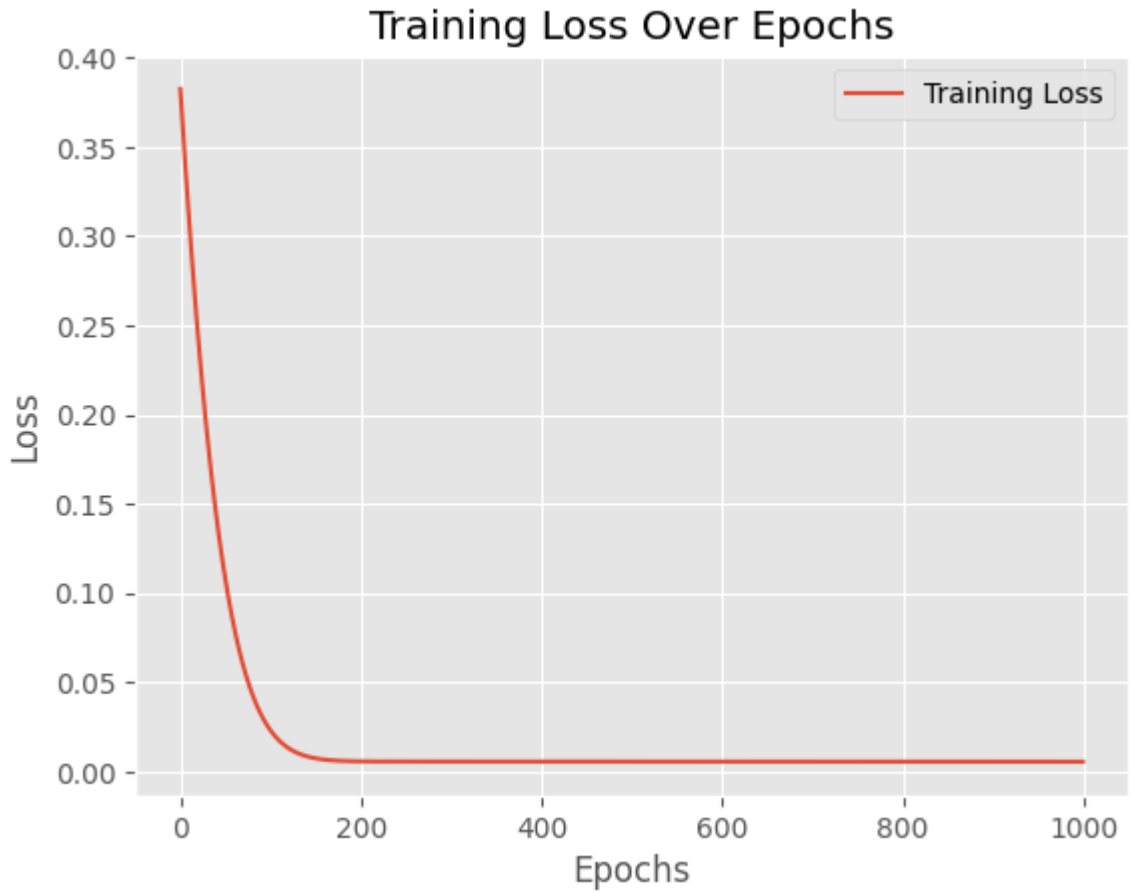
```
#lb=1
y_predicted_MLP_pm25_lb1, y_test_pm25_lb1_rescaled, model_MLP_pm25_lb1, criterion_MLP_pm25_lb1, optimizer_MLP_pm25_lb1,
```

```
Training Progress:  15%|██          | 153/1000 [00:00<00:01, 479.31it/s]Epoch
Training Progress:  20%|██          | 202/1000 [00:00<00:01, 453.49it/s]Epoch
Training Progress:  33%|███         | 333/1000 [00:00<00:01, 354.83it/s]Epoch
Training Progress:  45%|████        | 454/1000 [00:01<00:01, 341.85it/s]Epoch
Training Progress:  52%|████        | 525/1000 [00:01<00:01, 305.21it/s]Epoch
Training Progress:  65%|██████      | 650/1000 [00:01<00:01, 329.79it/s]Epoch
Training Progress:  77%|███████     | 774/1000 [00:02<00:00, 363.68it/s]Epoch
Training Progress:  86%|████████    | 858/1000 [00:02<00:00, 371.55it/s]Epoch
Training Progress:  93%|████████    | 933/1000 [00:02<00:00, 343.55it/s]Epoch
Training Progress: 100%|██████████| 1000/1000 [00:02<00:00, 355.05it/s]
Epoch [1000/1000], Loss: 0.0153
Mean Squared Error (MSE) on test data rescaled: 28.96814727783203
```



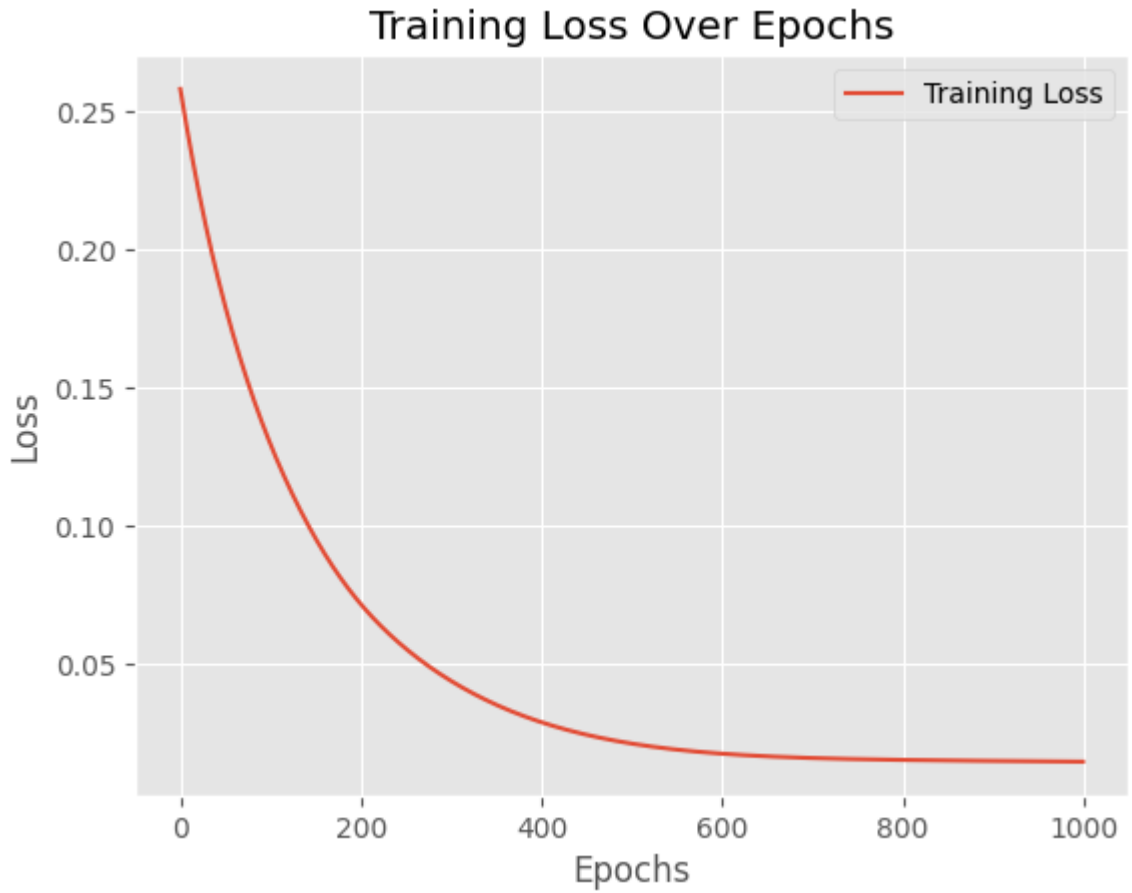
```
#lb=12
y_predicted_MLP_pm25_lb12, y_test_pm25_lb12_rescaled, model_MLP_pm25_lb12, criterion_MLP_pm25_lb12, optimizer_MLP_pm25_l
```

```
Training Progress:  21%|██        | 212/1000 [00:00<00:01, 712.28it/s]Epoch
Epoch [200/1000], Loss: 0.0058
Training Progress:  43%|████      | 434/1000 [00:00<00:00, 727.92it/s]Epoch
Epoch [400/1000], Loss: 0.0057
Training Progress:  58%|██████    | 581/1000 [00:00<00:00, 720.00it/s]Epoch
Epoch [600/1000], Loss: 0.0056
Training Progress:  80%|████████  | 803/1000 [00:01<00:00, 691.25it/s]Epoch
Epoch [800/1000], Loss: 0.0056
Training Progress: 100%|██████████| 1000/1000 [00:01<00:00, 709.67it/s]
Epoch [900/1000], Loss: 0.0056
Epoch [1000/1000], Loss: 0.0056
Mean Squared Error (MSE) on test data rescaled: 9.370349884033203
```



```
#lb=24
y_predicted_MLP_pm25_lb24, y_test_pm25_lb24_rescaled, model_MLP_pm25_lb24, criterion_MLP_pm25_lb24, optimizer_MLP_pm25_l
```

```
Training Progress:   7%|▊         | 73/1000 [00:00<00:01, 723.51it/s]Epoch
Training Progress:  23%|██▎       | 227/1000 [00:00<00:02, 284.20it/s]Epoch
Training Progress:  44%|████▍     | 445/1000 [00:01<00:01, 505.31it/s]Epoch
Epoch [400/1000], Loss: 0.0288
Training Progress:  58%|█████▊    | 581/1000 [00:01<00:00, 592.69it/s]Epoch
Epoch [600/1000], Loss: 0.0173
Training Progress:  74%|███████▍  | 738/1000 [00:02<00:00, 297.84it/s]Epoch
Training Progress:  90%|█████████ | 904/1000 [00:02<00:00, 437.38it/s]Epoch
Epoch [900/1000], Loss: 0.0147
Training Progress: 100%|██████████| 1000/1000 [00:02<00:00, 401.76it/s]
Epoch [1000/1000], Loss: 0.0144
Mean Squared Error (MSE) on test data rescaled: 26.12778663635254
```



```
#lb=1
predicted_mlp_pm25_lb1 = crea_e_stampadf_predicted(y_predicted_MLP_pm25_lb1, y_test_pm25_lb1_rescaled, test_df_pm25_lb_1
print(predicted_mlp_pm25_lb1)
```

```
            misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00       13.2   13.541112
1    2019-01-01 01:00:00       13.2   13.500495
2    2019-01-01 02:00:00       13.2   13.500495
3    2019-01-01 03:00:00       13.2   13.500495
4    2019-01-01 04:00:00       13.2   13.500495
...                  ...        ...         ...
8034 2019-12-31 18:00:00       13.0   13.502188
8035 2019-12-31 19:00:00       13.0   13.502188
8036 2019-12-31 20:00:00       13.0   13.502188
8037 2019-12-31 21:00:00       13.0   13.502188
8038 2019-12-31 22:00:00       13.0   13.502188

[8039 rows x 3 columns]
```

```
#lb=12
predicted_mlp_pm25_lb12 = crea_e_stampadf_predicted(y_predicted_MLP_pm25_lb12, y_test_pm25_lb12_rescaled, test_df_pm25_l
print(predicted_mlp_pm25_lb12)
```

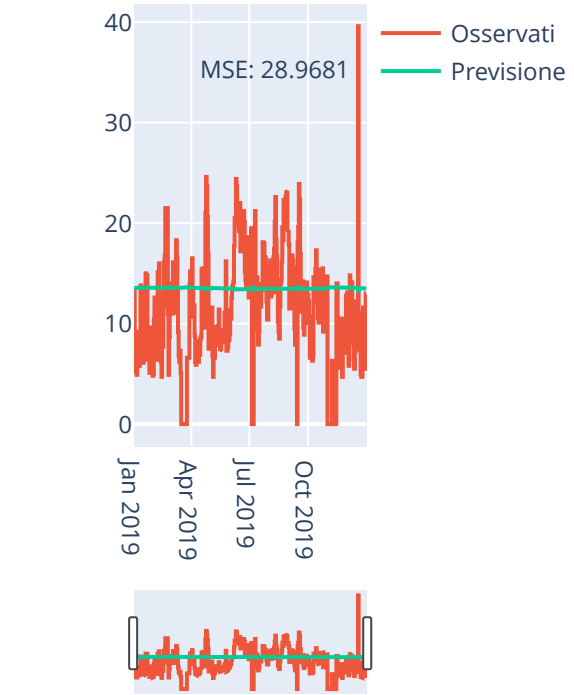
```
            misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00       13.2    9.735905
1    2019-01-01 01:00:00       13.2   13.282193
2    2019-01-01 02:00:00       13.2   13.282193
3    2019-01-01 03:00:00       13.2   13.282193
4    2019-01-01 04:00:00       13.2   13.282193
...                  ...        ...         ...
8023 2019-12-31 07:00:00       13.0   13.134431
8024 2019-12-31 08:00:00       13.0   13.134431
8025 2019-12-31 09:00:00       13.0   13.134431
8026 2019-12-31 10:00:00       13.0   13.134431
8027 2019-12-31 11:00:00       13.0   13.134431

[8028 rows x 3 columns]
```

```
#lb=24
predicted_mlp_pm25_lb24 = crea_e_stampadf_predicted(y_predicted_MLP_pm25_lb24, y_test_pm25_lb24_rescaled, test_df_pm25_l
print(predicted_mlp_pm25_lb24)
```

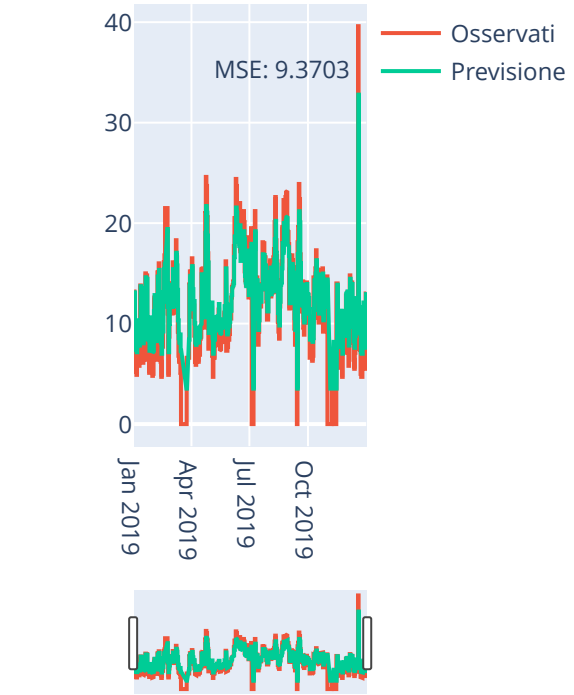
```
            misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00        13.2   12.937775
1    2019-01-01 01:00:00         8.8   13.240960
2    2019-01-01 02:00:00         8.8   13.240960
3    2019-01-01 03:00:00         8.8   13.240960
4    2019-01-01 04:00:00         8.8   13.240960
...                  ...         ...         ...
8011 2019-12-30 19:00:00        13.0   12.874612
8012 2019-12-30 20:00:00        13.0   12.874612
8013 2019-12-30 21:00:00        13.0   12.874612
8014 2019-12-30 22:00:00        13.0   12.874612
8015 2019-12-30 23:00:00        13.0   12.874612

[8016 rows x 3 columns]
```

Visualizziamo i grafici

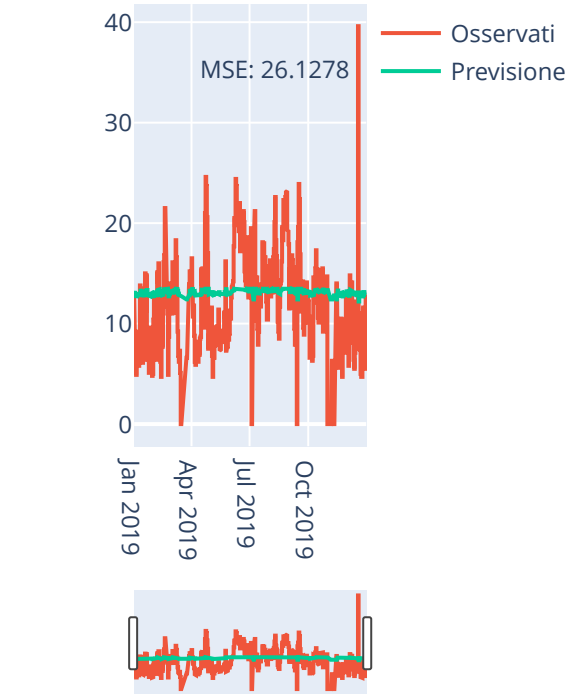
```
grafico_interattivo(predicted_mlp_pm25_lb1, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simbo
```




```
grafico_interattivo(predicted_mlp_pm25_lb12, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simb
```




```
grafico_interattivo(predicted_mlp_pm25_lb24, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simb
```



Scegliamo il modello con l'MSE pari a 9.37

```
print(f'LB=1  - MSE = {mse_mlp_pm25_lb1}')
print(f'LB=12 - MSE = {mse_mlp_pm25_lb12}')
print(f'LB=24 - MSE = {mse_mlp_pm25_lb24}')
```

```
LB=1  - MSE = 28.96814727783203
LB=12 - MSE = 9.370349884033203
LB=24 - MSE = 26.12778663635254
```

```
model_MLP_pm25 = model_MLP_pm25_lb12
criterion_MLP_pm25 = criterion_MLP_pm25_lb12
optimizer_MLP_pm25 = optimizer_MLP_pm25_lb12

X_train_pm25_tensor = X_train_pm25_tensor_lb_12
y_train_pm25_tensor = y_train_pm25_tensor_lb_12
X_test_pm25_tensor = X_test_pm25_tensor_lb_12
y_test_pm25_tensor = y_test_pm25_tensor_lb_12

test_df_pm25 = test_df_pm25_lb_12
test_scaler_pm25 = test_scaler_pm25_lb12
mse_model['mlp_pm25'] = mse_mlp_pm25_lb12
```

```
print(mse_model)
```

```
{'mlp_pm10': 14.768537521362305, 'mlp_pm25': 9.370349884033203}
```

3.2.3 Predizione per una settimana del 2020 per PM2.5

inquinante PM 2.5

Addestriamo il modello MLP dell'inquinante PM2.5 - model_MLP_pm25 - con i dati del 2019 per prevedere una settimana del 2020

I dati del 2019 attualmente sono memorizzati nei tensori:

- X_test_pm25_tensor
- y_test_pm25_tensor

Oggetti del modello:

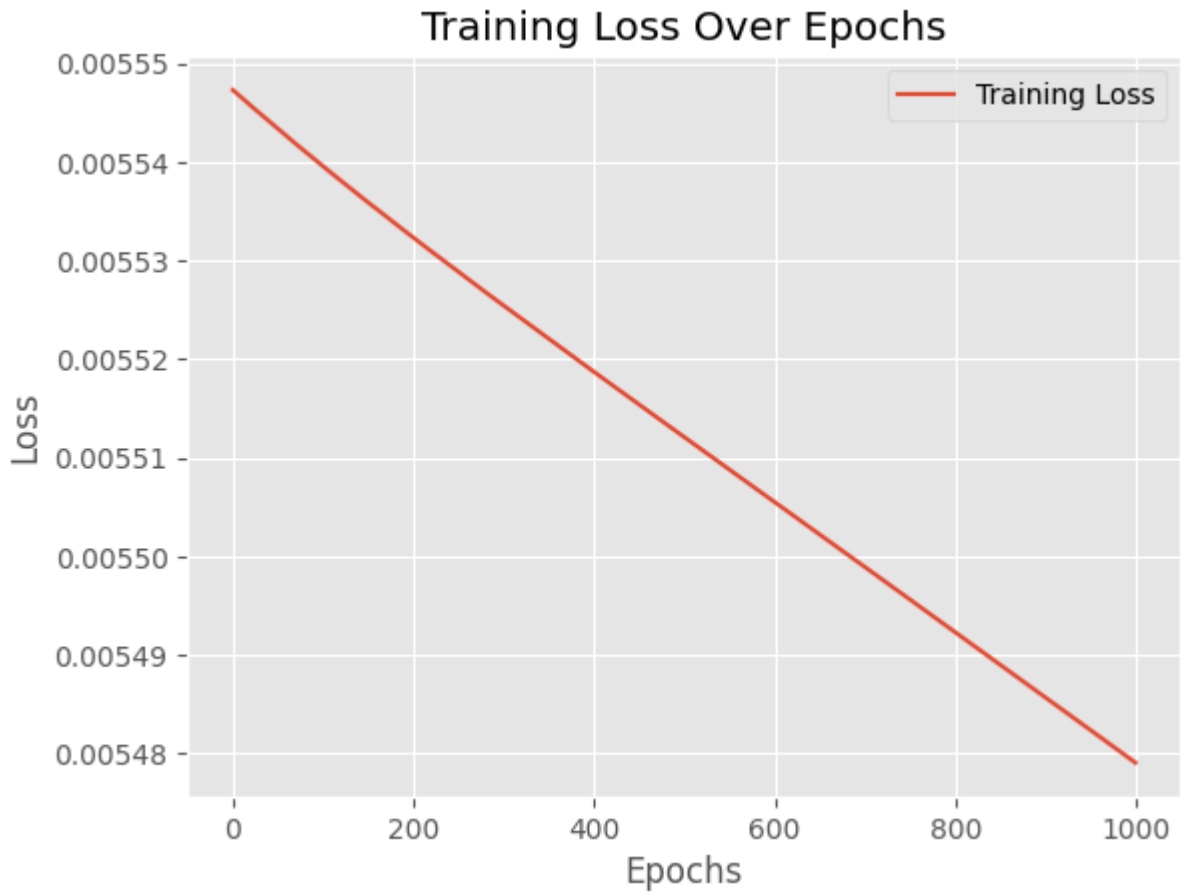
- criterion_MLP_pm25 = Criterio funzione di perdita
- optimizer_MLP_pm25 = Ottimizzatore

```
#memorizziamo i tensori per il training e test
X_2019_pm25 = X_test_pm25_tensor
y_2019_pm25 = y_test_pm25_tensor
```

Split del dataset in training e test, con percentuali 70% e 30%

```
from sklearn.model_selection import train_test_split

# Converti i tensori in array NumPy
X_np = X_2019_pm25.numpy()
y_np = y_2019_pm25.numpy()

# Suddividi i dati in set di addestramento e test
X_train, X_test, y_train, y_test = train_test_split(X_np, y_np, test_size=0.3, random_state=42)

# Converti i dati divisi nuovamente in tensori PyTorch
X_train_tensor_2019_pm25 = torch.from_numpy(X_train)
X_test_tensor_2019_pm25 = torch.from_numpy(X_test)
y_train_tensor_2019_pm25 = torch.from_numpy(y_train)
y_test_tensor_2019_pm25 = torch.from_numpy(y_test)


#Addestriamo il modello
model_MLP_pm25_train2019, y_predicted_mlp_pm25_2020, mse_mlp_train2019_pm25 = new_train_and_evaluate_model(model_MLP_pm2
```

```
Training Progress:  16%|█          | 162/1000 [00:00<00:01, 517.04it/s]Epoch
Epoch [200/1000], Loss: 0.0055
Training Progress:  38%|████       | 376/1000 [00:00<00:01, 518.81it/s]Epoch
Epoch [400/1000], Loss: 0.0055
Training Progress:  65%|███████    | 653/1000 [00:01<00:00, 670.93it/s]Epoch
Epoch [600/1000], Loss: 0.0055
Training Progress:  81%|████████   | 810/1000 [00:01<00:00, 726.57it/s]Epoch
Epoch [800/1000], Loss: 0.0055
Training Progress: 100%|██████████| 1000/1000 [00:01<00:00, 591.70it/s]Epoc
Epoch [1000/1000], Loss: 0.0055
Mean Squared Error (MSE) on test data rescaled: 9.77176284790039
```



```
y_predicted_mlp_pm25_2020, y_2019_pm25 = predict_and_transform(model_MLP_pm25_train2019, X_2019_pm25, scaler)


predicted_mlp_2020_pm25 = crea_e_stampadf_predicted_2020(y_predicted_mlp_pm25_2020, y_2019_pm25, test_df_pm25)


#carichiamo i dati del 2020

collection_name = "aria_2020"
aria2020_df_pm25 = load_and_filter_aria_data(collection_name, inquinante_id_pm25, pm25_id_stazione)
count_null_values(aria2020_df_pm25)
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)

Count del DataFrame: 8207
+-------------------+-------------+
|     misura_dataora|misura_valore|
+-------------------+-------------+
|2020-01-01 01:00:00|  26.60000038|
|2020-01-01 02:00:00|  26.60000038|
|2020-01-01 03:00:00|  26.60000038|
|2020-01-01 04:00:00|  26.60000038|
|2020-01-01 05:00:00|  26.60000038|
+-------------------+-------------+
only showing top 5 rows

Il Dataframe non contiene valori mancanti
```

```
reference_date = datetime.strptime("2020-03-29", "%Y-%m-%d").date()
week_df_pm25 = filter_dataframe_by_week_spark(aria2020_df_pm10, reference_date)
week_df_pm25.printSchema()
print(f"Il DataFrame contiene: {week_df_pm25.count()} rilevazioni")
```

```
root
 |-- misura_dataora: timestamp (nullable = true)
 |-- misura_valore: double (nullable = true)
```

```
    Il DataFrame contiene: 168 rilevazioni
```

```
df_filtered = filter_dataframe_by_week_pandas(predicted_mlp_2020_pm25, reference_date)
print(f"Il DataFrame filtrato contiene: {len(df_filtered)} righe.")
```

```
    Il DataFrame filtrato contiene: 168 righe.
```

```
plot_mlp_pm25_predict_week2020 = merge_spark_and_pandas(week_df_pm25, df_filtered)
```

```
plot_mlp_pm25_predict_week2020.drop('misura_dataora2', axis=1, inplace=True)
```

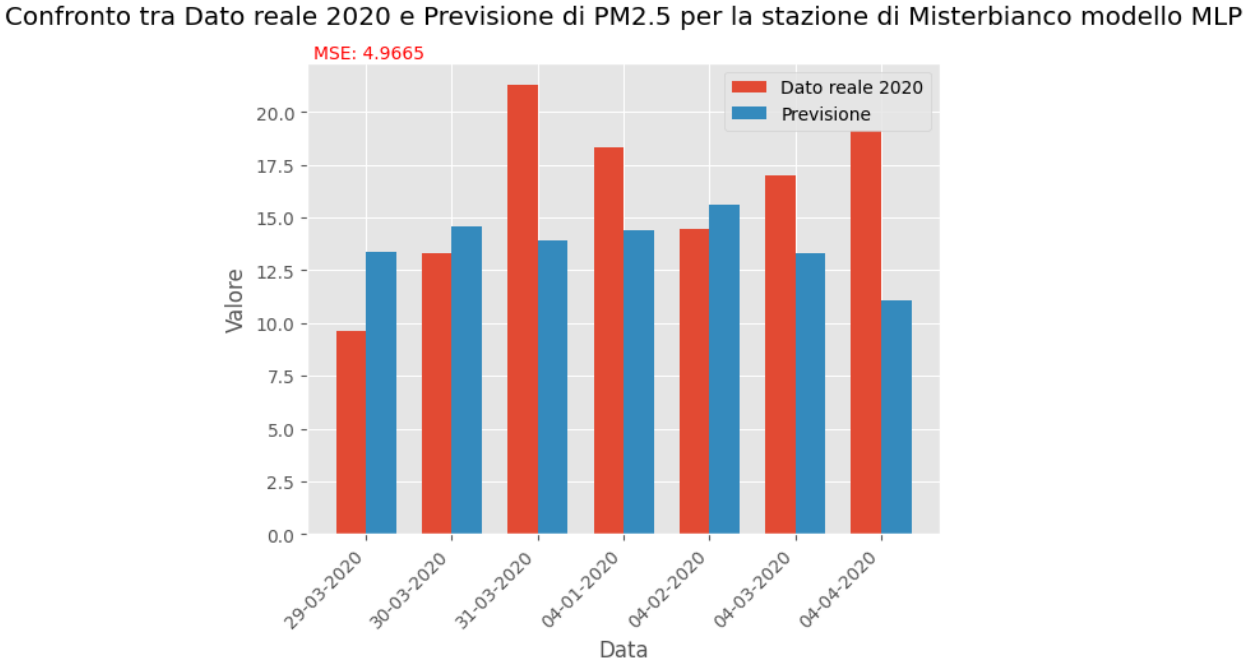
```
plot_mlp_pm25_predict_week2020_groupby = group_by_date_and_average(plot_mlp_pm25_predict_week2020)
```

```
print(plot_mlp_pm25_predict_week2020_groupby)
```

```
           Data  Dato reale 2020  Previsione
    0  2020-03-29         9.600000   13.340890
    1  2020-03-30        13.337500   14.593754
    2  2020-03-31        21.262500   13.940381
    3  2020-04-01        18.341667   14.374843
    4  2020-04-02        14.462500   15.634440
    5  2020-04-03        16.983334   13.287005
    6  2020-04-04        19.112501   11.091126
```

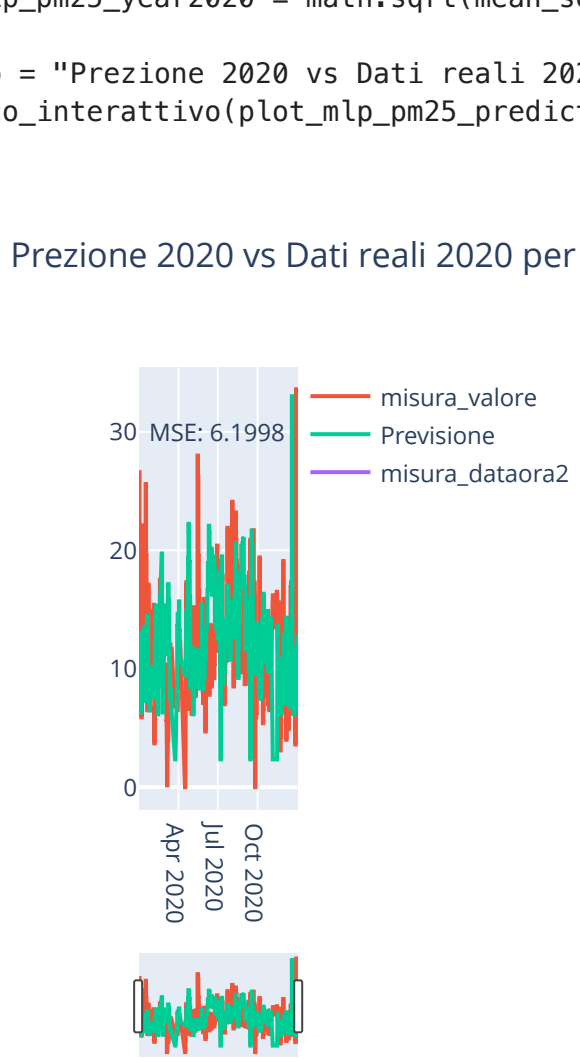
```
from sklearn.metrics import mean_squared_error
import math

mse_mlp_pm25_week2020 = math.sqrt(mean_squared_error(plot_mlp_pm25_predict_week2020['misura_valore'], plot_mlp_pm25_pred
```

```
plot_histogram(plot_mlp_pm25_predict_week2020_groupby, simbolo_pm25, pm25_nome_stazione, 'MLP', mse_mlp_pm25_week2020)
```




```
plot_mlp_pm25_predict2020_year = merge_spark_and_pandas(aria2020_df_pm25, predicted_mlp_2020_pm25)
```

```
from sklearn.metrics import mean_squared_error
import math

mse_mlp_pm25_year2020 = math.sqrt(mean_squared_error(plot_mlp_pm25_predict2020_year['misura_valore'], plot_mlp_pm25_pred

titolo = "Prezione 2020 vs Dati reali 2020 per inquinante " + simbolo_pm25 + " presso la stazione di " + pm25_nome_stazi
grafico_interattivo(plot_mlp_pm25_predict2020_year, titolo, mse_mlp_pm25_year2020)
```



3.2.4 Si riesegua il compito ai punti 3.2.2 e 3.2.3 usando un modello LSTM

Preparazione dataframe di training e di test per l'inquinante PM10

Rispetto a MLP eseguiamo uno scaling dei dati tra -1 e 1

```
#Processiamo nuovamente i dati per ricavare i tensori PyTorch

#Scegliamo di impostare la finestra look_back = 1

train_df_pm10 = df_in_Pandas(aria2018_df_pm10, 1)
test_df_pm10 = df_in_Pandas(aria2019_df_pm10, 1)

train_df_pm10 = df_in_Pandas(aria2018_df_pm10, 1)
test_df_pm10 = df_in_Pandas(aria2019_df_pm10, 1)

scaler_lstm = MinMaxScaler(feature_range = (-1, 1))

train_scaler_pm10 = scale_dataframe(train_df_pm10, scaler_lstm)
test_scaler_pm10 = scale_dataframe(test_df_pm10, scaler_lstm)

X_train_pm10, y_train_pm10, X_test_pm10, y_test_pm10 = prepare_dataframes(train_scaler_pm10, test_scaler_pm10)

X_train_pm10_tensor = torch.tensor(X_train_pm10, dtype=torch.float32)
y_train_pm10_tensor = torch.tensor(y_train_pm10, dtype=torch.float32)
X_test_pm10_tensor = torch.tensor(X_test_pm10, dtype=torch.float32)
y_test_pm10_tensor = torch.tensor(y_test_pm10, dtype=torch.float32)
```

```
(8423, 1)
(8423, 1)
(8207, 1)
(8207, 1)
```

```
#Eseguiamo la trasformazione della forma dei tensori per renderli compatibili con la rete neurale LSTM

X_train_pm10_tensor = torch.reshape(X_train_pm10_tensor, (X_train_pm10_tensor.shape[0], X_train_pm10_tensor.shape[1], 1)
X_test_pm10_tensor = torch.reshape(X_test_pm10_tensor, (X_test_pm10_tensor.shape[0], X_test_pm10_tensor.shape[1], 1))
```

Creiamo la classe per generare il modello LSTM con PyTorch

```
class modelLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_dim):
        super(modelLSTM, self).__init__()
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_dim)

    def forward(self, x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)).requires_grad_()
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)).requires_grad_()
        # Propagate input through LSTM
        out, (h_0, c_0) = self.lstm(x, (h_0.detach(), c_0.detach()))
        out = self.fc(out[:, -1, :])  # Final Output
        return out
```

Impostiamo i parametri del modello

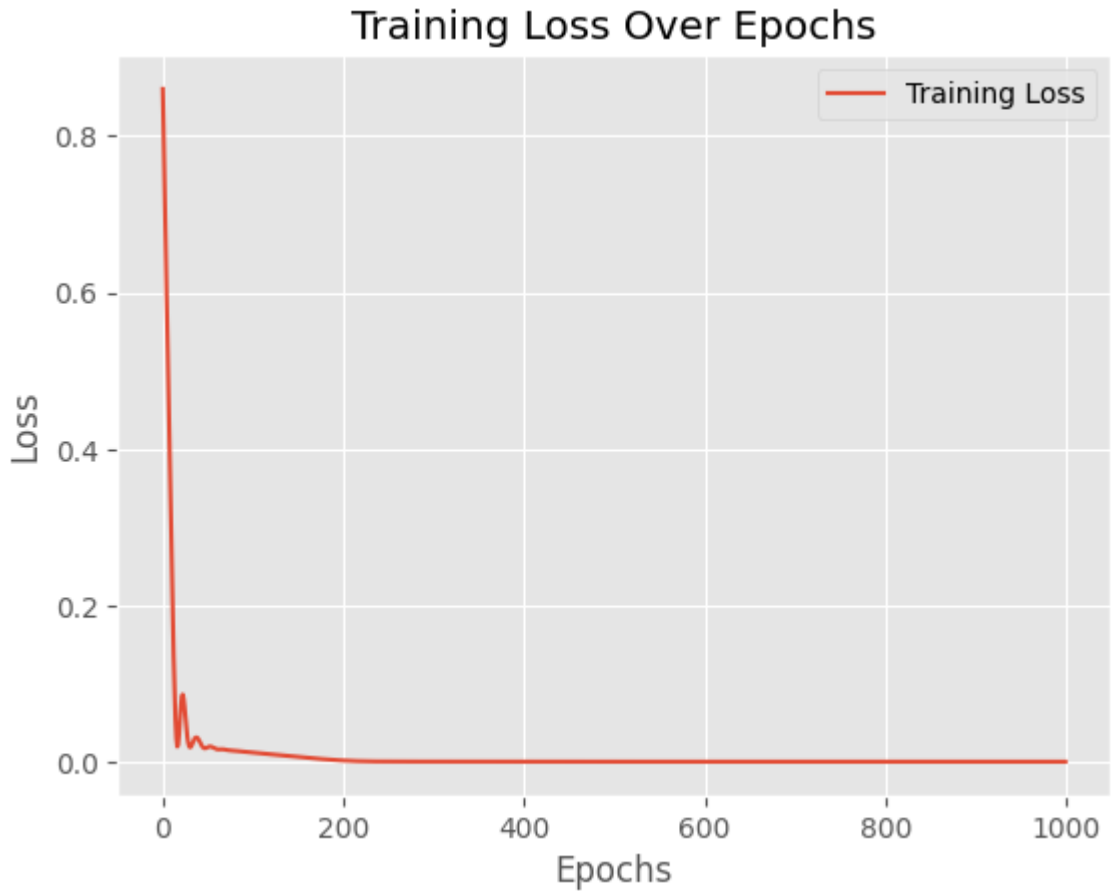
```
num_epochs = 1000 #1000 epoche
learning_rate = 0.01 #capacità di apprendimento
input_size = 1 #numbeo di features
hidden_size = 32 #strati nascosti
num_layers = 1 #numero di livelli lstm
```

Creiamo il modello e gli oggetti criterion e optimizer Impostiamo il criterio MSE e Ottimizzatore Adam

```
model_LSTM_pm10 = modelLSTM(input_size, hidden_size, num_layers, 1)
criterion_LSTM_pm10 = torch.nn.MSELoss(reduction='mean')    # mean-squared error
optimizer_LSTM_pm10 = torch.optim.Adam(model_LSTM_pm10.parameters(), lr=learning_rate)


lstm = []

from tqdm import tqdm

# Ciclo di addestramento con barra di avanzamento
for epoch in tqdm(range(num_epochs), desc="Training Progress"):
    y_train_pred = model_LSTM_pm10(X_train_pm10_tensor)
    loss = criterion_LSTM_pm10(y_train_pred, y_train_pm10_tensor)  # Calcolo della loss
    lstm.append(loss.item())
    optimizer_LSTM_pm10.zero_grad()  # Reset dei gradienti
    loss.backward()  # Backpropagation
    optimizer_LSTM_pm10.step()  # Ottimizzazione

    # Aggiungi il valore della loss alla lista

    if epoch % 100 == 0:
        tqdm.write(f"Epoch: {epoch}, Loss: {loss.item():.5f}")
```

```
Training Progress:   1%|          | 6/1000 [00:00<00:38, 25.63it/s]Epoch: 0, Loss: 0.86015
Training Progress:  10%|▉         | 105/1000 [00:03<00:32, 27.20it/s]Epoch: 100, Loss: 0.01264
Training Progress:  20%|██        | 204/1000 [00:07<00:37, 21.16it/s]Epoch: 200, Loss: 0.00251
Training Progress:  31%|███       | 306/1000 [00:12<00:26, 26.53it/s]Epoch: 300, Loss: 0.00116
Training Progress:  40%|████      | 405/1000 [00:16<00:21, 27.75it/s]Epoch: 400, Loss: 0.00108
Training Progress:  50%|█████     | 504/1000 [00:19<00:18, 26.34it/s]Epoch: 500, Loss: 0.00106
Training Progress:  60%|██████    | 603/1000 [00:23<00:18, 21.37it/s]Epoch: 600, Loss: 0.00106
Training Progress:  70%|███████   | 705/1000 [00:28<00:11, 26.25it/s]Epoch: 700, Loss: 0.00106
Training Progress:  80%|████████  | 804/1000 [00:31<00:07, 27.19it/s]Epoch: 800, Loss: 0.00106
Training Progress:  91%|█████████ | 906/1000 [00:35<00:03, 27.20it/s]Epoch: 900, Loss: 0.00106
Training Progress: 100%|██████████| 1000/1000 [00:39<00:00, 25.33it/s]
```

```
# Stampare il grafico della loss
plt.plot(lstm, label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss Over Epochs')
plt.legend()
plt.show()
```




```
# Eseguiamo la predizione
with torch.no_grad():
    y_predicted_lstm_pm10 = model_LSTM_pm10(X_test_pm10_tensor)
```

```
#La funzione esegue il rescaling della predizione restituendo un dataframe

def rescaler_predicted(y_predicted, df, df_scaler, scaler):
    # Converte il tensore PyTorch in un array NumPy
    y_predicted = np.array(y_predicted)
    predicted_model = [i[0] for i in y_predicted]
    misura_valore = df_scaler.iloc[:, 0].values
    df_predicted = pd.DataFrame({
        'Osservati': misura_valore,
        'Previsione': predicted_model
    })

    # Inverti la trasformazione utilizzando lo scaler
    df_predicted = pd.DataFrame(scaler.inverse_transform(df_predicted), columns=['Osservati', 'Previsione'])

    # Creare un DataFrame con la colonna "misura_dataora" da df_scaler
    df_misura_dataora = df[['misura_dataora']].copy()

    # Creare un nuovo DataFrame che includa la colonna "misura_dataora" e le altre colonne da df_predicted
    df_predicted_final = pd.concat([df_misura_dataora, df_predicted], axis=1)

    # Restituisci il DataFrame risultante
    return df_predicted_final



predicted_lstm_pm10 = rescaler_predicted(y_predicted_lstm_pm10, test_df_pm10, test_scaler_pm10, scaler_lstm)
print(predicted_lstm_pm10)
```

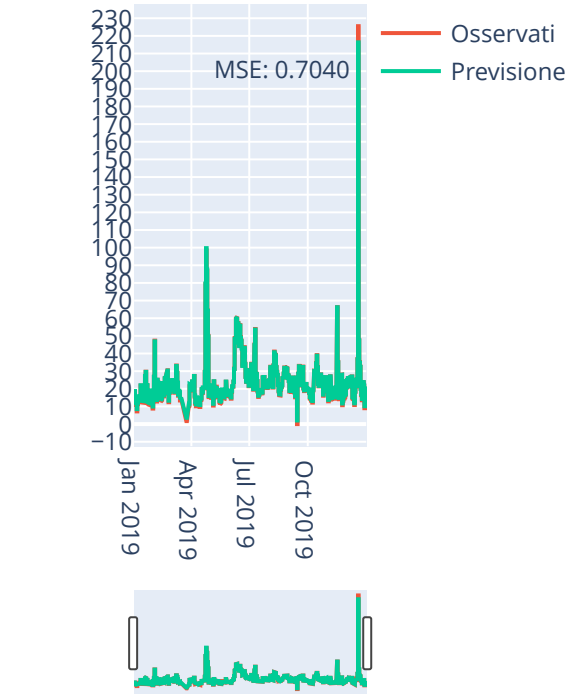
```
           misura_dataora  Osservati  Previsione
0    2019-01-01 00:00:00  11.500000   12.420573
1    2019-01-01 01:00:00  19.400000   19.757501
2    2019-01-01 02:00:00  19.400000   19.757501
3    2019-01-01 03:00:00  19.400000   19.757501
4    2019-01-01 04:00:00  19.400000   19.757501
...                  ...        ...         ...
8202 2019-12-31 18:00:00  21.700001   21.924327
8203 2019-12-31 19:00:00  21.700001   21.924327
```

```
      8204 2019-12-31 20:00:00  21.700001   21.924327
      8205 2019-12-31 21:00:00  21.700001   21.924327
      8206 2019-12-31 22:00:00  21.700001   21.924327

      [8207 rows x 3 columns]
```

```
#Calcoliamo MSE per validare il modello
from sklearn.metrics import mean_squared_error
import math

mse_lstm_pm10 = math.sqrt(mean_squared_error(predicted_lstm_pm10['Osservati'], predicted_lstm_pm10['Previsione']))
print(f'Mean Squared Error (MSE) on test data rescaled: {mse_lstm_pm10}')
mse_model['lstm_pm10'] = mse_lstm_pm10
```

```
    Mean Squared Error (MSE) on test data rescaled: 0.7040250858015171
```

```
grafico_interattivo(predicted_lstm_pm10, "Valori rilevati nel 2019 vs Valori predetti nel 2019 di inquinante "+ simbolo_
```



Predizione per una settimana del 2020 per PM10 con modello LSTM

inquinante PM 10

Addestriamo il modello LSTM dell'inquinante PM10 - model_LSTM_pm10 - con i dati del 2019 per prevedere una settimana del 2020

I dati del 2019 attualmente sono memorizzati nei tensori:

- X_test_pm10_tensor = torch.tensor(X_test_pm10, dtype=torch.float32)
- y_test_pm10_tensor = torch.tensor(y_test_pm10, dtype=torch.float32)

Oggetti del modello:

- criterion_lstm_pm10 = Criterio funzione di perdita
- optimizer_lstm_pm10 = Ottimizzatore

```
X_2019_pm10 = X_test_pm10_tensor
y_2019_pm10 = y_test_pm10_tensor


from sklearn.model_selection import train_test_split

# Specifica le colonne che costituiranno le feature (X) e il target (y)
# Sostituisci queste con le colonne effettive del tuo DataFrame
feature_cols = ['misura_valore']
target_col = 'target'

# Suddividi il DataFrame in feature e target
X = test_scaler_pm10[feature_cols]
y = test_scaler_pm10[target_col]

# Utilizza train_test_split per suddividere i dati
X_train_test_scaler_pm10, X_test_test_scaler_pm10, y_train_test_scaler_pm10, y_test_test_scaler_pm10 = train_test_split(
test_data = pd.concat([X_test_test_scaler_pm10, y_test_test_scaler_pm10], axis=1)
```

```
from sklearn.model_selection import train_test_split

# Converti i tensori in array NumPy
X_np = X_2019_pm10.numpy()
y_np = y_2019_pm10.numpy()

# Suddividi i dati in set di addestramento e test
X_train, X_test, y_train, y_test = train_test_split(X_np, y_np, test_size=0.3, random_state=42)

# Converti i dati divisi nuovamente in tensori PyTorch
X_train_tensor_2019_pm10 = torch.from_numpy(X_train)
X_test_tensor_2019_pm10 = torch.from_numpy(X_test)
y_train_tensor_2019_pm10 = torch.from_numpy(y_train)
y_test_tensor_2019_pm10 = torch.from_numpy(y_test)


lstm = []

from tqdm import tqdm

# Ciclo di addestramento con barra di avanzamento
for epoch in tqdm(range(num_epochs), desc="Training Progress"):
    y_train_pred = model_LSTM_pm10(X_train_tensor_2019_pm10)
    loss = criterion_LSTM_pm10(y_train_pred, y_train_tensor_2019_pm10)  # Calcolo della loss
    lstm.append(loss.item())
    optimizer_LSTM_pm10.zero_grad()  # Reset dei gradienti
    loss.backward()  # Backpropagation
    optimizer_LSTM_pm10.step()  # Ottimizzazione

    # Aggiungi il valore della loss alla lista

    if epoch % 100 == 0:
        tqdm.write(f"Epoch: {epoch}, Loss: {loss.item():.5f}")

 Training Progress:   0%|          | 4/1000 [00:00<00:30, 32.67it/s]Epoch: 0, Loss: 0.00110
 Training Progress:  11%|▉         | 110/1000 [00:02<00:18, 47.72it/s]Epoch: 100, Loss: 0.00108
 Training Progress:  21%|██        | 207/1000 [00:04<00:16, 46.97it/s]Epoch: 200, Loss: 0.00108
 Training Progress:  31%|███       | 308/1000 [00:06<00:14, 48.36it/s]Epoch: 300, Loss: 0.00108
```

```
Training Progress:  41%|████      | 408/1000 [00:08<00:12, 47.67it/s]Epoch: 400, Loss: 0.00108
Training Progress:  51%|█████     | 507/1000 [00:10<00:10, 47.28it/s]Epoch: 500, Loss: 0.00108
Training Progress:  60%|██████    | 605/1000 [00:13<00:10, 37.64it/s]Epoch: 600, Loss: 0.00108
Training Progress:  71%|███████   | 707/1000 [00:15<00:07, 36.91it/s]Epoch: 700, Loss: 0.00108
```