Importing libraries

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in
./env/lib/python3.10/site-packages (2.13.0rc1)
Requirement already satisfied: tensorflow-macos==2.13.0-rc1 in
./env/lib/python3.10/site-packages (from tensorflow) (2.13.0rc1)
Requirement already satisfied: termcolor>=1.1.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (2.3.0)
Requirement already satisfied: packaging in ./env/lib/python3.10/site-
packages (from tensorflow-macos==2.13.0-rc1->tensorflow) (22.0)
Requirement already satisfied: setuptools in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (65.6.3)
Requirement already satisfied: six>=1.12.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (1.16.0)
Requirement already satisfied: flatbuffers>=23.1.21 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (23.5.26)
Requirement already satisfied: h5py>=2.9.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (3.8.0)
Requirement already satisfied: wrapt>=1.11.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (1.15.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (0.4.0)
Requirement already satisfied: numpy>=1.22 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (1.23.5)
Requirement already satisfied: typing-extensions>=3.6.6 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (4.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (4.23.3)
Requirement already satisfied: opt-einsum>=2.3.2 in
```

```
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (3.3.0)
Requirement already satisfied: tensorflow-estimator<2.14,>=2.13.0rc0
in ./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-
rc1->tensorflow) (2.13.0rc0)
Requirement already satisfied: tensorboard<2.14,>=2.13 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (2.13.0)
Requirement already satisfied: google-pasta>=0.1.1 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (16.0.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (1.54.2)
Requirement already satisfied: keras<2.14,>=2.13.1rc0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (2.13.1rc0)
Requirement already satisfied: astunparse>=1.6.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (1.6.3)
Requirement already satisfied: absl-py>=1.0.0 in
./env/lib/python3.10/site-packages (from tensorflow-macos==2.13.0-rc1-
>tensorflow) (1.4.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
./env/lib/python3.10/site-packages (from astunparse>=1.6.0-
>tensorflow-macos==2.13.0-rc1->tensorflow) (0.38.4)
Requirement already satisfied: markdown>=2.6.8 in
./env/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow-macos==2.13.0-rc1->tensorflow) (3.4.3)
Requirement already satisfied: requests<3,>=2.21.0 in
./env/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow-macos==2.13.0-rc1->tensorflow) (2.28.1)
Requirement already satisfied: werkzeug>=1.0.1 in
./env/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow-macos==2.13.0-rc1->tensorflow) (2.3.6)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in ./env/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow-macos==2.13.0-rc1->tensorflow) (0.7.1)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
./env/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow-macos==2.13.0-rc1->tensorflow) (1.0.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
./env/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow-macos==2.13.0-rc1->tensorflow) (2.20.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
./env/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
```

```
(5.3.1)
Requirement already satisfied: urllib3<2.0 in
./env/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(1.26.14)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
./env/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
./env/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
./env/lib/python3.10/site-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-
rc1->tensorflow) (1.3.1)
Requirement already satisfied: certifi>=2017.4.17 in
./env/lib/python3.10/site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(2022.6.15)
Requirement already satisfied: charset-normalizer<3,>=2 in
./env/lib/python3.10/site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
./env/lib/python3.10/site-packages (from requests<3,>=2.21.0-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(3.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in
./env/lib/python3.10/site-packages (from werkzeug>=1.0.1-
>tensorboard<2.14,>=2.13->tensorflow-macos==2.13.0-rc1->tensorflow)
(2.1.1)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
./env/lib/python3.10/site-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow-
macos==2.13.0-rc1->tensorflow) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in
./env/lib/python3.10/site-packages (from requests-oauthlib>=0.7.0-
>google-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow-
macos==2.13.0-rc1->tensorflow) (3.2.2)
```

```python
import pandas as pd
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.neural_network import MLPRegressor
import tensorflow as tf
import time
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

importing dataset

```
df = pd.read_excel("Dataset.xlsx", names=[ "Time_Interval", "NO",
"NO2", "SO2", "Air_temp", "Rel_Humidity", "Wind_Dir", "Wind_speed",
"Lag1", "Lag2", "PM"],header=None);

df

             Time_Interval    NO    NO2  SO2  Air_temp
Rel_Humidity  \
0      2020-01-01 00:00:00  5.25  14.80  NaN      18.0          76.00

1      2020-01-01 01:00:00  4.90  15.05  NaN      18.0          77.60

2      2020-01-01 02:00:00  2.35  14.10  0.0      18.0          79.10

3      2020-01-01 03:00:00  1.55  16.00  0.0      18.0          80.20

4      2020-01-01 04:00:00  1.60  16.75  0.0      18.0          80.35

...                    ...   ...    ...  ...       ...            ...

26300  2022-12-31 20:00:00  0.50   2.10  0.2       NaN            NaN

26301  2022-12-31 21:00:00  0.50   1.35  0.2       NaN            NaN

26302  2022-12-31 22:00:00  0.50   1.20  0.4       NaN            NaN

26303                  NaN   NaN    NaN  NaN       NaN            NaN

26304                  NaN   NaN    NaN  NaN       NaN            NaN


       Wind_Dir  Wind_speed   Lag1   Lag2     PM
0         223.5        2.20    NaN    NaN  10.60
1         217.0        2.10  10.60    NaN   8.60
2         212.0        1.75   8.60  10.60   7.85
3         188.0        1.20   7.85   8.60   8.60
4         172.5        1.05   8.60   7.85   9.30
```

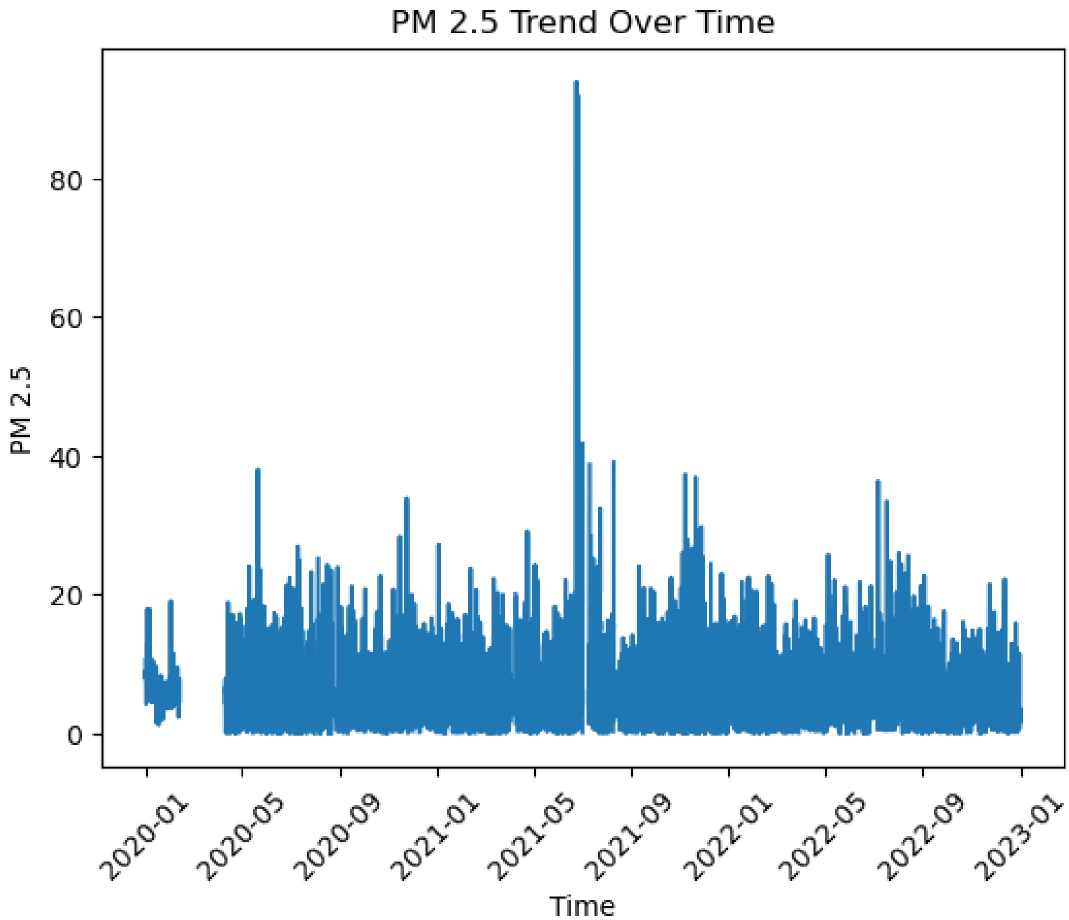
```
...        ...        ...    ...    ...    ...
26300      NaN        NaN  -0.25  -0.75   0.90
26301      NaN        NaN   0.90  -0.25   2.00
26302      NaN        NaN   2.00   0.90   1.95
26303      NaN        NaN   1.95   2.00    NaN
26304      NaN        NaN    NaN   1.95    NaN

[26305 rows x 11 columns]
```

```
df['Time_Interval'] = pd.to_datetime(df['Time_Interval'])  # Convert the timestamp column to datetime format
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26305 entries, 0 to 26304
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Time_Interval  26303 non-null  datetime64[ns]
 1   NO             22790 non-null  float64
 2   NO2            22948 non-null  float64
 3   SO2            23766 non-null  float64
 4   Air_temp       24776 non-null  float64
 5   Rel_Humidity   24912 non-null  float64
 6   Wind_Dir       23653 non-null  float64
 7   Wind_speed     24727 non-null  float64
 8   Lag1           22798 non-null  float64
 9   Lag2           22798 non-null  float64
 10  PM             22798 non-null  float64
dtypes: datetime64[ns](1), float64(10)
memory usage: 2.2 MB
```

```
plt.plot(df["Time_Interval"], df["PM"])
plt.xlabel("Time")
plt.ylabel("PM 2.5")
plt.title("PM 2.5 Trend Over Time")
plt.xticks(rotation=45)
plt.show()
```




```
# Identify and handle missing data
missing_data = df.isnull().sum()
missing_data
```

```
Time_Interval       2
NO               3515
NO2              3357
SO2              2539
Air_temp         1529
Rel_Humidity     1393
Wind_Dir         2652
Wind_speed       1578
Lag1             3507
Lag2             3507
PM               3507
dtype: int64
```

```
# Replace negative values with NaN in PM2.5 column as mentioned by
SARA zandi in her presentation
df['PM'] = np.where(df['PM'] < 0, np.nan, df['PM'])
```

```
missing_data = df.isnull().sum()
missing_data
```

```
Time_Interval       2
NO               3515
NO2              3357
SO2              2539
Air_temp         1529
Rel_Humidity     1393
Wind_Dir         2652
Wind_speed       1578
Lag1             3507
Lag2             3507
PM               4722
dtype: int64
```

```
# Replace missing values with the mean, median and mode imputation
mean_df = df.fillna(df.mean(), inplace=False)
median_df=df.fillna(df.median(), inplace=False)
mode_df= df.fillna(df.mode().iloc[0])
```

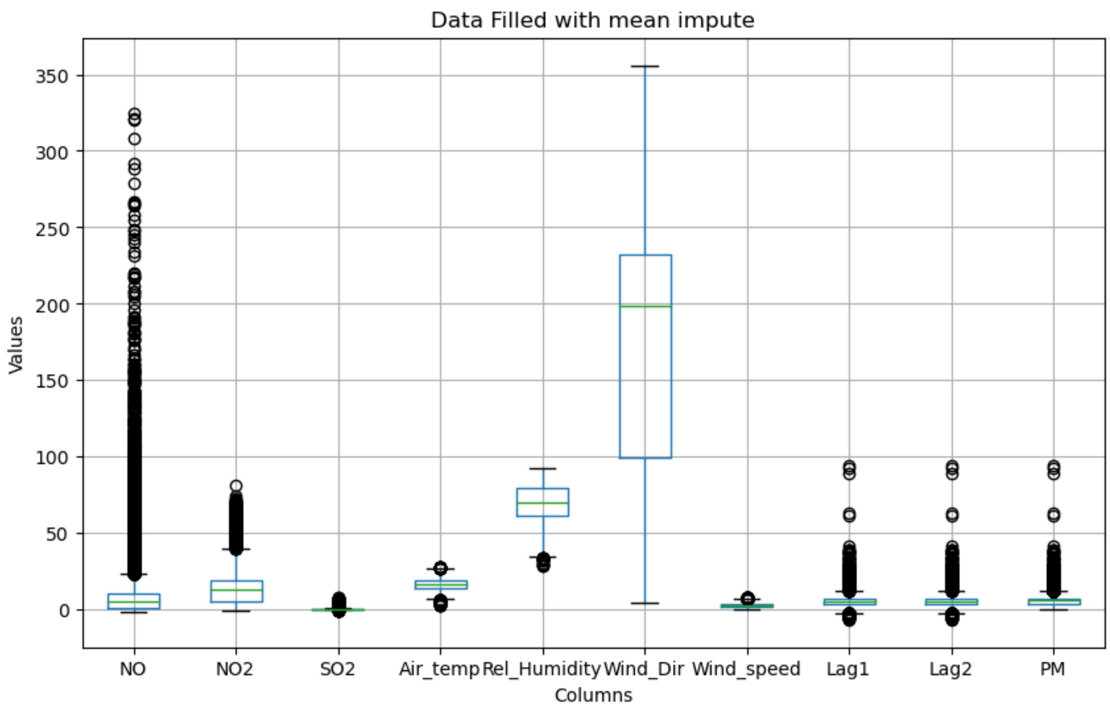
```
/var/folders/r9/ftlfg09n2rbgf2q9jy3b7j7h0000gn/T/
ipykernel_21959/2800757332.py:2: FutureWarning: DataFrame.mean and
DataFrame.median with numeric_only=None will include datetime64 and
datetime64tz columns in a future version.
  mean_df = df.fillna(df.mean(), inplace=False)
/var/folders/r9/ftlfg09n2rbgf2q9jy3b7j7h0000gn/T/ipykernel_21959/28007
57332.py:3: FutureWarning: DataFrame.mean and DataFrame.median with
numeric_only=None will include datetime64 and datetime64tz columns in
a future version.
  median_df=df.fillna(df.median(), inplace=False)
```

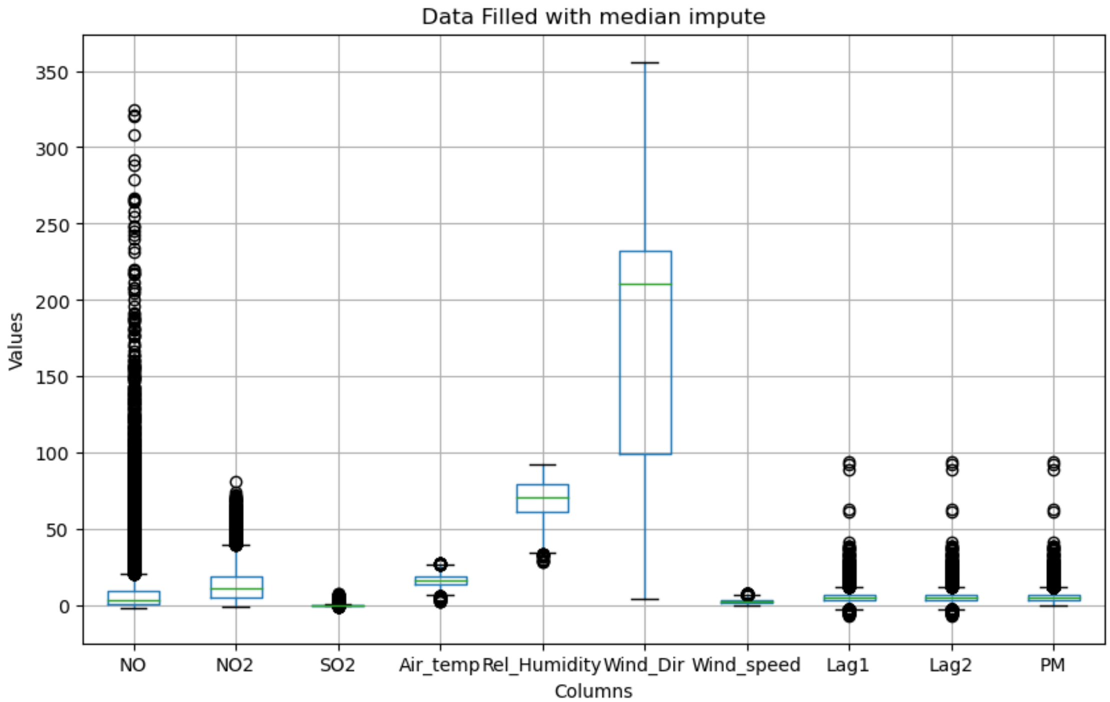
```
### ploting boxplot for all three imputation method to compare and see
which works best
```

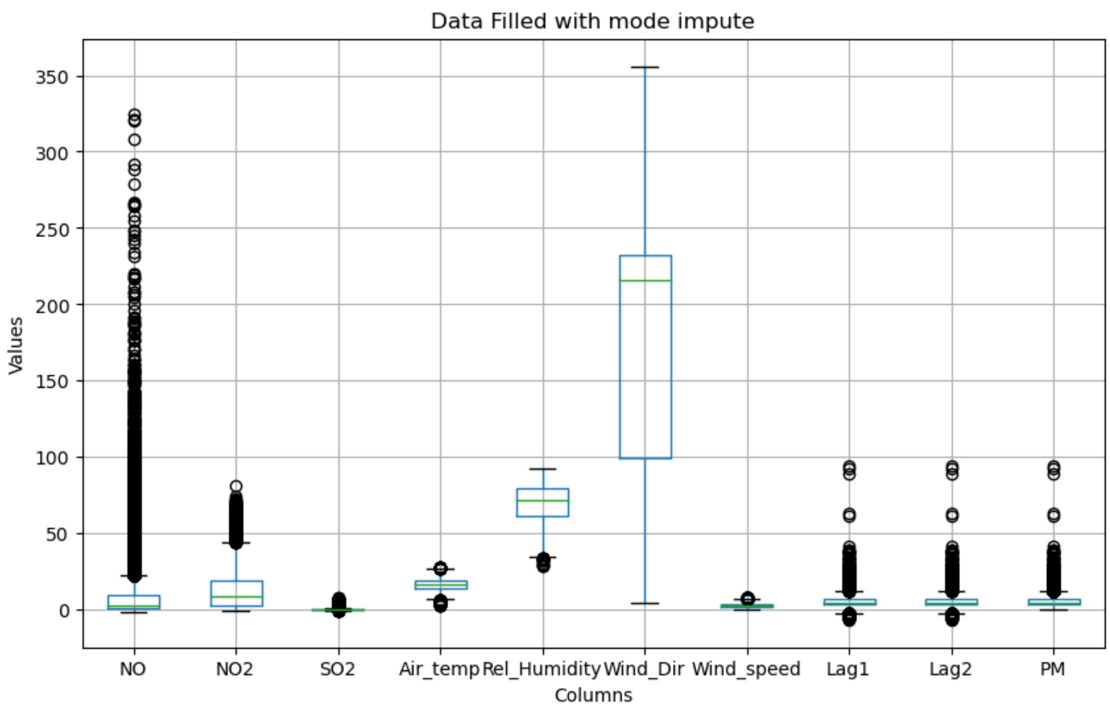
```
#ploting mean impute
fig, ax = plt.subplots(figsize=(10, 6))
mean_df.boxplot(ax=ax)
plt.title("Data Filled with mean impute")
# # Set the labels for the x-axis and y-axis
plt.xlabel("Columns")
plt.ylabel("Values")
# # Display the plot
plt.show()
```

```
#ploting median impute
fig, ax = plt.subplots(figsize=(10, 6))
median_df.boxplot(ax=ax)
plt.title("Data Filled with median impute")
# # Set the labels for the x-axis and y-axis
plt.xlabel("Columns")
```

```
plt.ylabel("Values")
# # Display the plot
plt.show()


#ploting mode impute
fig, ax = plt.subplots(figsize=(10, 6))
mode_df.boxplot(ax=ax)
plt.title("Data Filled with mode impute")
# # Set the labels for the x-axis and y-axis
plt.xlabel("Columns")
plt.ylabel("Values")
# # Display the plot
plt.show()
```








```
#Since there is not significant difference when comparing we'll go
with mean imputation

# Identify and handle outliers
outliers = mean_df[["NO", "NO2", "SO2", "Air_temp", "Rel_Humidity",
"Wind_Dir", "Wind_speed", "Lag1", "Lag2", "PM"]].apply(lambda x:
```

```
np.abs(x - x.mean()) / x.std() > 3).sum()
outliers
```

```
NO              539
NO2             361
SO2             493
Air_temp         45
Rel_Humidity     14
Wind_Dir          0
Wind_speed       55
Lag1            351
Lag2            351
PM              390
dtype: int64
```

```
# Define the columns with outliers
outlier_columns = [ 'NO', 'NO2', 'SO2', 'Air_temp', 'Rel_Humidity',
'Wind_speed','PM', 'Lag1', 'Lag2']

# Calculate z-scores for outlier detection and removal
z_scores = np.abs(stats.zscore(mean_df[outlier_columns]))
threshold = 3

# Remove rows with outliers
mean_df = mean_df[(z_scores < threshold).all(axis=1)]

# Verify the updated dataset
(mean_df.shape)
```

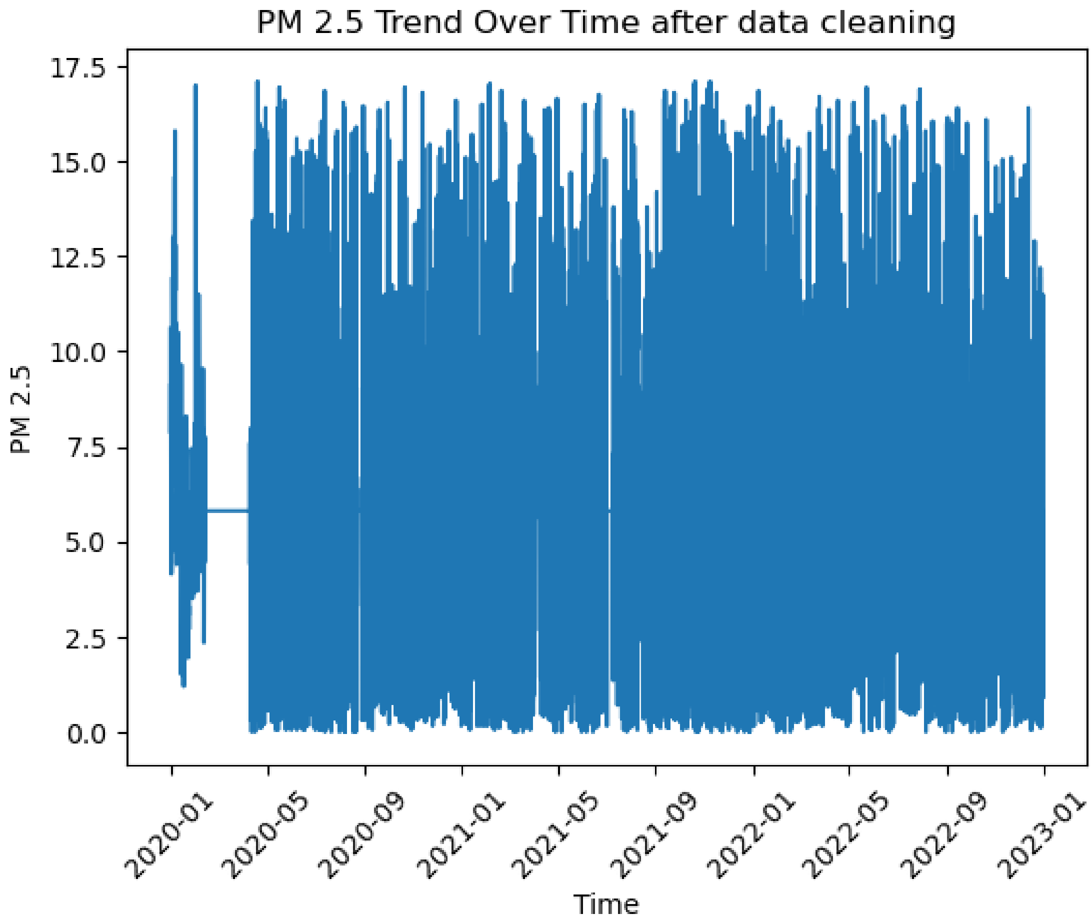
```
(24549, 11)
```

```
mean_df
```

```
            Time_Interval        NO        NO2       SO2   Air_temp  \
0     2020-01-01 00:00:00  5.250000  14.800000  0.326079  18.000000
1     2020-01-01 01:00:00  4.900000  15.050000  0.326079  18.000000
2     2020-01-01 02:00:00  2.350000  14.100000  0.000000  18.000000
3     2020-01-01 03:00:00  1.550000  16.000000  0.000000  18.000000
4     2020-01-01 04:00:00  1.600000  16.750000  0.000000  18.000000
...                   ...       ...        ...       ...        ...
26300 2022-12-31 20:00:00  0.500000   2.100000  0.200000  16.567666
26301 2022-12-31 21:00:00  0.500000   1.350000  0.200000  16.567666
26302 2022-12-31 22:00:00  0.500000   1.200000  0.400000  16.567666
26303                 NaT  9.995042  14.425854  0.326079  16.567666
26304                 NaT  9.995042  14.425854  0.326079  16.567666

       Rel_Humidity    Wind_Dir  Wind_speed       Lag1       Lag2
PM
0         76.000000  223.500000    2.200000   5.428717   5.428717
10.60000
1         77.600000  217.000000    2.100000  10.600000   5.428717
```

```
8.60000
2          79.100000  212.000000    1.750000   8.600000  10.600000
7.85000
3          80.200000  188.000000    1.200000   7.850000   8.600000
8.60000
4          80.350000  172.500000    1.050000   8.600000   7.850000
9.30000
...              ...         ...         ...        ...        ...
...
26300      69.785531  175.673297    2.777638  -0.250000  -0.750000
0.90000
26301      69.785531  175.673297    2.777638   0.900000  -0.250000
2.00000
26302      69.785531  175.673297    2.777638   2.000000   0.900000
1.95000
26303      69.785531  175.673297    2.777638   1.950000   2.000000
5.80859
26304      69.785531  175.673297    2.777638   5.428717   1.950000
5.80859

[24549 rows x 11 columns]
```

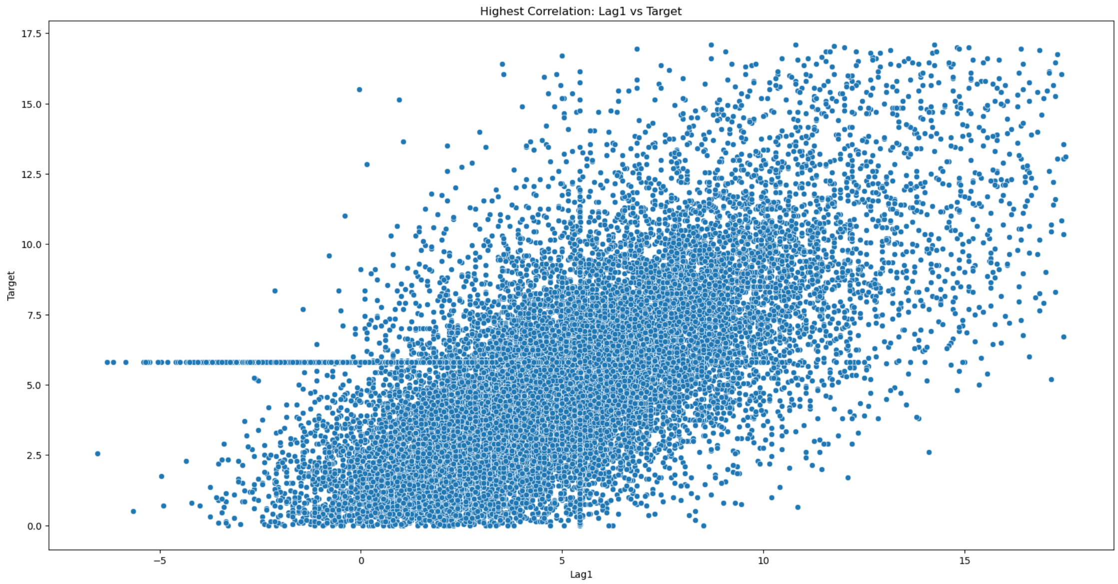
```
plt.plot(mean_df["Time_Interval"], mean_df["PM"])
plt.xlabel("Time")
plt.ylabel("PM 2.5")
plt.title("PM 2.5 Trend Over Time after data cleaning")
plt.xticks(rotation=45)
plt.show()
```




```python
correlation_matrix = mean_df.corr()
target_correlation = correlation_matrix['PM'].drop('PM')
highest_correlation_predictor = target_correlation.idxmax()
```

```
/var/folders/r9/ftlfg09n2rbgf2q9jy3b7j7h0000gn/T/
ipykernel_21959/1486501280.py:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  correlation_matrix = mean_df.corr()
```

```python
# Plotting the highest correlated predictor against the target
variable
sns.scatterplot(x=highest_correlation_predictor, y='PM', data=mean_df)

plt.xlabel(highest_correlation_predictor)
plt.ylabel('Target')
plt.title(f'Highest Correlation: {highest_correlation_predictor} vs
Target')
plt.gcf().set_size_inches((20, 10))
plt.show()
```



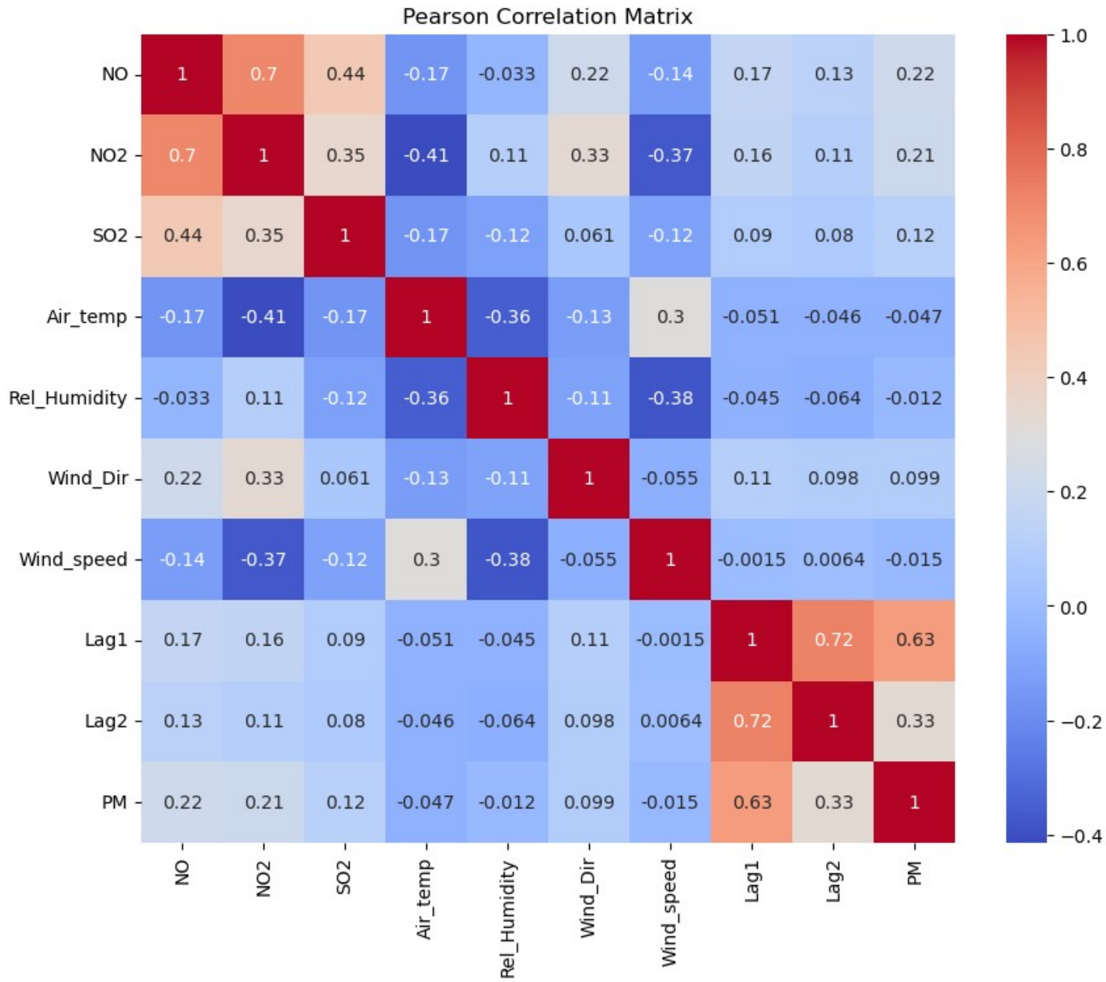

```
mean_df.describe()   #Statisical summary
```

```
                 NO           NO2           SO2      Air_temp
Rel_Humidity  \
count  24549.000000  24549.000000  24549.000000  24549.000000
24549.000000
mean       7.415754     13.285116      0.254007     16.727087
69.911950
std        9.047286     10.239914      0.261966      3.630268
11.795967
min       -1.750000     -0.900000     -1.000000      5.500000
35.950000
25%        1.100000      4.900000      0.050000     14.000000
61.500000
50%        4.550000     12.500000      0.250000     16.567666
69.800000
75%        9.995042     17.600000      0.350000     19.000000
79.400000
max       64.700000     49.500000      1.800000     27.500000
91.950000

           Wind_Dir    Wind_speed          Lag1          Lag2
PM
count  24549.000000  24549.000000  24549.000000  24549.000000
24549.000000
mean     174.269569      2.824292      5.004147      5.062617
5.399753
std       87.084853      1.456024      3.243318      3.315584
2.903126
min        4.000000      0.250000     -6.550000     -6.550000
0.000000
25%       92.500000      1.700000      2.950000      2.950000
```

```
3.400000
50%      197.500000      2.750000      5.250000      5.300000
5.750000
75%      232.500000      3.750000      6.500000      6.600000
6.500000
max      356.000000      7.150000     17.500000     17.550000
17.100000
```

```
# Compute the Pearson correlation matrix
correlation_matrix = mean_df.corr()

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Pearson Correlation Matrix')
plt.show()
```

```
/var/folders/r9/ftlfg09n2rbgf2q9jy3b7j7h0000gn/T/
ipykernel_21959/3291743950.py:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  correlation_matrix = mean_df.corr()
```



Based on the correlation coefficients: It appears that Lag1 and Lag2 have the highest positive correlations with PM2.5. Other variables such as NO, NO2, and SO2 also show moderate positive correlations. On the other hand, variables like Air_temp, Rel_Humidity and Wind_Dir have weak negative correlations with PM2.5.

```
# Selecting the features with the highest absolute correlation
coefficients
selected_features = ['NO', 'NO2', 'SO2','Lag1', 'Lag2']

# Create a new DataFrame with the selected features
df_selected = mean_df[selected_features + ['PM']]

# Split the dataset into features (X) and target variable (y)
X = df_selected.drop('PM', axis=1)
y = df_selected['PM']

# Split the data into 70% training and 30% testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

```python
reg_model = LinearRegression()
reg_model.fit(X_train, y_train)
y_pred = reg_model.predict(X_test)

# Calculate evaluation metrics
rmse_reg = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print("Linear Regression Model Performance:")
print("RMSE:", rmse_reg)
print("MAE:", mae)
print("R2 Score:", r2)
```

```
Linear Regression Model Performance:
RMSE: 2.1659689714490153
MAE: 1.6162891175679295
R2 Score: 0.44726052657038085
```

```python
import statsmodels.formula.api as smf
mod1 = smf.ols("PM ~ NO + NO2 + SO2 + Lag1 + Lag2", data=mean_df)
mod1_res = mod1.fit()
mod1_res.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:                     PM   R-squared:
0.449
Model:                            OLS   Adj. R-squared:
0.449
Method:                 Least Squares   F-statistic:
3999.
Date:                Fri, 16 Jun 2023   Prob (F-statistic):
0.00
Time:                        20:54:42   Log-Likelihood:
-53683.
No. Observations:               24549   AIC:
1.074e+05
Df Residuals:                   24543   BIC:
1.074e+05
Df Model:                           5

Covariance Type:            nonrobust

======================================================================
========
```

```
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
Intercept      2.5681      0.032     81.078      0.000       2.506
2.630
NO             0.0234      0.002     10.386      0.000       0.019
0.028
NO2            0.0137      0.002      7.197      0.000       0.010
0.017
SO2            0.1963      0.059      3.343      0.001       0.081
0.311
Lag1           0.7165      0.006    116.403      0.000       0.704
0.729
Lag2          -0.2289      0.006    -38.293      0.000      -0.241
-0.217
======================================================================
========
Omnibus:                     2021.777   Durbin-Watson:
1.600
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3153.086
Skew:                           0.638   Prob(JB):
0.00
Kurtosis:                       4.205   Cond. No.
89.3
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
"""
```

Multilayer Perceptron (MLP)

```
# Part 2: Single hidden layer with k = 25 neurons
# Create an MLPRegressor with default values and a single hidden layer
of 25 neurons
mlp = MLPRegressor(hidden_layer_sizes=(25, ), random_state=42)

# Train the model using the training dataset
mlp.fit(X_train, y_train)

MLPRegressor(hidden_layer_sizes=(25,), random_state=42)

# Make predictions on the testing dataset
y_pred_mlp = mlp.predict(X_test)
```

```python
# Evaluate the performance using mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred_mlp)
print("Mean Squared Error with default learning rate:", mse)
```

```
Mean Squared Error with default learning rate: 4.2905703603859955
```

```python
mse = mean_squared_error(y_test, y_pred_mlp)
rmse_mlp = mean_squared_error(y_test, y_pred_mlp, squared=False)
mae = mean_absolute_error(y_test, y_pred_mlp)
r2 = r2_score(y_test, y_pred_mlp)

# Print the evaluation metrics
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse_mlp)
print("Mean Absolute Error:", mae)
print("R2 Score:", r2)
```

```
Mean Squared Error: 4.2905703603859955
Root Mean Squared Error: 2.0713691994393457
Mean Absolute Error: 1.559268079477574
R2 Score: 0.49448849168583275
```

```python
k = 25  # Total number of neurons across two hidden layers
# Part 3: Experimenting with two hidden layers

best_mse = float('inf')
best_hidden_layers = None

# Iterate over different configurations of neurons in two hidden
layers
for neurons_in_first_layer in range(k-1, 0, -1):
    neurons_in_second_layer = k - neurons_in_first_layer

    # Create an MLPRegressor with two hidden layers
    mlp = MLPRegressor(hidden_layer_sizes=(neurons_in_first_layer,
neurons_in_second_layer), random_state=42)

    # Train the model using the training dataset
    mlp.fit(X_train, y_train)

    # Make predictions on the testing dataset
    y_pred_mlp = mlp.predict(X_test)

    # Evaluate the performance using mean squared error (MSE)
    mse = mean_squared_error(y_test, y_pred_mlp)

    # Check if this configuration gives the lowest MSE
    if mse < best_mse:
        best_mse = mse
        best_hidden_layers = (neurons_in_first_layer,
neurons_in_second_layer)
```

```
print("Best hidden layer configuration:", best_hidden_layers)
print("Lowest Mean Squared Error:", best_mse)
```

```
Best hidden layer configuration: (15, 10)
Lowest Mean Squared Error: 4.258910014126226
```

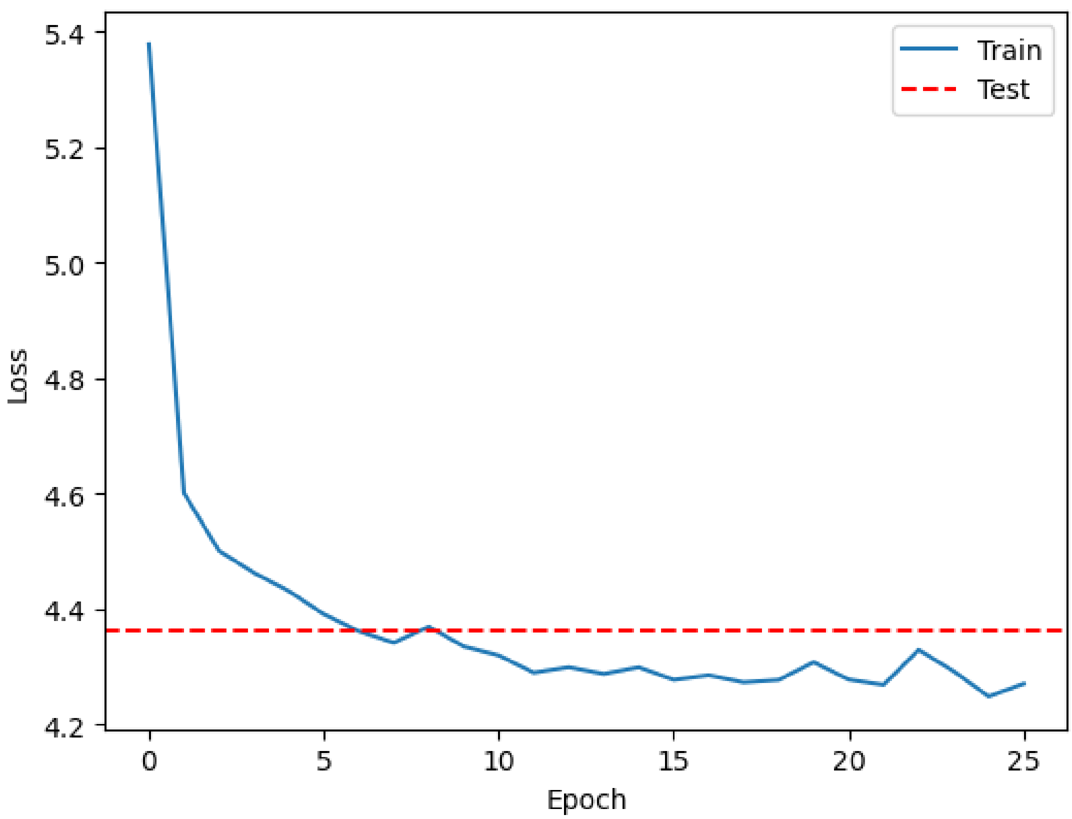
```
###the best configuration consists of two hidden layers with 15
neurons in the first layer and 10 neurons in the second layer.
##The lowest MSE achieved with this configuration is approximately
4.259.

##we can conclude that the architecture with (15, 10) neurons in the
two hidden layers performs slightly better than the default
architecture in part .
```

Long Short-Term Memory (LSTM)

```
# Normalize the data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape the input data for LSTM
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0],
X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0],
X_test_scaled.shape[1], 1)

##apply LSTM model
model = Sequential()
model.add(LSTM(25, activation='relu',
input_shape=(X_train_reshaped.shape[1], 1)))
model.add(Dense(1))

learning_rate = 0.01
batch_size = 4

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learnin
g_rate),
              loss='mean_squared_error')

history = model.fit(X_train_reshaped, y_train,
validation_data=(X_test_reshaped, y_test), batch_size=batch_size,
epochs=32, verbose=0)
```

```
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
```

```
train_loss = history.history['loss']
test_loss = model.evaluate(X_test_reshaped, y_test, verbose=0)

plt.plot(train_loss, label='Train')
plt.axhline(y=test_loss, color='r', linestyle='--', label='Test')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

best_epoch = np.argmin(test_loss) + 1
```




```
summary_stats = {
    'Mean': np.mean(train_loss),
    'Standard Deviation': np.std(train_loss),
    'Minimum': np.min(train_loss),
```

```
    'Best Epoch': best_epoch,
    'Maximum': np.max(train_loss),
    'Runtime': history.epoch[-1] + 1  # Epochs start from 0, so adding
1 to get the runtime
}

summary_stats
```

```
{'Mean': 4.376399333660419,
 'Standard Deviation': 0.21613654042756908,
 'Minimum': 4.248939514160156,
 'Best Epoch': 25,
 'Maximum': 5.378185272216797,
 'Runtime': 26}
```

```
best_epoch = 25

batch_sizes = [2, 4, 8, 16, 32, 64, 90 , 150]
summary_stats_batch = []

for batch_size in batch_sizes:
    model = Sequential()
    model.add(LSTM(25, activation='relu',
input_shape=(X_train_reshaped.shape[1], 1)))
    model.add(Dense(1))


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learnin
g_rate),
                  loss='mean_squared_error')

    start_time = time.time()
    history = model.fit(X_train_reshaped, y_train,
batch_size=batch_size, epochs=best_epoch, verbose=0)
    end_time = time.time()

    train_loss = history.history['loss']
    runtime = end_time - start_time

    # Calculate summary statistics
    mean_loss = np.mean(train_loss)
    std_loss = np.std(train_loss)
    min_loss = np.min(train_loss)
    max_loss = np.max(train_loss)

    summary_stats_batch.append({
        'Batch Size': batch_size,
        'Mean': mean_loss,
```

```python
        'Standard Deviation': std_loss,
        'Minimum': min_loss,
        'Maximum': max_loss,
        'Runtime': runtime
    })

summary_stats_batch
```

```
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
```

```
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.

[{'Batch Size': 2,
  'Mean': 4.4142498016357425,
  'Standard Deviation': 0.3453042507018569,
  'Minimum': 4.253872394561768,
  'Maximum': 6.041283130645752,
  'Runtime': 166.30285096168518},
 {'Batch Size': 4,
  'Mean': 4.382492961883545,
  'Standard Deviation': 0.3646838634618605,
  'Minimum': 4.240434646606445,
  'Maximum': 6.108152866363525,
  'Runtime': 81.73005795478821},
 {'Batch Size': 8,
  'Mean': 4.348611583709717,
  'Standard Deviation': 0.41452505050741245,
  'Minimum': 4.189599990844727,
  'Maximum': 6.330445289611816,
  'Runtime': 40.6296010017395},
 {'Batch Size': 16,
  'Mean': 4.409920616149902,
  'Standard Deviation': 0.5709795356460061,
  'Minimum': 4.187167167663574,
  'Maximum': 7.124744415283203,
  'Runtime': 21.053802967071533},
 {'Batch Size': 32,
  'Mean': 4.430623512268067,
  'Standard Deviation': 0.5748771832378529,
  'Minimum': 4.219583034515381,
  'Maximum': 7.179099082946777,
  'Runtime': 11.376978158950806},
 {'Batch Size': 64,
  'Mean': 4.7081369781494145,
  'Standard Deviation': 0.9844180699680888,
  'Minimum': 4.225187301635742,
  'Maximum': 8.683056831359863,
```

```
  'Runtime': 9.567314863204956},
 {'Batch Size': 90,
  'Mean': 4.731443424224853,
  'Standard Deviation': 1.1338990095597246,
  'Minimum': 4.25293493270874,
  'Maximum': 9.498494148254395,
  'Runtime': 8.35860300064087},
 {'Batch Size': 150,
  'Mean': 5.022337608337402,
  'Standard Deviation': 1.5644735628215338,
  'Minimum': 4.248717784881592,
  'Maximum': 11.382460594177246,
  'Runtime': 5.691593885421753}]
```

The batch size of 8 shows a relatively low mean value (4.35), indicating good overall performance in terms of minimizing the cost function. Additionally, it has a moderate standard deviation (0.41), suggesting reasonably consistent results across the runs. Moreover, the runtime for a batch size of 8 is shorter compared to smaller batch sizes (2, 4), making it a reasonable choice in terms of computational efficiency.

```
neuron_counts = [10, 25, 50, 75, 100]  # Define the different numbers
of neurons

results = []  # Store the results for each run

for neurons in neuron_counts:
    model = Sequential()
    model.add(LSTM(neurons, activation='relu',
input_shape=(X_train_reshaped.shape[1], 1)))
    model.add(Dense(1))


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
                  loss='mean_squared_error')

    history = model.fit(X_train_reshaped, y_train, batch_size=8,
epochs=26, verbose=0)

    train_loss = history.history['loss']
    runtime = len(train_loss)  # Runtime is the number of epochs

    result = {
        'Neuron Count': neurons,
        'Mean': np.mean(train_loss),
        'Standard Deviation': np.std(train_loss),
        'Minimum': np.min(train_loss),
```

```python
        'Maximum': np.max(train_loss),
        'Runtime': runtime
    }

    results.append(result)
```

```
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras
optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer,
i.e., `tf.keras.optimizers.legacy.Adam`.
```

```python
# Print the results
for result in results:
    print(result)
```

```
{'Neuron Count': 10, 'Mean': 4.468050296490009, 'Standard Deviation':
0.4784714206810787, 'Minimum': 4.279744625091553, 'Maximum':
6.799226760864258, 'Runtime': 26}
{'Neuron Count': 25, 'Mean': 4.373248210320106, 'Standard Deviation':
0.388681963775685, 'Minimum': 4.21902322769165, 'Maximum':
```

```
6.265530586242676, 'Runtime': 26}
{'Neuron Count': 50, 'Mean': 4.335052435214703, 'Standard Deviation':
0.2378184563130353, 'Minimum': 4.202245712280273, 'Maximum':
5.442875862121582, 'Runtime': 26}
{'Neuron Count': 75, 'Mean': 4.390933678700374, 'Standard Deviation':
0.5121579647218228, 'Minimum': 4.183339595794678, 'Maximum':
6.880396842956543, 'Runtime': 26}
{'Neuron Count': 100, 'Mean': 4.338151968442476, 'Standard Deviation':
0.34214244750766304, 'Minimum': 4.188814163208008, 'Maximum':
5.97617101669311 5, 'Runtime': 26}
```

```
# the optimal number of neurons would be 100.
# This is because the model with 100 neurons in the hidden layer
achieved a lower mean and
# standard deviation of the cost function compared to the other neuron
counts.
```

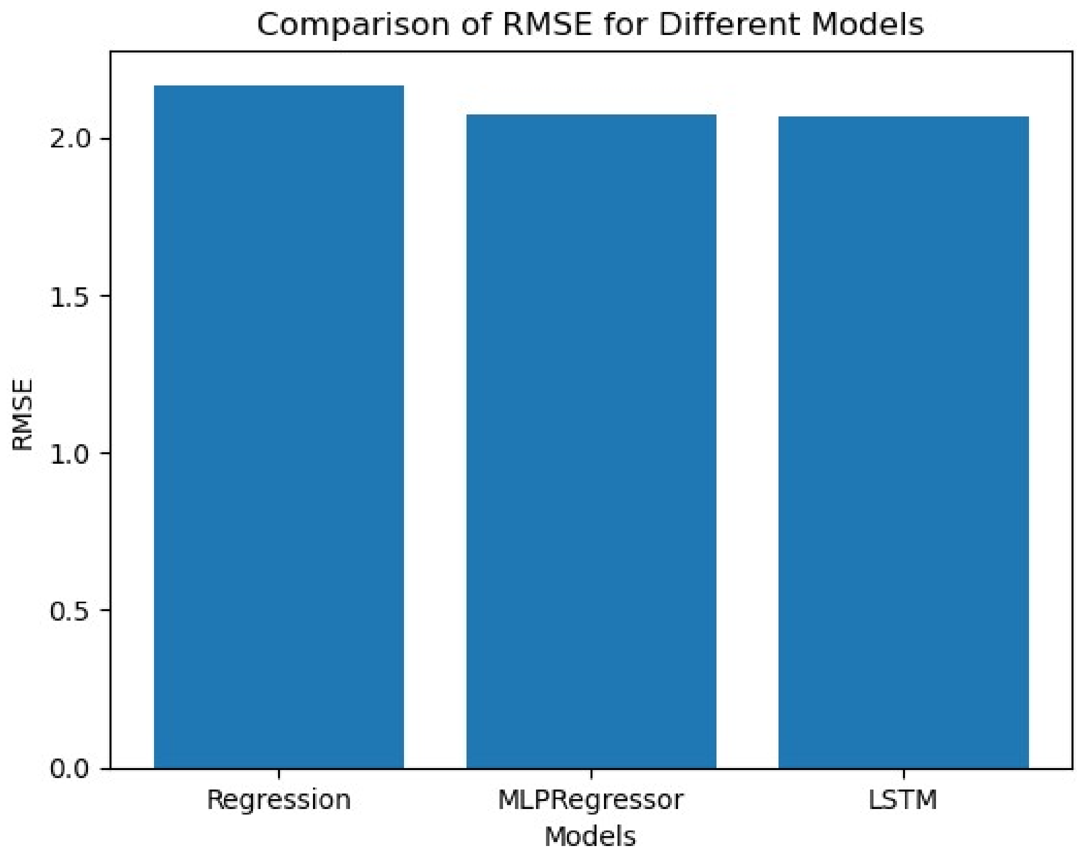
```
# Make predictions on the test set
y_pred_lstm = model.predict(X_test_reshaped)

# Calculate RMSE
rmse_lstm = np.sqrt(mean_squared_error(y_test, y_pred_lstm))

# Calculate MAE
mae = mean_absolute_error(y_test, y_pred_lstm)

# Calculate R2 score
r2 = r2_score(y_test, y_pred_lstm)

# Print the performance metrics
print("Root Mean Square Error (RMSE):", rmse_lstm)
print("Mean Absolute Error (MAE):", mae)
print("R2 Score:", r2)
```

```
231/231 [==============================] - 0s 927us/step
Root Mean Square Error (RMSE): 2.0629785790824524
Mean Absolute Error (MAE): 1.5489273762151918
R2 Score: 0.49857560893330066
```

Compare the performance of the models using RMSE. Which model performed better?

```
# Labels for the models
models = ['Regression', 'MLPRegressor', 'LSTM']

# RMSE values
rmse_values = [rmse_reg, rmse_mlp, rmse_lstm]

# Plotting the bar graph
plt.bar(models, rmse_values)
plt.xlabel('Models')
plt.ylabel('RMSE')
```

```python
plt.title('Comparison of RMSE for Different Models')

plt.show()
```




```python
print (rmse_reg)
print (rmse_mlp)
print (rmse_lstm)
```

```
2.1659689714490153
2.0713691994393457
2.0629785790824524
```

```python
###the LSTM model is considered the best model as it has the lowest
rmse value
```