```
#Check for pytorch being installed
import torch
print(torch.__version__)
```

```
2.3.0+cu121
```

```
#Libraries
import pandas as pd
import os
import functools
import numpy as np
from statsmodels.tsa.seasonal import STL
import matplotlib.pyplot as plt
import sklearn
from sklearn.impute import KNNImputer

# Set the maximum number of rows and columns to display
pd.set_option('display.max_rows', None)  # None means unlimited
pd.set_option('display.max_columns', None)  # None means unlimited
```

```
#Create dfs to handle data
US_df = pd.read_csv(r"C:\Users\User\Code\databook\data\US\CombinedDat


print(US_df)
```

```
                 DateTime  PM2.5    PM10     CO   NO2     O3  \
0      2013-01-01 00:00:00   12.0    16.0  0.400  20.9  0.012
1      2013-01-01 01:00:00   16.0    18.0  0.400  23.2  0.021
2      2013-01-01 02:00:00    9.0    10.0  0.300  18.3  0.000
3      2013-01-01 03:00:00    8.0    10.0  0.300  16.9  0.019
4      2013-01-01 04:00:00   10.0     7.0  0.400  19.9  0.016
5      2013-01-01 05:00:00   13.0    12.0  0.400  24.1  0.009
6      2013-01-01 06:00:00   12.0     9.0  0.400  17.8  0.017
7      2013-01-01 07:00:00    9.9     6.0  0.300  22.2  0.013
8      2013-01-01 08:00:00    6.0     6.0  0.300  18.1  0.014
9      2013-01-01 09:00:00    6.0     7.0  0.200  14.2  0.023
10     2013-01-01 10:00:00    9.8     9.0  0.200  15.7  0.029
11     2013-01-01 11:00:00    7.5     8.0  0.200  12.7  0.032
12     2013-01-01 12:00:00    6.9     3.0  0.200   6.4  0.015
13     2013-01-01 13:00:00   15.8     3.0  0.200   6.3  0.038
14     2013-01-01 14:00:00   13.9     1.0  0.200   8.0  0.038
15     2013-01-01 15:00:00   16.9     7.0  0.200   8.7  0.037
16     2013-01-01 16:00:00   21.6     5.0  0.200   8.6  0.031
17     2013-01-01 17:00:00   26.4    17.0  0.200  11.5  0.023
18     2013-01-01 18:00:00   31.9     7.0  0.300  12.2  0.020
19     2013-01-01 19:00:00   30.9    16.0  0.300  11.1  0.018
20     2013-01-01 20:00:00   30.9    16.0  0.300   9.1  0.026
21     2013-01-01 21:00:00   32.9     9.0  0.300   9.9  0.025
22     2013-01-01 22:00:00   25.1    13.0  0.200   6.9  0.019
23     2013-01-01 23:00:00   22.1    10.0  0.300   9.2  0.001
24     2013-01-02 00:00:00   17.0    10.0  0.200   6.9  0.022
25     2013-01-02 01:00:00   15.0     7.0  0.200   6.7  0.005
26     2013-01-02 02:00:00   25.9     5.0  0.200   4.9  0.004
```

```
27     2013-01-02 03:00:00   10.0    16.0  0.200   4.6  0.025
28     2013-01-02 04:00:00   31.6     3.0  0.200   5.1  0.008
29     2013-01-02 05:00:00   30.9    15.0  0.200   6.3  0.024
30     2013-01-02 06:00:00   25.9    18.0  0.200   8.3  0.023
31     2013-01-02 07:00:00   14.0    13.0  0.200  10.7  0.014
32     2013-01-02 08:00:00   21.0    18.0  0.300  10.1  0.013
33     2013-01-02 09:00:00   26.5    18.0  0.200   9.9  0.014
34     2013-01-02 10:00:00   16.0    17.0  0.200   7.9  0.025
35     2013-01-02 11:00:00   16.7    11.0  0.200   6.7  0.035
36     2013-01-02 12:00:00    9.0    10.0  0.200   6.0  0.017
37     2013-01-02 13:00:00   11.0    17.0  0.200   5.8  0.018
38     2013-01-02 14:00:00   12.0    10.0  0.200   6.7  0.030
39     2013-01-02 15:00:00   12.0     9.0  0.200   8.5  0.022
40     2013-01-02 16:00:00   13.0    10.0  0.300  10.8  0.027
41     2013-01-02 17:00:00   18.7    15.0  0.300  16.3  0.020
42     2013-01-02 18:00:00   17.7    14.0  0.300  21.3  0.017
43     2013-01-02 19:00:00   12.0    13.0  0.300  23.1  0.016
44     2013-01-02 20:00:00   12.0    19.0  0.400  29.6  0.002
45     2013-01-02 21:00:00   15.7    13.0  0.300  17.9  0.020
46     2013-01-02 22:00:00   18.9    16.0  0.300  15.5  0.022
47     2013-01-02 23:00:00   13.2    15.0  0.300  17.2  0.001
48     2013-01-03 00:00:00   13.0    18.0  0.300  15.9  0.000
49     2013-01-03 01:00:00   13.0    13.0  0.300  16.7  0.018
50     2013-01-03 02:00:00   20.0    14.0  0.300  16.2  0.017
51     2013-01-03 03:00:00   14.0    14.0  0.300  14.8  0.005
52     2013-01-03 04:00:00   14.0    16.0  0.200  19.3  0.010
53     2013-01-03 05:00:00   13.0    24.0  0.300  24.1  0.011
54     2013-01-03 06:00:00   26.4    25.0  0.400  29.5  0.002
55     2013-01-03 07:00:00   32.0    28.0  0.400  29.8  0.013
56     2013-01-03 08:00:00   23.0    28.0  0.400  33.7  0.005
```

```
 1 # Function to handle datetime conversion and setting index
 2 def preprocess_dataframe(df):
 3
 4     #Ensure that the DateTime is set to a processable format
 5     df['DateTime'] = pd.to_datetime(df['DateTime'])
 6     df.set_index('DateTime', inplace=True)      #set DateTime to inde
 7
 8     # Convert 'No data' to NaN and change the data type to float so t
 9     #df.replace('No data', np.nan, inplace=True)
10     df = df.apply(pd.to_numeric, errors='coerce')
11
12
13
14     return df
```

```
 1 #Function to remove Outliers using InterQuartile Range method
 2 def remove_outliers(df, column):
 3     # Calculate Q1, Q3 and IQR
 4     Q1 = df[column].quantile(0.25)
 5     Q3 = df[column].quantile(0.75)
 6     IQR = Q3 - Q1
 7
 8     # Define bounds for outliers
 9     lower_bound = Q1 - 2.5 * IQR
10     upper_bound = Q3 + 3.5 * IQR
```

```
    print(f"Working on {column}")

    # Replace outliers with NaN
    def handle_outliers(x):
        original_x = x  # store original value
        if x < lower_bound or x > upper_bound or x < 0:
            x = np.nan
            print(f"Original: {original_x}, Modified: {x}")  # Print
        return x

    df[column] = df[column].apply(handle_outliers)


    return df
```

```
#Function to remove Outliers using STL method
def remove_outliers_STL(df, column, seasonal=49):

    #Replace any negative values with NaN
    df[column] = df[column].apply(lambda x: np.nan if x < 0 else x)

    #Print length of column
    print(f"The length of the {column} column is {len(df[column].drop

    # STL Decomposition
    stl = STL(df[column], seasonal=seasonal , robust=True)  #Seasonal
    result = stl.fit()
    df['residuals'] = result.resid

    # Detect anomalies in residuals
    threshold = 5 * np.std(df['residuals'].dropna())  # Calculate thr
    df['stl_anomaly'] = np.abs(df['residuals']) > threshold  # Identi

    print(f"Data being checked: {column}")
    checking_df = df[[column,'stl_anomaly']]

    #Checking quality of annomaly detection
    anomalies = checking_df[df['stl_anomaly']== True]

    print(anomalies)

     # Replace anomalies with NaN
    df.loc[df['stl_anomaly'], column] = np.nan  # Replace detected an


    #remove stl anomoly column so that it can be used again for the n
    df.drop('stl_anomaly', axis=1, inplace=True)

    return df

```

```
#Removing outliers from Data
```

```
#Removing outliers from Data
def apply_outliers_removal(df, time_column, seasonal=49):
    '''
    Parameters:
        Input:
            df - Dataframe to be processed by function
            time_column - column that will be ignored by function.
    '''
    for col in df.columns:
        if col != time_column:
            df = remove_outliers_STL(df, col, seasonal)
    return df

```

```
#Function applying STL to each column of the dataset to fill in the m

def STL_fill_in_data(df, period):
    '''
    Parameters:
    Input: df - Dataframe who's data will be filled
    period - The time period in which the pattern for the seasonal tr
    '''

    for column in df.columns:
        if df[column].isnull().any():
            # Ensure the column is not all NaN and has enough data po
            if df[column].dropna().shape[0] < period:
                print(f"Not enough data points in {column} to apply S
                continue

            # Apply STL decomposition
            stl = STL(df[column].interpolate(), seasonal=period)
            res = stl.fit()

            # Deseasonalize and impute
            seasonal_component = res.seasonal
            deseasonalised = df[column] - seasonal_component
            deseasonalised_imputed = deseasonalised.interpolate(metho

            # Recombine and fill missing values
            df_imputed = deseasonalised_imputed + seasonal_component
            imputed_indices = df[df[column].isnull()].index

            # Ensure no negative values
            df_imputed = df_imputed.apply(lambda x: max(x, 0))

            df.loc[imputed_indices, column] = df_imputed[imputed_indi

    return df

```

```
def KNN_fill_in_data(df, n_neighbors=5, weights='distance'):
    """
```

```
    Impute missing data in each column of the dataset using the KNN m

    Parameters:
    df (pd.DataFrame): DataFrame whose data will be filled.
    n_neighbors (int): Number of neighboring samples to use for imput

    Returns:
    pd.DataFrame: DataFrame with missing values imputed using KNN.
    """
    imputer = KNNImputer(n_neighbors=n_neighbors, weights = weights)

    # Ensure all data is numeric for KNN imputer, converts non-numeri
    df_numeric = df.apply(pd.to_numeric, errors='coerce')

    # Apply KNN imputation
    imputed_data = imputer.fit_transform(df_numeric)

    # Create a DataFrame with the imputed data (NB changed from: colu
    df_imputed = pd.DataFrame(imputed_data, columns=df_numeric.column

    # Replace any resulting negative values with zero, assuming only
    df_imputed = df_imputed.applymap(lambda x: max(x, 0))

    return df_imputed
```

```
print(US_df.columns.to_list)
```

```
<bound method IndexOpsMixin.tolist of Index(['DateTime', 'PM2.5', 'PM10',
       'Temp'],
      dtype='object')>
```

```
#US data Cleaning
#Create dfs to handle data
US_df = pd.read_csv(r"C:\Users\User\Code\databook\data\US\CombinedDat

US_PM_df = US_df[['DateTime','PM2.5','PM10']]

US_pp_df = preprocess_dataframe(US_PM_df)
US_or_df = apply_outliers_removal(US_pp_df, 'DateTime')
US_or_df2 = US_or_df.copy(deep=True) #Creating a completely seperate
US_or_df2['DateTime'] = US_or_df2.index
US_K_fill = KNN_fill_in_data(US_or_df2, 26)


#print(US_fill_df)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 92054
Data being checked: PM2.5
```

```
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 93064
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
Cell In[101], line 11
      9 US_or_df2 = US_or_df.copy(deep=True) #Creating a completely
seperate object so that the original dataframe can be kept
     10 US_or_df2['DateTime'] = US_or_df2.index
---> 11 US_K_fill = KNN_fill_in_data(US_or_df2, 26)
     14 #print(US_fill_df)

Cell In[100], line 21, in KNN_fill_in_data(df, n_neighbors, weights)
     18 imputed_data = imputer.fit_transform(df_numeric)
     20 # Create a DataFrame with the imputed data (NB changed from:
columns=df.columns )
---> 21 df_imputed = pd.DataFrame(imputed_data,
columns=df_numeric.columns, index=df.index)
     23 # Replace any resulting negative values with zero, assuming only
non-negative values are acceptable
     24 df_imputed = df_imputed.applymap(lambda x: max(x, 0))

File c:\Users\User\Code\databook\Lib\site-
packages\pandas\core\frame.py:827, in DataFrame.__init__(self, data,
index, columns, dtype, copy)
    816         mgr = dict_to_mgr(
    817             # error: Item "ndarray" of "Union[ndarray, Series,
Index]" has no
    818             # attribute "name"
   (...)
    824             copy=_copy,
    825         )
    826     else:
--> 827         mgr = ndarray_to_mgr(
    828             data,
    829             index,
    830             columns,
    831             dtype=dtype,
    832             copy=copy,
    833             typ=manager,
    834         )
    836 # For data is list-like, or Iterable (will consume into list)
    837 elif is_list_like(data):

File c:\Users\User\Code\databook\Lib\site-
packages\pandas\core\internals\construction.py:336, in
ndarray_to_mgr(values, index, columns, dtype, copy, typ)
    331 # _prep_ndarraylike ensures that values.ndim == 2 at this point
    332 index, columns = _get_axes(
    333     values.shape[0], values.shape[1], index=index, columns=columns
    334 )
--> 336 _check_values_indices_shape_match(values, index, columns)
```

```
#UK data cleaning
UK_df = pd.read_csv(r"C:\Users\User\Code\databook\data\UK\AllData\UKF

UK_PM_df = UK_df[['DateTime','PM2.5','PM10']]
UK_pp_df = preprocess_dataframe(UK_PM_df)
print(UK_pp_df)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
                     PM2.5     PM10
DateTime
2019-01-01 01:00:00  14.741   19.328
2019-01-01 02:00:00  11.408   16.705
2019-01-01 03:00:00   5.407   10.074
2019-01-01 04:00:00   5.503   10.326
2019-01-01 05:00:00   5.755    9.708
2019-01-01 06:00:00   5.527    9.518
2019-01-01 07:00:00   5.667    9.802
2019-01-01 08:00:00   5.512    8.935
2019-01-01 09:00:00   5.576    9.508
2019-01-01 10:00:00   5.780    9.294
2019-01-01 11:00:00   5.616    9.060
2019-01-01 12:00:00   4.002    7.053
2019-01-01 13:00:00   3.767    5.629
2019-01-01 14:00:00   3.788    6.083
2019-01-01 15:00:00   4.141    6.405
2019-01-01 16:00:00   4.679    7.407
2019-01-01 17:00:00   4.531    6.830
2019-01-01 18:00:00   6.571   11.988
2019-01-01 19:00:00   8.677   17.038
2019-01-01 20:00:00   8.935   16.323
2019-01-01 21:00:00   6.283   11.761
2019-01-01 22:00:00   5.826   10.900
2019-01-01 23:00:00   5.973   11.615
2019-01-02 00:00:00   5.973   11.855
2019-01-02 01:00:00  10.267   11.835
2019-01-02 02:00:00   8.309    9.584
2019-01-02 03:00:00   6.833    8.250
2019-01-02 04:00:00   4.404    5.042
2019-01-02 05:00:00   4.539    5.730
2019-01-02 06:00:00   5.783    7.555
2019-01-02 07:00:00   6.381    8.306
2019-01-02 08:00:00   7.630    9.329
2019-01-02 09:00:00   8.259   11.255
2019-01-02 10:00:00  11.094   16.608
2019-01-02 11:00:00  10.472   12.740
2019-01-02 12:00:00  10.675   13.998
2019-01-02 13:00:00  12.769   16.115
2019-01-02 14:00:00   8.520   10.795
2019-01-02 15:00:00   8.267   11.287
2019-01-02 16:00:00   6.920    9.032
2019-01-02 17:00:00   7.923   10.682
2019-01-02 18:00:00  11.163   13.740
2019-01-02 19:00:00  10.962   14.248
```

```
2019-01-02 20:00:00   9.688   13.743
2019-01-02 21:00:00   9.557   13.740
2019-01-02 22:00:00  10.375   13.145
2019-01-02 23:00:00  10.576   13.163
2019-01-03 00:00:00  10.300   12.880
2019-01-03 01:00:00  15.151   18.780
2019-01-03 02:00:00  15.061   19.070
```

```
#US data cleaning
US_df = pd.read_csv(r"C:\Users\User\Code\databook\data\US\CombinedDat

US_PM_df = US_df[['DateTime','PM2.5','PM10']]
US_pp_df = preprocess_dataframe(US_PM_df)
US_or_df = apply_outliers_removal(US_pp_df, 'DateTime')
US_or_df2 = US_or_df.copy(deep=True) #Creating a completely seperate
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 92054
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 93064
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
US_or_df3 = US_or_df.copy(deep=True) #Creating a completely seperate

US_or_df3['DateTime'] = US_or_df3.index

US_or_df3.drop('residuals', axis=1, inplace=True)

```

```
US_K_fill = KNN_fill_in_data(US_or_df3, 26)
US_K_fill.drop('DateTime', axis=1, inplace=True)
print(US_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\3676177356.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                         PM2.5         PM10
DateTime
2013-01-01 00:00:00   12.000000    16.000000
2013-01-01 01:00:00   16.000000    18.000000
2013-01-01 02:00:00    9.000000    10.000000
2013-01-01 03:00:00    8.000000    10.000000
2013-01-01 04:00:00   10.000000     7.000000
2013-01-01 05:00:00   13.000000    12.000000
2013-01-01 06:00:00   12.000000     9.000000
2013-01-01 07:00:00    9.900000     6.000000
```

```
2013-01-01 07:00:00    9.900000     8.000000
2013-01-01 08:00:00    6.000000     6.000000
2013-01-01 09:00:00    6.000000     7.000000
2013-01-01 10:00:00    9.800000     9.000000
2013-01-01 11:00:00    7.500000     8.000000
2013-01-01 12:00:00    6.900000     3.000000
2013-01-01 13:00:00   15.800000     3.000000
2013-01-01 14:00:00   13.900000     1.000000
2013-01-01 15:00:00   16.900000     7.000000
2013-01-01 16:00:00   21.600000     5.000000
2013-01-01 17:00:00   26.400000    17.000000
2013-01-01 18:00:00   31.900000     7.000000
2013-01-01 19:00:00   30.900000    16.000000
2013-01-01 20:00:00   30.900000    16.000000
2013-01-01 21:00:00   32.900000     9.000000
2013-01-01 22:00:00   25.100000    13.000000
2013-01-01 23:00:00   22.100000    10.000000
2013-01-02 00:00:00   17.000000    10.000000
2013-01-02 01:00:00   15.000000     7.000000
2013-01-02 02:00:00   25.900000     5.000000
2013-01-02 03:00:00   10.000000    16.000000
2013-01-02 04:00:00   31.600000     3.000000
2013-01-02 05:00:00   30.900000    15.000000
2013-01-02 06:00:00   25.900000    18.000000
2013-01-02 07:00:00   14.000000    13.000000
2013-01-02 08:00:00   21.000000    18.000000
2013-01-02 09:00:00   26.500000    18.000000
2013-01-02 10:00:00   16.000000    17.000000
2013-01-02 11:00:00   16.700000    11.000000
2013-01-02 12:00:00    9.000000    10.000000
2013-01-02 13:00:00   11.000000    17.000000
2013-01-02 14:00:00   12.000000    10.000000
2013-01-02 15:00:00   12.000000     9.000000
2013-01-02 16:00:00   13.000000    10.000000
2013-01-02 17:00:00   18.700000    15.000000
2013-01-02 18:00:00   17.700000    14.000000
2013-01-02 19:00:00   12.000000    13.000000
2013-01-02 20:00:00   12.000000    19.000000
2013-01-02 21:00:00   15.700000    13.000000
2013-01-02 22:00:00   18.900000    16.000000
2013-01-02 23:00:00   13.200000    15.000000
2013-01-03 00:00:00   13.000000    18.000000
2013-01-03 01:00:00   13.000000    13.000000
2013-01-03 02:00:00   20.000000    14.000000
2013-01-03 03:00:00   14.000000    14.000000
2013-01-03 04:00:00   14.000000    16.000000
2013-01-03 05:00:00   13.000000    24.000000
```

```
 1 #UK data cleaning
 2 UK_df = pd.read_csv(r"C:\Users\User\Code\databook\data\UK\AllData\UKF
 3
 4 UK_PM_df = UK_df[['DateTime','PM2.5','PM10']]
 5 UK_pp_df = preprocess_dataframe(UK_PM_df)
 6 UK_or_df = apply_outliers_removal(UK_pp_df, 'DateTime')
 7 UK_or_df2 = UK_or_df.copy(deep=True) #Creating a completely seperate
 8 period = 167  #Using weekly period data (24 hours * 7 days)-1 1 hour
 9 #UK_fill_df = STL_fill_in_data(UK_or_df2, period)
10
11
```

```
11
12 #print(UK_fill_df)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 43337
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 43336
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
1 UK_or_df3 = UK_or_df.copy(deep=True) #Creating a completely seperate
2
3 UK_or_df3['DateTime'] = UK_or_df3.index
4
5 UK_or_df3.drop('residuals', axis=1, inplace=True)
6
7
8
9 print(UK_or_df3)
```

```
                     PM2.5     PM10            DateTime
DateTime
2019-01-01 01:00:00  14.741   19.328 2019-01-01 01:00:00
2019-01-01 02:00:00  11.408   16.705 2019-01-01 02:00:00
2019-01-01 03:00:00   5.407   10.074 2019-01-01 03:00:00
2019-01-01 04:00:00   5.503   10.326 2019-01-01 04:00:00
2019-01-01 05:00:00   5.755    9.708 2019-01-01 05:00:00
2019-01-01 06:00:00   5.527    9.518 2019-01-01 06:00:00
2019-01-01 07:00:00   5.667    9.802 2019-01-01 07:00:00
2019-01-01 08:00:00   5.512    8.935 2019-01-01 08:00:00
2019-01-01 09:00:00   5.576    9.508 2019-01-01 09:00:00
2019-01-01 10:00:00   5.780    9.294 2019-01-01 10:00:00
2019-01-01 11:00:00   5.616    9.060 2019-01-01 11:00:00
2019-01-01 12:00:00   4.002    7.053 2019-01-01 12:00:00
2019-01-01 13:00:00   3.767    5.629 2019-01-01 13:00:00
2019-01-01 14:00:00   3.788    6.083 2019-01-01 14:00:00
2019-01-01 15:00:00   4.141    6.405 2019-01-01 15:00:00
2019-01-01 16:00:00   4.679    7.407 2019-01-01 16:00:00
2019-01-01 17:00:00   4.531    6.830 2019-01-01 17:00:00
2019-01-01 18:00:00   6.571   11.988 2019-01-01 18:00:00
2019-01-01 19:00:00   8.677   17.038 2019-01-01 19:00:00
2019-01-01 20:00:00   8.935   16.323 2019-01-01 20:00:00
2019-01-01 21:00:00   6.283   11.761 2019-01-01 21:00:00
2019-01-01 22:00:00   5.826   10.900 2019-01-01 22:00:00
2019-01-01 23:00:00   5.973   11.615 2019-01-01 23:00:00
2019-01-02 00:00:00   5.973   11.855 2019-01-02 00:00:00
2019-01-02 01:00:00  10.267   11.835 2019-01-02 01:00:00
2019-01-02 02:00:00   8.309    9.584 2019-01-02 02:00:00
```

```
2019-01-02 03:00:00   6.833    8.250 2019-01-02 03:00:00
2019-01-02 04:00:00   4.404    5.042 2019-01-02 04:00:00
2019-01-02 05:00:00   4.539    5.730 2019-01-02 05:00:00
2019-01-02 06:00:00   5.783    7.555 2019-01-02 06:00:00
2019-01-02 07:00:00   6.381    8.306 2019-01-02 07:00:00
2019-01-02 08:00:00   7.630    9.329 2019-01-02 08:00:00
2019-01-02 09:00:00   8.259   11.255 2019-01-02 09:00:00
2019-01-02 10:00:00  11.094   16.608 2019-01-02 10:00:00
2019-01-02 11:00:00  10.472   12.740 2019-01-02 11:00:00
2019-01-02 12:00:00  10.675   13.998 2019-01-02 12:00:00
2019-01-02 13:00:00  12.769   16.115 2019-01-02 13:00:00
2019-01-02 14:00:00   8.520   10.795 2019-01-02 14:00:00
2019-01-02 15:00:00   8.267   11.287 2019-01-02 15:00:00
2019-01-02 16:00:00   6.920    9.032 2019-01-02 16:00:00
2019-01-02 17:00:00   7.923   10.682 2019-01-02 17:00:00
2019-01-02 18:00:00  11.163   13.740 2019-01-02 18:00:00
2019-01-02 19:00:00  10.962   14.248 2019-01-02 19:00:00
2019-01-02 20:00:00   9.688   13.743 2019-01-02 20:00:00
2019-01-02 21:00:00   9.557   13.740 2019-01-02 21:00:00
2019-01-02 22:00:00  10.375   13.145 2019-01-02 22:00:00
2019-01-02 23:00:00  10.576   13.163 2019-01-02 23:00:00
2019-01-03 00:00:00  10.300   12.880 2019-01-03 00:00:00
2019-01-03 01:00:00  15.151   18.780 2019-01-03 01:00:00
2019-01-03 02:00:00  15.061   19.070 2019-01-03 02:00:00
2019-01-03 03:00:00  11.498   14.048 2019-01-03 03:00:00
2019-01-03 04:00:00  12.234   14.243 2019-01-03 04:00:00
2019-01-03 05:00:00  18.182   20.488 2019-01-03 05:00:00
2019-01-03 06:00:00  20.382   22.808 2019-01-03 06:00:00
2019-01-03 07:00:00  18.797   22.340 2019-01-03 07:00:00
2019-01-03 08:00:00  21.356   25.893 2019-01-03 08:00:00
```

```
1 UK_K_fill = KNN_fill_in_data(UK_or_df3, 26)
2 UK_K_fill.drop('DateTime', axis=1, inplace=True)
3 print(UK_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\3676177356.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                         PM2.5        PM10
DateTime
2019-01-01 01:00:00  14.741000   19.328000
2019-01-01 02:00:00  11.408000   16.705000
2019-01-01 03:00:00   5.407000   10.074000
2019-01-01 04:00:00   5.503000   10.326000
2019-01-01 05:00:00   5.755000    9.708000
2019-01-01 06:00:00   5.527000    9.518000
2019-01-01 07:00:00   5.667000    9.802000
2019-01-01 08:00:00   5.512000    8.935000
2019-01-01 09:00:00   5.576000    9.508000
2019-01-01 10:00:00   5.780000    9.294000
2019-01-01 11:00:00   5.616000    9.060000
2019-01-01 12:00:00   4.002000    7.053000
2019-01-01 13:00:00   3.767000    5.629000
2019-01-01 14:00:00   3.788000    6.083000
2019-01-01 15:00:00   4.141000    6.405000
2019-01-01 16:00:00   4.679000    7.407000
2019-01-01 17:00:00   4.531000    6.830000
2019-01-01 18:00:00   6.571000   11.988000
2019-01-01 19:00:00   8.677000   17.038000
2019-01-01 20:00:00   8.935000   16.333000
```

```
2019-01-01 20:00:00   8.935000   16.323000
2019-01-01 21:00:00   6.283000   11.761000
2019-01-01 22:00:00   5.826000   10.900000
2019-01-01 23:00:00   5.973000   11.615000
2019-01-02 00:00:00   5.973000   11.855000
2019-01-02 01:00:00  10.267000   11.835000
2019-01-02 02:00:00   8.309000    9.584000
2019-01-02 03:00:00   6.833000    8.250000
2019-01-02 04:00:00   4.404000    5.042000
2019-01-02 05:00:00   4.539000    5.730000
2019-01-02 06:00:00   5.783000    7.555000
2019-01-02 07:00:00   6.381000    8.306000
2019-01-02 08:00:00   7.630000    9.329000
2019-01-02 09:00:00   8.259000   11.255000
2019-01-02 10:00:00  11.094000   16.608000
2019-01-02 11:00:00  10.472000   12.740000
2019-01-02 12:00:00  10.675000   13.998000
2019-01-02 13:00:00  12.769000   16.115000
2019-01-02 14:00:00   8.520000   10.795000
2019-01-02 15:00:00   8.267000   11.287000
2019-01-02 16:00:00   6.920000    9.032000
2019-01-02 17:00:00   7.923000   10.682000
2019-01-02 18:00:00  11.163000   13.740000
2019-01-02 19:00:00  10.962000   14.248000
2019-01-02 20:00:00   9.688000   13.743000
2019-01-02 21:00:00   9.557000   13.740000
2019-01-02 22:00:00  10.375000   13.145000
2019-01-02 23:00:00  10.576000   13.163000
2019-01-03 00:00:00  10.300000   12.880000
2019-01-03 01:00:00  15.151000   18.780000
2019-01-03 02:00:00  15.061000   19.070000
2019-01-03 03:00:00  11.498000   14.048000
2019-01-03 04:00:00  12.234000   14.243000
2019-01-03 05:00:00  18.182000   20.488000
2019-01-03 06:00:00  20.382000   22.808000
```

```
 1 #Columbia data cleaning
 2 Columbia_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Columbia
 3
 4 Columbia_PM_df = Columbia_df[['DateTime','PM2.5','PM10']]
 5 Columbia_pp_df = preprocess_dataframe(Columbia_PM_df)
 6 Columbia_or_df = apply_outliers_removal(Columbia_pp_df, 'DateTime')
 7
 8 Columbia_or_df3 = Columbia_or_df.copy(deep=True) #Creating a complete
 9
10 Columbia_or_df3['DateTime'] = Columbia_or_df3.index
11
12 Columbia_or_df3.drop('residuals', axis=1, inplace=True)
13
14
15
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
```

```
The length of the PM2.5 column is 46076
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 47059
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
1 Columbia_K_fill = KNN_fill_in_data(Columbia_or_df3, 3)
2 Columbia_K_fill.drop('DateTime', axis=1, inplace=True)
3 print(Columbia_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\1136749358.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                        PM2.5        PM10
DateTime
2016-01-01 01:00:00   28.800000   33.000000
2016-01-01 02:00:00   20.260024   16.000000
2016-01-01 03:00:00   16.000000   16.000000
2016-01-01 04:00:00   11.700000   16.000000
2016-01-01 05:00:00    9.100000   17.000000
2016-01-01 06:00:00    8.900000   10.000000
2016-01-01 07:00:00    7.000000   22.000000
2016-01-01 08:00:00    5.800000   12.000000
2016-01-01 09:00:00    3.500000   26.000000
2016-01-01 10:00:00    3.400000   40.000000
2016-01-01 11:00:00    6.400000   73.000000
2016-01-01 12:00:00    8.200000   63.000000
2016-01-01 13:00:00    8.300000   42.000000
2016-01-01 14:00:00    7.800000   32.000000
2016-01-01 15:00:00    6.600000   52.000000
2016-01-01 16:00:00   10.000000   34.000000
2016-01-01 17:00:00   10.900000   31.000000
2016-01-01 18:00:00   10.900000   38.000000
2016-01-01 19:00:00   13.700000   24.000000
2016-01-01 20:00:00   17.200000   34.000000
2016-01-01 21:00:00   18.300000   36.000000
2016-01-01 22:00:00   18.600000   45.000000
2016-01-01 23:00:00   16.700000   26.000000
2016-01-02 00:00:00   12.100000   16.000000
2016-01-02 01:00:00    9.100000   22.000000
2016-01-02 02:00:00    7.900000   25.000000
2016-01-02 03:00:00    7.500000   27.000000
2016-01-02 04:00:00    8.700000   35.000000
2016-01-02 05:00:00   12.400000   38.000000
2016-01-02 06:00:00   16.700000   34.000000
2016-01-02 07:00:00   18.400000   39.000000
2016-01-02 08:00:00   17.800000   35.000000
2016-01-02 09:00:00   13.100000   30.000000
2016-01-02 10:00:00   13.200000   33.000000
2016-01-02 11:00:00   14.300000   33.000000
2016-01-02 12:00:00   15.600000   47.000000
2016-01-02 13:00:00   13.800000   43.000000
2016-01-02 14:00:00   12.500000   52.000000
2016-01-02 15:00:00   13.200000   34.000000
```

```
2016-01-02 16:00:00   12.500000   31.000000
2016-01-02 17:00:00   13.200000   23.000000
2016-01-02 18:00:00   15.200000   33.000000
2016-01-02 19:00:00   17.800000   30.000000
2016-01-02 20:00:00   17.800000   35.000000
2016-01-02 21:00:00   17.000000   46.000000
2016-01-02 22:00:00   17.900000   42.000000
2016-01-02 23:00:00   18.000000   39.000000
2016-01-03 00:00:00   17.700000   29.000000
2016-01-03 01:00:00   13.400000   18.000000
2016-01-03 02:00:00    9.600000   40.000000
2016-01-03 03:00:00    7.800000   19.000000
2016-01-03 04:00:00    9.000000   38.000000
2016-01-03 05:00:00   11.000000   61.000000
2016-01-03 06:00:00   13.500000   66.000000
```

```
def check_data_intervals(df, time_column):
    df[time_column] = pd.to_datetime(df[time_column])
    #df = df.sort_values(by=time_column)
    df['diff'] = df[time_column].diff().dropna()
    print(df['diff'].describe())
    unusual_intervals = df[df['diff'] != pd.Timedelta(hours=1)]
    print("Unusual intervals:", unusual_intervals)


```

```
Australia_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Austral

Australia_PM_df = Australia_df[['DateTime','PM2.5','PM10']]
#Australia_pp_df = preprocess_dataframe(Australia_PM_df)
check_data_intervals(Australia_PM_df, 'DateTime')
```

```
count                         35063
mean                0 days 01:00:00
std      10 days 14:54:48.344591630
min             -324 days +01:00:00
25%                 0 days 01:00:00
50%                 0 days 01:00:00
75%                 0 days 01:00:00
max                30 days 01:00:00
Name: diff, dtype: object
Unusual intervals:                   DateTime  PM2.5   PM10
0     2020-01-01 01:00:00   15.0   43.5                NaT
24    2020-02-01 01:00:00   21.9   37.4    30 days 01:00:00
48    2020-03-01 01:00:00    NaN    NaN    28 days 01:00:00
72    2020-04-01 01:00:00    NaN    NaN    30 days 01:00:00
96    2020-05-01 01:00:00    NaN    NaN    29 days 01:00:00
120   2020-06-01 01:00:00    NaN    NaN    30 days 01:00:00
144   2020-07-01 01:00:00    NaN    NaN    29 days 01:00:00
168   2020-08-01 01:00:00   23.1   40.1    30 days 01:00:00
192   2020-09-01 01:00:00   67.9   77.0    30 days 01:00:00
216   2020-10-01 01:00:00   -0.3    6.6    29 days 01:00:00
240   2020-11-01 01:00:00   43.4   82.2    30 days 01:00:00
264   2020-12-01 01:00:00    9.2   28.0    29 days 01:00:00
288   2020-01-13 01:00:00   38.6   51.4 -324 days +01:00:00
744   2020-01-02 01:00:00    NaN    NaN  -30 days +01:00:00
768   2020-02-02 01:00:00    8.6   37.7    30 days 01:00:00
```

```
792   2020-03-02 01:00:00    3.7   20.7    28 days 01:00:00
816   2020-04-02 01:00:00    NaN    NaN    30 days 01:00:00
840   2020-05-02 01:00:00    NaN    NaN    29 days 01:00:00
864   2020-06-02 01:00:00   10.9   14.6    30 days 01:00:00
888   2020-07-02 01:00:00    1.7   14.3    29 days 01:00:00
912   2020-08-02 01:00:00   -4.3    3.8    30 days 01:00:00
936   2020-09-02 01:00:00   -3.9   14.9    30 days 01:00:00
960   2020-10-02 01:00:00   -0.7   12.2    29 days 01:00:00
984   2020-11-02 01:00:00    1.9    8.5    30 days 01:00:00
1008  2020-12-02 01:00:00    1.4    7.9    29 days 01:00:00
1032  2020-02-13 01:00:00    NaN    NaN -294 days +01:00:00
1440  2020-01-03 01:00:00    NaN    NaN  -58 days +01:00:00
1464  2020-02-03 01:00:00    NaN    NaN    30 days 01:00:00
1488  2020-03-03 01:00:00    NaN    NaN    28 days 01:00:00
1512  2020-04-03 01:00:00    4.8   10.8    30 days 01:00:00
1536  2020-05-03 01:00:00    3.8   17.4    29 days 01:00:00
1560  2020-06-03 01:00:00    4.1    8.9    30 days 01:00:00
1584  2020-07-03 01:00:00    2.9   14.4    29 days 01:00:00
1608  2020-08-03 01:00:00    5.2   11.7    30 days 01:00:00
1632  2020-09-03 01:00:00    2.2    5.4    30 days 01:00:00
1656  2020-10-03 01:00:00    2.3    7.4    29 days 01:00:00
1680  2020-11-03 01:00:00    3.9   13.4    30 days 01:00:00
1704  2020-12-03 01:00:00   -1.4    4.3    29 days 01:00:00
1728  2020-03-13 01:00:00    5.3   12.5 -266 days +01:00:00
2184  2020-01-04 01:00:00    5.8   14.8  -88 days +01:00:00
2208  2020-02-04 01:00:00    3.1    9.2    30 days 01:00:00
2232  2020-03-04 01:00:00    4.2    6.8    28 days 01:00:00
2256  2020-04-04 01:00:00    6.3    9.8    30 days 01:00:00
2280  2020-05-04 01:00:00   -2.0   -0.1    29 days 01:00:00
2304  2020-06-04 01:00:00    0.4    3.0    30 days 01:00:00
2328  2020-07-04 01:00:00    1.9    5.6    29 days 01:00:00
2352  2020-08-04 01:00:00   -3.3   -2.6    30 days 01:00:00
2376  2020-09-04 01:00:00   -2.0    4.2    30 days 01:00:00
```

```
 1 Australia_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Austral
 2
 3 Australia_PM_df = Australia_df[['DateTime','PM2.5','PM10']]
 4 Australia_PM_df['DateTime'] = pd.to_datetime(Australia_PM_df['DateTim
 5
 6 # Check for NaT values
 7 nat_present = Australia_PM_df['DateTime'].isna()
 8
 9 if nat_present.any():
10     print("NaT values found at index positions:")
11     print(nat_present[nat_present].index.tolist())
12 else:
13     print("No NaT values in the DateTime column.")
```

```
No NaT values in the DateTime column.
C:\Users\User\AppData\Local\Temp\ipykernel_8864\2916009012.py:4: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  Australia_PM_df['DateTime'] = pd.to_datetime(Australia_PM_df['DateTime']
```

```
 1 #Australia data cleaning
```

```
#Australia data cleaning
Australia_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Austral

Australia_PM_df = Australia_df[['DateTime','PM2.5','PM10']]
Australia_pp_df = preprocess_dataframe(Australia_PM_df)
Australia_or_df = apply_outliers_removal(Australia_pp_df, 'DateTime')

Australia_or_df3 = Australia_or_df.copy(deep=True) #Creating a comple

Australia_or_df3['DateTime'] = Australia_or_df3.index

Australia_or_df3.drop('residuals', axis=1, inplace=True)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 28745
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 30885
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
Australia_K_fill = KNN_fill_in_data(Australia_or_df3, 7)
Australia_K_fill.drop('DateTime', axis=1, inplace=True)
print(Australia_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\1136749358.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                         PM2.5        PM10
DateTime
2020-01-01 01:00:00   15.000000   43.500000
2020-01-01 02:00:00    6.400000   37.300000
2020-01-01 03:00:00    8.700000   37.000000
2020-01-01 04:00:00    8.800000   30.800000
2020-01-01 05:00:00   12.900000   35.700000
2020-01-01 06:00:00   14.500000   37.900000
2020-01-01 07:00:00   22.700000   45.700000
2020-01-01 08:00:00   22.500000   45.500000
2020-01-01 09:00:00   28.100000   56.300000
2020-01-01 10:00:00    8.400000   32.200000
2020-01-01 11:00:00   51.300000   82.700000
2020-01-01 12:00:00   86.800000  121.100000
2020-01-01 13:00:00   60.000000   79.200000
2020-01-01 14:00:00   74.500000  100.700000
2020-01-01 15:00:00    3.700000   22.500000
2020-01-01 16:00:00   32.700000   61.900000
```

```
2020-01-01 17:00:00   31.200000   55.500000
2020-01-01 18:00:00   36.100000   64.400000
2020-01-01 19:00:00   30.300000   65.900000
2020-01-01 20:00:00   23.563836   21.100000
2020-01-01 21:00:00   15.500000   39.600000
2020-01-01 22:00:00   10.000000   25.100000
2020-01-01 23:00:00   14.600000   33.700000
2020-01-02 00:00:00   21.000000   35.500000
2020-01-02 01:00:00   19.705430   36.646835
2020-01-02 02:00:00   20.017068   37.348439
2020-01-02 03:00:00   20.334125   37.838131
2020-01-02 04:00:00   20.596398   38.201776
2020-01-02 05:00:00   20.809777   38.483371
2020-01-02 06:00:00   17.088231   35.933152
2020-01-02 07:00:00   14.788878   32.282502
2020-01-02 08:00:00   14.654867   33.709768
2020-01-02 09:00:00   13.903879   32.423069
2020-01-02 10:00:00   13.862241   33.041111
2020-01-02 11:00:00   13.239883   32.953790
2020-01-02 12:00:00   11.919341   32.847382
2020-01-02 13:00:00   12.018798   32.919559
2020-01-02 14:00:00   12.165153   33.034577
2020-01-02 15:00:00   12.406932   33.244631
2020-01-02 16:00:00   12.912204   33.748106
2020-01-02 17:00:00   15.500000   37.000000
2020-01-02 18:00:00   13.600000   32.000000
2020-01-02 19:00:00   10.400000   32.700000
2020-01-02 20:00:00   10.400000   28.500000
2020-01-02 21:00:00   11.100000   30.300000
2020-01-02 22:00:00   10.500000   33.200000
2020-01-02 23:00:00    8.000000   34.200000
2020-01-03 00:00:00    8.700000   39.000000
2020-01-03 01:00:00    9.538199   34.889350
2020-01-03 02:00:00    9.768399   34.159670
2020-01-03 03:00:00    9.892505   33.826606
2020-01-03 04:00:00    9.972546   33.631402
2020-01-03 05:00:00   10.029163   33.502018
2020-01-03 06:00:00   10.071602   33.409601
```

```
 1 #China data cleaning
 2 China_df = pd.read_csv(r"C:\Users\User\Code\databook\data\China\China
 3
 4 China_PM_df = China_df[['DateTime','PM2.5','PM10']]
 5 China_pp_df = preprocess_dataframe(China_PM_df)
 6 China_or_df = apply_outliers_removal(China_pp_df, 'DateTime')
 7
 8 China_or_df3 = China_or_df.copy(deep=True) #Creating a completely sep
 9
10 China_or_df3['DateTime'] = China_or_df3.index
11
12 China_or_df3.drop('residuals', axis=1, inplace=True)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
```

```
The length of the PM2.5 column is 35040
Data being checked: PM2.5
                     PM2.5  stl_anomaly
DateTime
2017-01-02 14:00:00   79.0         True
2017-01-02 15:00:00   73.0         True
2017-01-02 16:00:00   65.0         True
2017-01-03 17:00:00   59.0         True
2017-01-27 10:00:00  143.0         True
2017-02-06 07:00:00  164.0         True
2017-02-09 22:00:00  132.0         True
2017-02-09 23:00:00  131.0         True
2017-02-17 06:00:00  149.0         True
2017-02-17 07:00:00  152.0         True
2017-02-17 08:00:00  131.0         True
2017-02-17 09:00:00  130.0         True
2017-02-17 10:00:00  123.0         True
2017-03-10 22:00:00  157.0         True
2017-03-10 23:00:00  170.0         True
2017-03-11 00:00:00  160.0         True
2017-03-18 15:00:00  106.0         True
2017-03-21 21:00:00  118.0         True
2017-03-21 22:00:00  117.0         True
2017-03-21 23:00:00  126.0         True
2017-04-08 18:00:00  143.0         True
2017-10-29 09:00:00  129.0         True
2017-10-29 10:00:00  114.0         True
2017-10-29 11:00:00  118.0         True
2017-11-03 13:00:00  126.0         True
2017-11-03 14:00:00  151.0         True
2017-11-03 15:00:00  157.0         True
2017-11-03 16:00:00  128.0         True
2017-11-26 19:00:00  135.0         True
2017-11-26 20:00:00  128.0         True
2017-11-29 03:00:00  130.0         True
2017-11-29 04:00:00  119.0         True
2017-11-29 05:00:00  150.0         True
2017-11-29 06:00:00  109.0         True
2017-12-04 08:00:00  161.0         True
2017-12-04 09:00:00  166.0         True
2017-12-04 10:00:00  171.0         True
2017-12-04 11:00:00  174.0         True
2017-12-04 12:00:00  145.0         True
2017-12-04 13:00:00  136.0         True
2017-12-04 14:00:00  123.0         True
2017-12-04 16:00:00  137.0         True
2017-12-07 23:00:00  133.0         True
2017-12-08 00:00:00  139.0         True
2017-12-08 01:00:00  147.0         True
2017-12-08 02:00:00  136.0         True
2017-12-08 03:00:00  128.0         True
2017-12-16 11:00:00  160.0         True
```

```
1 China_K_fill = KNN_fill_in_data(China_or_df3, 26)
2 China_K_fill.drop('DateTime', axis=1, inplace=True)
3 print(China_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\3676177356.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
```

```
df_imputed = df_imputed.applymap(lambda x: max(x, 0))
```

DateTime	PM2.5	PM10
2017-01-02 00:00:00	164.000000	282.000000
2017-01-02 01:00:00	164.000000	282.000000
2017-01-02 02:00:00	164.000000	282.000000
2017-01-02 03:00:00	164.000000	282.000000
2017-01-02 04:00:00	160.000000	279.000000
2017-01-02 05:00:00	156.000000	276.000000
2017-01-02 06:00:00	156.000000	263.000000
2017-01-02 07:00:00	136.000000	232.000000
2017-01-02 08:00:00	126.000000	196.000000
2017-01-02 09:00:00	124.000000	202.000000
2017-01-02 10:00:00	135.000000	224.000000
2017-01-02 11:00:00	143.000000	245.000000
2017-01-02 12:00:00	148.000000	237.000000
2017-01-02 13:00:00	90.000000	141.000000
2017-01-02 14:00:00	132.061968	121.000000
2017-01-02 15:00:00	135.267405	123.000000
2017-01-02 16:00:00	133.500895	194.604084
2017-01-02 17:00:00	89.000000	138.000000
2017-01-02 18:00:00	168.000000	252.000000
2017-01-02 19:00:00	163.000000	257.000000
2017-01-02 20:00:00	164.000000	268.000000
2017-01-02 21:00:00	156.000000	266.000000
2017-01-02 22:00:00	155.000000	288.000000
2017-01-02 23:00:00	138.000000	266.000000
2017-01-03 00:00:00	127.000000	213.000000
2017-01-03 01:00:00	127.000000	218.000000
2017-01-03 02:00:00	137.000000	229.000000
2017-01-03 03:00:00	132.000000	218.000000
2017-01-03 04:00:00	139.000000	225.000000
2017-01-03 05:00:00	147.000000	234.000000
2017-01-03 06:00:00	142.000000	237.000000
2017-01-03 07:00:00	141.000000	233.000000
2017-01-03 08:00:00	148.000000	247.000000
2017-01-03 09:00:00	164.000000	259.000000
2017-01-03 10:00:00	168.000000	300.000000
2017-01-03 11:00:00	194.000000	289.000000
2017-01-03 12:00:00	180.000000	285.000000
2017-01-03 13:00:00	107.000000	194.000000
2017-01-03 14:00:00	110.000000	181.000000
2017-01-03 15:00:00	104.000000	179.000000
2017-01-03 16:00:00	84.000000	149.000000
2017-01-03 17:00:00	130.231167	220.700350
2017-01-03 18:00:00	89.000000	164.000000
2017-01-03 19:00:00	112.000000	215.000000
2017-01-03 20:00:00	154.000000	263.000000
2017-01-03 21:00:00	178.000000	292.000000
2017-01-03 22:00:00	204.000000	330.000000
2017-01-03 23:00:00	208.000000	338.000000
2017-01-04 00:00:00	195.000000	322.000000
2017-01-04 01:00:00	194.000000	309.000000
2017-01-04 02:00:00	180.000000	292.000000
2017-01-04 03:00:00	193.000000	294.000000
2017-01-04 04:00:00	190.000000	284.000000
2017-01-04 05:00:00	154.000000	255.000000

```
#India data cleaning
India_df = pd.read_csv(r"C:\Users\User\Code\databook\data\India\India
```

```
India_df = pd.read_csv(r'C:\Users\User\Code\databook\data\India\India

India_PM_df = India_df[['DateTime','PM2.5','PM10']]
India_pp_df = preprocess_dataframe(India_PM_df)
India_or_df = apply_outliers_removal(India_pp_df, 'DateTime')

India_or_df3 = India_or_df.copy(deep=True) #Creating a completely sep

India_or_df3['DateTime'] = India_or_df3.index

India_or_df3.drop('residuals', axis=1, inplace=True)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 47817
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 45771
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
India_K_fill = KNN_fill_in_data(India_or_df3, 26)
India_K_fill.drop('DateTime', axis=1, inplace=True)
print(India_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\3676177356.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                        PM2.5         PM10
DateTime
2015-01-01 01:00:00  454.580000   935.180000
2015-01-01 02:00:00  440.440000   753.677272
2015-01-01 03:00:00  409.090000   731.805253
2015-01-01 04:00:00  436.120000   751.168685
2015-01-01 05:00:00  415.880000   976.990000
2015-01-01 06:00:00  384.160000   862.230000
2015-01-01 07:00:00  344.440000   731.830000
2015-01-01 08:00:00  337.980000   725.740000
2015-01-01 09:00:00  285.220000   656.930000
2015-01-01 10:00:00  258.420000   525.540000
2015-01-01 11:00:00  265.620000   548.820000
2015-01-01 12:00:00  260.090000   522.930000
2015-01-01 13:00:00  222.010000   408.620000
2015-01-01 14:00:00  202.140000   340.280000
2015-01-01 15:00:00  244.140000   459.010000
2015-01-01 16:00:00  211.510000   340.660000
2015-01-01 17:00:00  191.120000   257.400000
2015-01-01 18:00:00  218.440000   351.010000
2015-01-01 19:00:00  296.800000   600.950000
2015-01-01 20:00:00  336.430000   714.630000
2015-01-01 21:00:00  326.640000   691.840000
```

```
2015-01-01 21:00:00  320.040000   691.840000
2015-01-01 22:00:00  327.680000   732.590000
2015-01-01 23:00:00  335.180000   776.360000
2015-01-02 00:00:00  358.680000   860.320000
2015-01-02 01:00:00  444.440000   806.070000
2015-01-02 02:00:00  464.640000   856.680000
2015-01-02 03:00:00  344.050000   765.000000
2015-01-02 04:00:00  358.070000   797.090000
2015-01-02 05:00:00  271.410000   516.140000
2015-01-02 06:00:00  260.950000   467.810000
2015-01-02 07:00:00  192.950000   301.480000
2015-01-02 08:00:00   73.860000   115.400000
2015-01-02 09:00:00   52.540000    62.600000
2015-01-02 10:00:00   70.400000    41.870000
2015-01-02 11:00:00   96.840000    65.110000
2015-01-02 12:00:00   93.350000    36.250000
2015-01-02 13:00:00  208.770000    28.970000
2015-01-02 14:00:00  239.860000    52.680000
2015-01-02 15:00:00  187.960000    49.890000
2015-01-02 16:00:00  185.200000    64.700000
2015-01-02 17:00:00  157.230000    60.090000
2015-01-02 18:00:00  137.380000    99.630000
2015-01-02 19:00:00   50.680000    79.180000
2015-01-02 20:00:00   59.150000    92.420000
2015-01-02 21:00:00   54.250000    84.770000
2015-01-02 22:00:00   55.810000    87.200000
2015-01-02 23:00:00   49.810000    77.820000
2015-01-03 00:00:00   51.210000    80.010000
2015-01-03 01:00:00   50.690000    79.210000
2015-01-03 02:00:00   41.060000    64.160000
2015-01-03 03:00:00   45.080000    60.410000
2015-01-03 04:00:00   44.780000    58.670000
2015-01-03 05:00:00   51.440000    77.920000
2015-01-03 06:00:00   65.000000   121.090000
```

```
#Africa data cleaning
Africa_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Africa\Pro

Africa_PM_df = Africa_df[['DateTime','PM2.5','PM10']]
Africa_pp_df = preprocess_dataframe(Africa_PM_df)
Africa_or_df = apply_outliers_removal(Africa_pp_df, 'DateTime')

Africa_or_df3 = Africa_or_df.copy(deep=True) #Creating a completely s

Africa_or_df3['DateTime'] = Africa_or_df3.index

Africa_or_df3.drop('residuals', axis=1, inplace=True)
```

```
The length of the PM2.5 column is 9408
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 9408
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
1 Africa_K_fill = KNN_fill_in_data(Africa_or_df3, 26)
2 Africa_K_fill.drop('DateTime', axis=1, inplace=True)
3 print(Africa_K_fill)
```

```
                        PM2.5        PM10
DateTime
2019-11-21 08:00:00   27.210000   35.410000
2019-11-21 09:00:00   15.660000   22.920000
2019-11-21 10:00:00   12.200000   13.110000
2019-11-21 11:00:00   10.930000   11.060000
2019-11-21 12:00:00    9.610000   10.500000
2019-11-21 13:00:00   10.940000   11.190000
2019-11-21 14:00:00   13.800000   17.300000
2019-11-21 15:00:00   17.610000   21.110000
2019-11-21 16:00:00   33.100000   40.210000
2019-11-21 17:00:00   32.150000   40.230000
2019-11-21 18:00:00   37.290000   43.080000
2019-11-21 19:00:00   34.370000   46.300000
2019-11-21 20:00:00   33.970000   43.660000
2019-11-21 21:00:00   30.620000   40.910000
2019-11-21 22:00:00   20.900000   33.650000
2019-11-21 23:00:00   14.520000   17.850000
2019-11-22 00:00:00   11.970000   13.790000
2019-11-22 01:00:00   13.970000   18.930000
2019-11-22 02:00:00   17.240000   24.140000
2019-11-22 03:00:00   15.860000   21.010000
2019-11-22 04:00:00   10.080000    8.540000
2019-11-22 05:00:00   12.020000   13.560000
2019-11-22 06:00:00   12.330000   13.030000
2019-11-22 07:00:00   21.030000   31.580000
2019-11-22 08:00:00   20.360000   30.660000
2019-11-22 09:00:00   21.860000   27.950000
2019-11-22 10:00:00   20.840000   27.870000
2019-11-22 11:00:00   10.940000    9.540000
2019-11-22 12:00:00   12.380000   16.560000
2019-11-22 13:00:00   14.360000   24.350000
2019-11-22 14:00:00   15.100000   23.320000
2019-11-22 15:00:00   22.050000   34.840000
2019-11-22 16:00:00   35.050000   47.910000
2019-11-22 17:00:00   28.800000   41.650000
2019-11-22 18:00:00   40.030000   50.930000
2019-11-22 19:00:00   21.040000   34.170000
2019-11-22 20:00:00    8.310000    4.060000
2019-11-22 21:00:00    8.920000    3.730000
2019-11-22 22:00:00   15.180659   17.087233
2019-11-22 23:00:00   10.810000    4.630000
2019-11-23 00:00:00   14.927399   17.128448
2019-11-23 01:00:00   14.717289   17.345526
2019-11-23 02:00:00   10.520000    5.990000
2019-11-23 03:00:00   15.390000   22.770000
2019-11-23 04:00:00   21.510000   35.260000
2019-11-23 05:00:00   10.980000   12.290000
2019-11-23 06:00:00   20.670000   32.130000
2019-11-23 07:00:00   16.420000   27.390000
2019-11-23 08:00:00   13.890000   19.790000
2019-11-23 09:00:00   10.910000   14.720000
2019-11-23 10:00:00    8.340000    9.900000
```

```
2019-11-23 11:00:00    8.690000    9.470000
2019-11-23 12:00:00   11.360000    9.180000
2019-11-23 13:00:00   10.630000    6.680000
2019-11-23 14:00:00   10.210000    9.240000
2019-11-23 15:00:00    9.970000    7.230000
```

```
#Mexico data cleaning
Mexico_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Mexico\Pro

Mexico_PM_df = Mexico_df[['DateTime','PM2.5','PM10']]
Mexico_pp_df = preprocess_dataframe(Mexico_PM_df)
Mexico_or_df = apply_outliers_removal(Mexico_pp_df, 'DateTime')

Mexico_or_df3 = Mexico_or_df.copy(deep=True) #Creating a completely s

Mexico_or_df3['DateTime'] = Mexico_or_df3.index

Mexico_or_df3.drop('residuals', axis=1, inplace=True)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 42980
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 43273
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
Mexico_K_fill = KNN_fill_in_data(Mexico_or_df3, 70)
Mexico_K_fill.drop('DateTime', axis=1, inplace=True)
print(Mexico_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\3676177356.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                          PM2.5        PM10
DateTime
2016-07-28 00:00:00    39.226000   59.960000
2016-07-28 01:00:00    59.496000   83.775000
2016-07-28 02:00:00    63.474000   90.966000
2016-07-28 03:00:00    46.661000   65.046000
2016-07-28 04:00:00    46.951000   78.580000
2016-07-28 05:00:00    37.223000   56.641000
2016-07-28 06:00:00    33.868000   54.605000
2016-07-28 07:00:00    35.657000   68.004000
2016-07-28 08:00:00    41.030000   79.380000
2016-07-28 09:00:00    43.069000  104.920000
2016-07-28 10:00:00    33.582000   76.111000
2016-07-28 11:00:00    34.113000   80.444000
2016-07-28 12:00:00    24.415000   52.781000
```

```
2016-07-28 13:00:00    19.982000   39.818000
2016-07-28 14:00:00    16.775000   39.510000
2016-07-28 15:00:00    22.981000   40.704000
2016-07-28 16:00:00    14.733000   35.359000
2016-07-28 17:00:00    17.797000   40.007000
2016-07-28 18:00:00    18.980000   43.497000
2016-07-28 19:00:00    25.034000   48.906000
2016-07-28 20:00:00    21.455000   39.172000
2016-07-28 21:00:00    16.398000   23.098000
2016-07-28 22:00:00    12.556000   28.059000
2016-07-28 23:00:00    16.372000   34.942000
2016-07-29 00:00:00    21.663000   35.453000
2016-07-29 01:00:00    13.123000   19.181000
2016-07-29 02:00:00    27.357000   38.772000
2016-07-29 03:00:00    52.205000   82.540000
2016-07-29 04:00:00    44.777000   81.392000
2016-07-29 05:00:00    42.552000   75.401000
2016-07-29 06:00:00    22.865000   33.700000
2016-07-29 07:00:00    27.923000   50.136000
2016-07-29 08:00:00    25.200000   40.539000
2016-07-29 09:00:00    25.766000   36.071000
2016-07-29 10:00:00    26.537000   47.159000
2016-07-29 11:00:00    21.228000   45.771000
2016-07-29 12:00:00    19.071000   37.639000
2016-07-29 13:00:00     8.050900   20.051000
2016-07-29 14:00:00     8.903300   20.880000
2016-07-29 15:00:00    10.428000   17.959000
2016-07-29 16:00:00    17.923000   47.622000
2016-07-29 17:00:00    15.975000   43.998000
2016-07-29 18:00:00    21.151000   42.334000
2016-07-29 19:00:00    17.085000   38.943000
2016-07-29 20:00:00    18.177000   33.995000
2016-07-29 21:00:00    15.981000   26.554000
2016-07-29 22:00:00    12.288000   25.812000
2016-07-29 23:00:00    11.446000   17.442000
2016-07-30 00:00:00    24.034000   35.559000
2016-07-30 01:00:00    12.076000   15.088000
2016-07-30 02:00:00    17.916000   29.970000
2016-07-30 03:00:00     7.823600   18.774000
2016-07-30 04:00:00    16.778000   34.918000
2016-07-30 05:00:00    16.165000   33.615000
```

```
 1 #Korea data cleaning
 2 Korea_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Korea\Korea
 3
 4 Korea_PM_df = Korea_df[['DateTime','PM2.5','PM10']]
 5 Korea_pp_df = preprocess_dataframe(Korea_PM_df)
 6 Korea_or_df = apply_outliers_removal(Korea_pp_df, 'DateTime')
 7
 8 Korea_or_df3 = Korea_or_df.copy(deep=True) #Creating a completely sep
 9
10 Korea_or_df3['DateTime'] = Korea_or_df3.index
11
12 Korea_or_df3.drop('residuals', axis=1, inplace=True)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\110586271.py:5: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row indexer,col indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
  df['DateTime'] = pd.to_datetime(df['DateTime'])
The length of the PM2.5 column is 25877
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 25882
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
1 Korea_K_fill = KNN_fill_in_data(Korea_or_df3, 26)
2 Korea_K_fill.drop('DateTime', axis=1, inplace=True)
3 print(Korea_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_8864\3676177356.py:24: FutureWa
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                         PM2.5        PM10
DateTime
2017-01-01 00:00:00   53.000000   68.000000
2017-01-01 01:00:00   63.000000   78.000000
2017-01-01 02:00:00   56.000000   72.000000
2017-01-01 03:00:00   46.000000   68.000000
2017-01-01 04:00:00   48.000000   66.000000
2017-01-01 05:00:00   44.000000   60.000000
2017-01-01 06:00:00   38.000000   67.000000
2017-01-01 07:00:00   44.000000   77.000000
2017-01-01 08:00:00   40.000000   73.000000
2017-01-01 09:00:00   42.000000   65.000000
2017-01-01 10:00:00   64.000000   67.000000
2017-01-01 11:00:00   60.000000   70.000000
2017-01-01 12:00:00   57.000000   66.000000
2017-01-01 13:00:00   53.000000   95.000000
2017-01-01 14:00:00   54.000000   88.000000
2017-01-01 15:00:00   46.000000   81.000000
2017-01-01 16:00:00   54.000000   87.000000
2017-01-01 17:00:00   48.000000   86.000000
2017-01-01 18:00:00   77.000000  106.000000
2017-01-01 19:00:00   75.000000   97.000000
2017-01-01 20:00:00   62.000000   91.000000
2017-01-01 21:00:00   57.000000   86.000000
2017-01-01 22:00:00   61.000000   90.000000
2017-01-01 23:00:00   59.000000  106.000000
2017-01-02 00:00:00   66.000000  101.000000
2017-01-02 01:00:00   74.000000  102.000000
2017-01-02 02:00:00   66.000000   95.000000
2017-01-02 03:00:00   62.000000   82.000000
2017-01-02 04:00:00   58.000000   86.000000
2017-01-02 05:00:00   58.000000   85.000000
2017-01-02 06:00:00   51.000000  102.000000
2017-01-02 07:00:00   72.000000   92.000000
2017-01-02 08:00:00   79.000000   93.000000
2017-01-02 09:00:00   56.000000   77.000000
2017-01-02 10:00:00   57.000000   97.000000
2017-01-02 11:00:00   59.000000  107.000000
```

```
2017-01-02 12:00:00   52.000000   90.000000
2017-01-02 13:00:00   48.000000   90.000000
2017-01-02 14:00:00   81.000000  117.000000
2017-01-02 15:00:00  120.000000  152.000000
2017-01-02 16:00:00  116.000000  159.000000
2017-01-02 17:00:00  121.000000  170.000000
2017-01-02 18:00:00  126.000000  183.000000
2017-01-02 19:00:00  131.000000  169.000000
2017-01-02 20:00:00  126.000000  192.000000
2017-01-02 21:00:00  111.000000  175.000000
2017-01-02 22:00:00  133.000000  159.000000
2017-01-02 23:00:00  111.729741  156.831689
2017-01-03 00:00:00  113.000000  155.000000
2017-01-03 01:00:00  127.000000  178.000000
2017-01-03 02:00:00  121.000000  170.000000
2017-01-03 03:00:00   92.000000  159.000000
2017-01-03 04:00:00   84.000000  114.000000
2017-01-03 05:00:00   87.000000  118.000000
```

```
#Spain data cleaning
Spain_df = pd.read_csv(r"C:\Users\User\Code\databook\data\Spain\Spain

Spain_PM_df = Spain_df[['DateTime','PM2.5','PM10']]
Spain_pp_df = preprocess_dataframe(Spain_PM_df)
Spain_or_df = apply_outliers_removal(Spain_pp_df, 'DateTime')

Spain_or_df3 = Spain_or_df.copy(deep=True) #Creating a completely sep

Spain_or_df3['DateTime'] = Spain_or_df3.index

Spain_or_df3.drop('residuals', axis=1, inplace=True)
```

```
The length of the PM2.5 column is 58691
Data being checked: PM2.5
Empty DataFrame
Columns: [PM2.5, stl_anomaly]
Index: []
The length of the PM10 column is 59024
Data being checked: PM10
Empty DataFrame
Columns: [PM10, stl_anomaly]
Index: []
```

```
Spain_K_fill = KNN_fill_in_data(Spain_or_df3, 26)
Spain_K_fill.drop('DateTime', axis=1, inplace=True)
print(Spain_K_fill)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_10772\3676177356.py:24: FutureW
  df_imputed = df_imputed.applymap(lambda x: max(x, 0))
                         PM2.5        PM10
DateTime
2015-01-01 00:00:00   45.000000   58.000000
2015-01-01 01:00:00   83.000000   99.000000
2015-01-01 02:00:00   72.000000   89.000000
2015-01-01 03:00:00   69.000000   92.000000
2015-01-01 04:00:00   78.000000   99.000000
2015-01-01 05:00:00   59.000000   79.000000
```

```
2015-01-01 06:00:00   49.000000   57.000000
2015-01-01 07:00:00   45.000000   52.000000
2015-01-01 08:00:00   38.000000   44.000000
2015-01-01 09:00:00   37.000000   44.000000
2015-01-01 10:00:00   45.000000   52.000000
2015-01-01 11:00:00   18.000000   19.000000
2015-01-01 12:00:00   12.000000   11.000000
2015-01-01 13:00:00    6.000000    4.000000
2015-01-01 14:00:00   16.000000   21.000000
2015-01-01 15:00:00   12.000000   15.000000
2015-01-01 16:00:00    9.000000   13.000000
2015-01-01 17:00:00   10.000000   19.000000
2015-01-01 18:00:00   10.000000   23.000000
2015-01-01 19:00:00   16.000000   25.000000
2015-01-01 20:00:00   19.000000   27.000000
2015-01-01 21:00:00   21.000000   29.000000
2015-01-01 22:00:00   17.000000   23.000000
2015-01-01 23:00:00   15.000000   21.000000
2015-01-02 00:00:00   17.000000   23.000000
2015-01-02 01:00:00   26.000000   42.000000
2015-01-02 02:00:00   17.000000   28.000000
2015-01-02 03:00:00    9.000000   11.000000
2015-01-02 04:00:00    7.000000   10.000000
2015-01-02 05:00:00    8.000000   14.000000
2015-01-02 06:00:00   11.000000   17.000000
2015-01-02 07:00:00   15.000000   21.000000
2015-01-02 08:00:00   18.000000   27.000000
2015-01-02 09:00:00   18.000000   34.000000
2015-01-02 10:00:00   13.000000   15.000000
2015-01-02 11:00:00   17.000000   22.000000
2015-01-02 12:00:00   19.403537   51.000000
2015-01-02 13:00:00   20.000000   33.000000
2015-01-02 14:00:00   17.000000   25.000000
2015-01-02 15:00:00   22.000000   32.000000
2015-01-02 16:00:00   22.000000   36.000000
2015-01-02 17:00:00   23.000000   44.000000
2015-01-02 18:00:00   26.000000   49.000000
2015-01-02 19:00:00   26.000000   48.000000
2015-01-02 20:00:00   31.000000   55.000000
2015-01-02 21:00:00   36.000000   61.000000
2015-01-02 22:00:00   53.000000   83.000000
2015-01-02 23:00:00   56.000000   78.000000
2015-01-03 00:00:00   22.000000   33.000000
2015-01-03 01:00:00   11.000000   17.000000
2015-01-03 02:00:00    7.000000   12.000000
2015-01-03 03:00:00   12.000000   22.000000
2015-01-03 04:00:00   10.000000   21.000000
2015-01-03 05:00:00    9.000000   16.000000
```

```
#Save PM Data of Spain data
Spain_K_fill.to_csv('C:/Users/User/Code/databook/data/Spain/SpainClea
```

```
Austrailia_PM10_Original_Mean = Australia_or_df['PM10'].mean()
print(f"PM10 Original Mean = {Austrailia_PM10_Original_Mean}")
Australia_PM10_Imp_Mean = Australia_K_fill['PM10'].mean()
print(f"PM10 Imputed Mean = {Australia_PM10_Imp_Mean}")
print(f"% Change in Mean = {(abs(Austrailia_PM10_Original_Mean-Austra
```

```
 6
 7 Austrailia_PM10_Original_STD = Australia_or_df['PM10'].std()
 8 print(f"PM10 Original STD = {Austrailia_PM10_Original_STD}")
 9 Australia_PM10_Imp_STD = Australia_K_fill['PM10'].std()
10 print(f"PM10 Imputed STD = {Australia_PM10_Imp_STD}")
11 print(f"% Change in STD = {(abs(Austrailia_PM10_Original_STD-Australi
12
13 Austrailia_PM25_Original_Mean = Australia_or_df['PM2.5'].mean()
14 print(f"PM25 Original Mean = {Austrailia_PM25_Original_Mean}")
15 Australia_PM25_Imp_Mean = Australia_K_fill['PM2.5'].mean()
16 print(f"PM2.5 Imputed Mean = {Australia_PM25_Imp_Mean}")
17 print(f"% Change in Mean = {(abs(Austrailia_PM25_Original_Mean-Austra
18
19 Austrailia_PM25_Original_STD = Australia_or_df['PM2.5'].std()
20 print(f"PM2.5 Original STD = {Austrailia_PM25_Original_STD}")
21 Australia_PM25_Imp_STD = Australia_K_fill['PM2.5'].std()
22 print(f"PM2.5 Imputed STD = {Australia_PM25_Imp_STD}")
23 print(f"% Change in STD = {(abs(Austrailia_PM25_Original_STD-Australi
24
```

```
PM10 Original Mean = 13.914885790059028
PM10 Imputed Mean = 15.49911605176426
% Change in Mean = 11.385147428497227
PM10 Original STD = 11.809938034057852
PM10 Imputed STD = 12.019752696767737
% Change in STD = 1.7765941032443617
PM25 Original Mean = 7.3767304315682445
PM2.5 Imputed Mean = 7.835602459208763
% Change in Mean = 6.220534041433931
PM2.5 Original STD = 9.779175130886165
PM2.5 Imputed STD = 9.700285300116004
% Change in STD = 0.8067125265094994
```

```
 1 China_PM10_Original_Mean = China_or_df['PM10'].mean()
 2 print(f"PM10 Original Mean = {China_PM10_Original_Mean}")
 3 China_PM10_Imp_Mean = China_K_fill['PM10'].mean()
 4 print(f"PM10 Imputed Mean = {China_PM10_Imp_Mean}")
 5 print(f"% Change in Mean = {(abs(China_PM10_Original_Mean-China_PM10_
 6 print("")
 7 China_PM10_Original_STD = China_or_df['PM10'].std()
 8 print(f"PM10 Original STD = {China_PM10_Original_STD}")
 9 China_PM10_Imp_STD = China_K_fill['PM10'].std()
10 print(f"PM10 Imputed STD = {China_PM10_Imp_STD}")
11 print(f"% Change in STD = {(abs(China_PM10_Original_STD-China_PM10_Im
12 print("")
13 China_PM25_Original_Mean = China_or_df['PM2.5'].mean()
14 print(f"PM25 Original Mean = {China_PM25_Original_Mean}")
15 China_PM25_Imp_Mean = China_K_fill['PM2.5'].mean()
16 print(f"PM2.5 Imputed Mean = {China_PM25_Imp_Mean}")
17 print(f"% Change in Mean = {(abs(China_PM25_Original_Mean-China_PM25_
18 print("")
19 China_PM25_Original_STD = China_or_df['PM2.5'].std()
20 print(f"PM2.5 Original STD = {China_PM25_Original_STD}")
21 China_PM25_Imp_STD = China_K_fill['PM2.5'].std()
22 print(f"PM2.5 Imputed STD = {China_PM25_Imp_STD}")
```

```
print(f"PM10 Imputed STD = {China_PM25_Imp_STD}")
print(f"% Change in STD = {(abs(China_PM25_Original_STD-China_PM25_Im
```

```
PM10 Original Mean = 76.79392950916481
PM10 Imputed Mean = 77.03470848032912
% Change in Mean = 0.31353906839155576

PM10 Original STD = 49.085042412905956
PM10 Imputed STD = 49.47699527566208
% Change in STD = 0.7985179262126181

PM25 Original Mean = 41.73384303119871
PM2.5 Imputed Mean = 41.92130506278232
% Change in Mean = 0.44918468554038277

PM2.5 Original STD = 30.38425585313831
PM2.5 Imputed STD = 30.73168817805868
% Change in STD = 1.1434616881837634
```

```
India_PM10_Original_Mean = India_or_df['PM10'].mean()
print(f"PM10 Original Mean = {India_PM10_Original_Mean}")
India_PM10_Imp_Mean = India_K_fill['PM10'].mean()
print(f"PM10 Imputed Mean = {India_PM10_Imp_Mean}")
print(f"% Change in Mean = {(abs(India_PM10_Original_Mean-India_PM10_
print("")
India_PM10_Original_STD = India_or_df['PM10'].std()
print(f"PM10 Original STD = {India_PM10_Original_STD}")
India_PM10_Imp_STD = India_K_fill['PM10'].std()
print(f"PM10 Imputed STD = {India_PM10_Imp_STD}")
print(f"% Change in STD = {(abs(India_PM10_Original_STD-India_PM10_Im
print("")
India_PM25_Original_Mean = India_or_df['PM2.5'].mean()
print(f"PM25 Original Mean = {India_PM25_Original_Mean}")
India_PM25_Imp_Mean = India_K_fill['PM2.5'].mean()
print(f"PM2.5 Imputed Mean = {India_PM25_Imp_Mean}")
print(f"% Change in Mean = {(abs(India_PM25_Original_Mean-India_PM25_
print("")
India_PM25_Original_STD = India_or_df['PM2.5'].std()
print(f"PM2.5 Original STD = {India_PM25_Original_STD}")
India_PM25_Imp_STD = India_K_fill['PM2.5'].std()
print(f"PM2.5 Imputed STD = {India_PM25_Imp_STD}")
print(f"% Change in STD = {(abs(India_PM25_Original_STD-India_PM25_Im
```

```
PM10 Original Mean = 241.20379937027016
PM10 Imputed Mean = 297.56168391489837
% Change in Mean = 23.365255726388305

PM10 Original STD = 149.41756469247497
PM10 Imputed STD = 134.05496529556518
% Change in STD = 10.281655592853793

PM25 Original Mean = 123.15807840447422
PM2.5 Imputed Mean = 171.5571606221127
% Change in Mean = 39.29834148490593
```

```
PM2.5 Original STD = 104.2813505783325
PM2.5 Imputed STD = 94.0248789940435
% Change in STD = 9.835384301610766
```

```
#print(US_fill_df)
#Save All US data
US_or_df.to_csv('C:/Users/User/Code/databook/data/US/CombinedData/USA
US_fill_df.to_csv('C:/Users/User/Code/databook/data/US/CombinedData/U

```

```
#Save PM Data of US data
US_PM_df = US_fill_df[['PM2.5','PM10']]
US_PM_df.to_csv('C:/Users/User/Code/databook/data/US/CombinedData/USC
print(US_PM_df)
```

```
                        PM2.5         PM10
DateTime
2013-01-01 00:00:00   12.000000    16.000000
2013-01-01 01:00:00   16.000000    18.000000
2013-01-01 02:00:00    9.000000    10.000000
2013-01-01 03:00:00    8.000000    10.000000
2013-01-01 04:00:00   10.000000     7.000000
2013-01-01 05:00:00   13.000000    12.000000
2013-01-01 06:00:00   12.000000     9.000000
2013-01-01 07:00:00    9.900000     6.000000
2013-01-01 08:00:00    6.000000     6.000000
2013-01-01 09:00:00    6.000000     7.000000
2013-01-01 10:00:00    9.800000     9.000000
2013-01-01 11:00:00    7.500000     8.000000
2013-01-01 12:00:00    6.900000     3.000000
2013-01-01 13:00:00   15.800000     3.000000
2013-01-01 14:00:00   13.900000     1.000000
2013-01-01 15:00:00   16.900000     7.000000
2013-01-01 16:00:00   21.600000     5.000000
2013-01-01 17:00:00   26.400000    17.000000
2013-01-01 18:00:00   31.900000     7.000000
2013-01-01 19:00:00   30.900000    16.000000
2013-01-01 20:00:00   30.900000    16.000000
2013-01-01 21:00:00   32.900000     9.000000
2013-01-01 22:00:00   25.100000    13.000000
2013-01-01 23:00:00   22.100000    10.000000
2013-01-02 00:00:00   17.000000    10.000000
2013-01-02 01:00:00   15.000000     7.000000
2013-01-02 02:00:00   25.900000     5.000000
2013-01-02 03:00:00   10.000000    16.000000
2013-01-02 04:00:00   31.600000     3.000000
2013-01-02 05:00:00   30.900000    15.000000
2013-01-02 06:00:00   25.900000    18.000000
2013-01-02 07:00:00   14.000000    13.000000
2013-01-02 08:00:00   21.000000    18.000000
2013-01-02 09:00:00   26.500000    18.000000
2013-01-02 10:00:00   16.000000    17.000000
2013-01-02 11:00:00   16.700000    11.000000
2013-01-02 12:00:00    9.000000    10.000000
2013-01-02 13:00:00   11.000000    17.000000
2013-01-02 14:00:00   12.000000    10.000000
2013-01-02 15:00:00   13.000000     9.000000
```

```
2013-01-02 15:00:00   12.000000     9.000000
2013-01-02 16:00:00   13.000000    10.000000
2013-01-02 17:00:00   18.700000    15.000000
2013-01-02 18:00:00   17.700000    14.000000
2013-01-02 19:00:00   12.000000    13.000000
2013-01-02 20:00:00   12.000000    19.000000
2013-01-02 21:00:00   15.700000    13.000000
2013-01-02 22:00:00   18.900000    16.000000
2013-01-02 23:00:00   13.200000    15.000000
2013-01-03 00:00:00   13.000000    18.000000
2013-01-03 01:00:00   13.000000    13.000000
2013-01-03 02:00:00   20.000000    14.000000
2013-01-03 03:00:00   14.000000    14.000000
2013-01-03 04:00:00   14.000000    16.000000
2013-01-03 05:00:00   13.000000    24.000000
2013-01-03 06:00:00   26.400000    25.000000
2013-01-03 07:00:00   32.000000    28.000000
```

```
1 #Save PM Data of US data
2 US_K_fill.to_csv('C:/Users/User/Code/databook/data/US/CombinedData/US
```

```
1 #Save PM Data of UK data
2 UK_K_fill.to_csv('C:/Users/User/Code/databook/data/UK/AllData/UKClean
```

```
1 #Save PM Data of Columbia data
2 Columbia_K_fill.to_csv('C:/Users/User/Code/databook/data/Columbia/All
```

```
1 #Save PM Data of Australia data
2 Australia_K_fill.to_csv('C:/Users/User/Code/databook/data/Australia/A
```

```
1 #Save PM Data of China data
2 China_K_fill.to_csv('C:/Users/User/Code/databook/data/China/ChinaClea
```

```
1 #Save PM Data of India data
2 India_K_fill.to_csv('C:/Users/User/Code/databook/data/India/IndiaClea
```

```
1 #Save PM Data of Africa data
2 Africa_K_fill.to_csv('C:/Users/User/Code/databook/data/Africa/AfricaC
```

```
1 #Save PM Data of Mexico data
2 Mexico_K_fill.to_csv('C:/Users/User/Code/databook/data/Mexico/MexicoC
```

```
1 #Save PM Data of Korea data
2 Korea_K_fill.to_csv('C:/Users/User/Code/databook/data/Korea/KoreaClea
```

```
1 #Select which data to plot:
2 old_data = UK_or_df['PM2.5']
3 new_data = UK_fill_df['PM2.5']
4
5 grouped_old = old_data.groupby(old_data.index.year)
```

```
5 grouped_old = old_data.groupby(old_data.index.year)
6 grouped_new = new_data.groupby(new_data.index.year)
7
8 print(old_data)
```

```
DateTime
2019-01-01 01:00:00    14.741
2019-01-01 02:00:00    11.408
2019-01-01 03:00:00     5.407
2019-01-01 04:00:00     5.503
2019-01-01 05:00:00     5.755
2019-01-01 06:00:00     5.527
2019-01-01 07:00:00     5.667
2019-01-01 08:00:00     5.512
2019-01-01 09:00:00     5.576
2019-01-01 10:00:00     5.780
2019-01-01 11:00:00     5.616
2019-01-01 12:00:00     4.002
2019-01-01 13:00:00     3.767
2019-01-01 14:00:00     3.788
2019-01-01 15:00:00     4.141
2019-01-01 16:00:00     4.679
2019-01-01 17:00:00     4.531
2019-01-01 18:00:00     6.571
2019-01-01 19:00:00     8.677
2019-01-01 20:00:00     8.935
2019-01-01 21:00:00     6.283
2019-01-01 22:00:00     5.826
2019-01-01 23:00:00     5.973
2019-01-02 00:00:00       NaN
2019-01-02 01:00:00    10.267
2019-01-02 02:00:00     8.309
2019-01-02 03:00:00     6.833
2019-01-02 04:00:00     4.404
2019-01-02 05:00:00     4.539
2019-01-02 06:00:00     5.783
2019-01-02 07:00:00     6.381
2019-01-02 08:00:00     7.630
2019-01-02 09:00:00     8.259
2019-01-02 10:00:00    11.094
2019-01-02 11:00:00    10.472
2019-01-02 12:00:00    10.675
2019-01-02 13:00:00    12.769
2019-01-02 14:00:00     8.520
2019-01-02 15:00:00     8.267
2019-01-02 16:00:00     6.920
2019-01-02 17:00:00     7.923
2019-01-02 18:00:00    11.163
2019-01-02 19:00:00    10.962
2019-01-02 20:00:00     9.688
2019-01-02 21:00:00     9.557
2019-01-02 22:00:00    10.375
2019-01-02 23:00:00    10.576
2019-01-03 00:00:00       NaN
2019-01-03 01:00:00    15.151
2019-01-03 02:00:00    15.061
2019-01-03 03:00:00    11.498
2019-01-03 04:00:00    12.234
2019-01-03 05:00:00    18.182
```

```
2019-01-03 06:00:00    20.382
2019-01-03 07:00:00    18.797
2019-01-03 08:00:00    21.356
2019-01-03 09:00:00    20.196
```

```
# Find negative values in the DataFrame
negative_mask = new_data < 0

# Print the negative values with their indices
print("Negative Values in DataFrame:")
print(new_data[negative_mask])

# Get indices of negative values
negative_indices = negative_mask.any(1)
indices_of_negative_values = new_data.index[negative_indices].tolist(

# Print indices where negative values are found
print("Indices with Negative Values:")
print(indices_of_negative_values)
```

```
Negative Values in DataFrame:
Series([], Name: PM2.5, dtype: float64)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last)
Cell In[36], line 9
      6 print(new_data[negative_mask])
      8 # Get indices of negative values
----> 9 negative_indices = negative_mask.any(1)
     10 indices_of_negative_values =
new_data.index[negative_indices].tolist()
     12 # Print indices where negative values are found

TypeError: Series.any() takes 1 positional argument but 2 were given
```

```
stdPM25K30 = ((abs(6.669333-6.653877)/6.669333+(8.989437-6.540468)/8.

print(stdPM25K30)
```

```
18.005620292199904
```

```
stdPM25K25 = ((abs(6.669333-7.416088)/6.669333+(8.989437-6.371768)/8.

print(stdPM25K25)
```

```
19.272254554275175
```

```
stdPM25K20 = ((abs(6.669333-7.935228)/6.669333+(8.989437-6.496008)/8.

print(stdPM25K20)
```

```
19.73062315394634
```

```
stdPM25K26 = ((abs(6.669333-7.235088)/6.669333+abs(8.989437-6.410535)

```

```
_
3 print(stdPM25K26)
```

```
18.60411366250863
```

```
1 stdPM25K27 = ((abs(6.669333-7.070391)/6.669333+abs(8.989437-6.453523)
2
3 print(stdPM25K27)
```

```
18.11261140312003
```

```
1 stdPM10K30 = (((8.476048-7.755913)/8.476048+(12.351880- 8.563604)/12.
2
3 print(stdPM10K30)
```

```
22.23827295953838
```

```
1 stdPM10K25 = ((abs(8.476048-8.595012)/8.476048+(12.351880-8.429258)/1
2
3 print(stdPM10K25)
```

```
19.20593153026512
```

```
1 stdPM10K27 = ((abs(8.476048-8.212715)/8.476048+abs(12.351880-8.470781
2
3 print(stdPM10K27)
```

```
20.366802886415016
```

```
1 stdPM10K26 = ((abs(8.476048-8.395274)/8.476048+abs(12.351880-8.443440
2
3 print(stdPM10K26)
```

```
19.583395746807906
```

```
1 def PM25_Mean_calculator(mean2019, mean2020, mean2021, mean2022, mean
2
3     PM25_Mean = ((abs(7.860134-mean2019)/7.860134 + abs(8.610198-mean
4
5     print(PM25_Mean)
6     return(PM25_Mean)
```

```
1 mean2019 =  9.967921
2 mean2020 = 8.383908
3 mean2021 = 7.342154
4 mean2022 = 9.015066
5 mean2023 = 6.900610
6 P25_K25_mean = PM25_Mean_calculator(mean2019,mean2020, mean2021, mean
7
```

```
14.090394229770942
```

```
mean2019 =  9.903337
mean2020 =  8.504357
mean2021 =  7.367673
mean2022 = 8.939114
mean2023 = 7.019881
P25_K26_mean = PM25_Mean_calculator(mean2019,mean2020, mean2021, mean
```

```
13.05252001213361
```

```
mean2019 =   9.836766
mean2020 =  8.595274
mean2021 =  7.376906
mean2022 = 8.867190
mean2023 =  7.109005
P25_K27_mean = PM25_Mean_calculator(mean2019,mean2020, mean2021, mean
7
```

```
12.21089919693115
```

```
def PM10_Mean_calculator(mean2019, mean2020, mean2021, mean2022, mean
2
3     PM10_Mean = ((abs(12.786331-mean2019)/12.786331 + abs(14.121757-m
4
5     print(PM10_Mean)
6     return(PM10_Mean)
```

```
mean2019 =  14.705845
mean2020 = 12.710035
mean2021 = 10.940160
mean2022 = 13.436019
mean2023 = 11.067249
P10_K25_mean = PM10_Mean_calculator(mean2019,mean2020, mean2021, mean
7
```

```
12.44403820297543
```

```
mean2019 =  14.611628
mean2020 = 12.861032
mean2021 = 10.982364
mean2022 = 13.346716
mean2023 = 11.226019
P10_K26_mean = PM10_Mean_calculator(mean2019,mean2020, mean2021, mean
7
```

```
11.617175147850848
```

```
mean2019 =  14.522029
mean2020 = 12.980646
mean2021 = 11.006023
mean2022 = 13.260031
mean2023 = 11.334385
```

```
6 P10_K27_mean = PM10_Mean_calculator(mean2019,mean2020, mean2021, mean
```

```
10.95526651785195
```

```
1 mean2019 = 14.192712
2 mean2020 = 13.175501
3 mean2021 = 11.009497
4 mean2022 = 13.021303
5 mean2023 = 11.546916
6 P10_K30_mean = PM10_Mean_calculator(mean2019,mean2020, mean2021, mean
```

```
9.427483614091189
```

```
1 Col_PM25_mean_change = ((abs(29.783682 - 28.821035)/29.783682+abs(28.
2
3 print(Col_PM25_mean_change)
```

```
1.1595619191829
```

```
1 omean1 = 37.525830
2 imean1 = 35.910604
3 omean2 = 33.046621
4 imean2 =  31.605245
5 omean3 =  33.320409
6 imean3 = 31.828604
7 omean4 = 28.522565
8 imean4 = 27.943796
9 omean5 = 34.553358
10 imean5 =  33.720975
11 omean6 =  34.001631
12 imean6 =  31.986114
13 n = 6
14
15 Percentage_Change = ((abs(imean1 - omean1)/omean1+abs(imean2- omean2)
16 print(Percentage_Change)
```

```
3.918157802280703
```

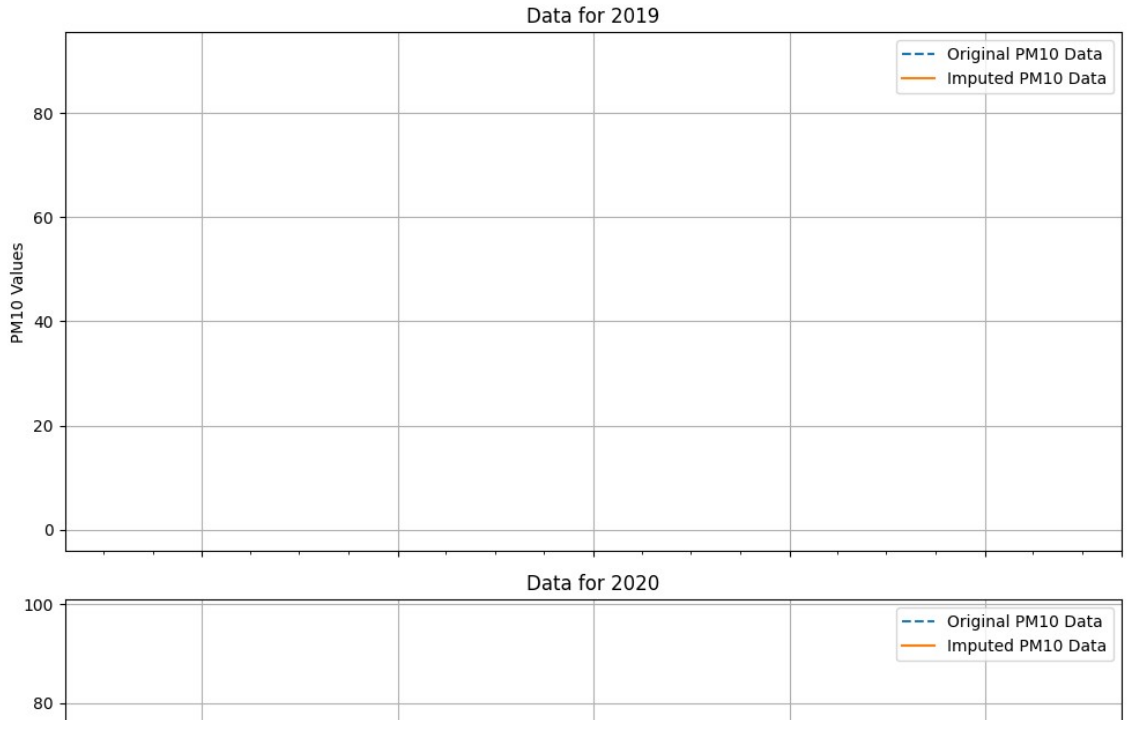
```
1 #Plot the data so that we can vizualise that the imputation does not
2 #Plot the data by years so that you can see the patterns in the data
3
4 #Select which data to plot:
5 old_data = UK_or_df['PM2.5']
6 new_data = UK_K_fill['PM2.5']
7
8 #Group the data by year
9 grouped_old = old_data.groupby(old_data.index.year)
10 grouped_new = new_data.groupby(new_data.index.year)
11
12
13
14 # Set up the matplotlib figure and axes
15 fig, axes = plt.subplots(nrows=len(grouped_old), figsize=(10, 5 * len
```

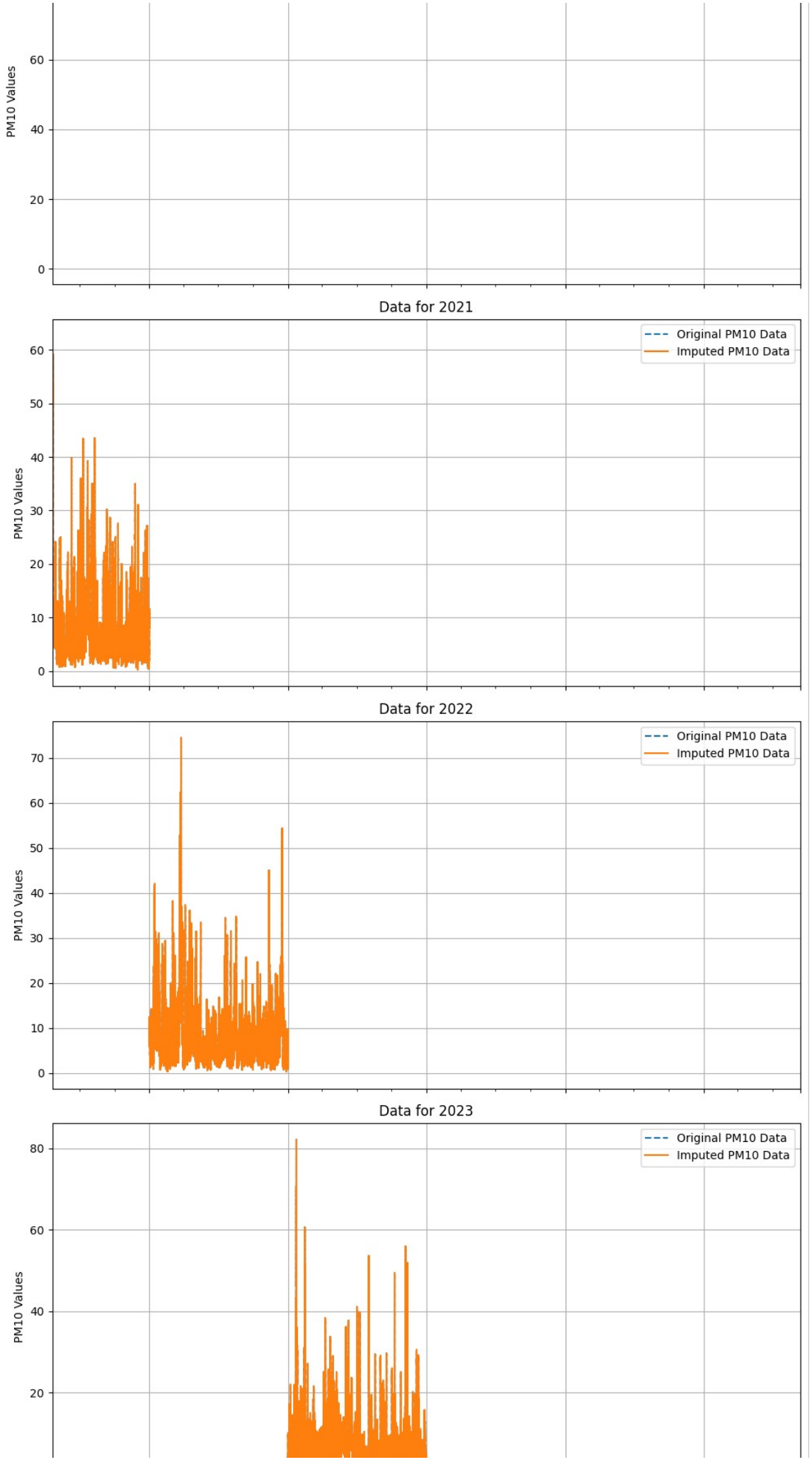
```
fig, axes = plt.subplots(nrows=len(grouped_old), figsize=(10, 5 * len
# Plot each group
for ((key1, group1), (key2, group2)), ax in zip(zip(grouped_old, grou
    print("Plotting Old Data year:", key1)
    print(group1.describe())
    print("Plotting Imputed Data year:", key2)
    print(group2.describe())

    # Plot data from the first dataset
    group1.plot(ax=ax, label='Original PM10 Data', linestyle='--')

    # Plot data from the second dataset
    group2.plot(ax=ax, label='Imputed PM10 Data')  # Dashed line for

    # Set the title to the current year from the first dataset's key
    ax.set_title(f'Data for {key1}')
    ax.set_ylabel('PM10 Values')
    ax.grid(True)
    ax.legend()

plt.tight_layout()
plt.show()
```

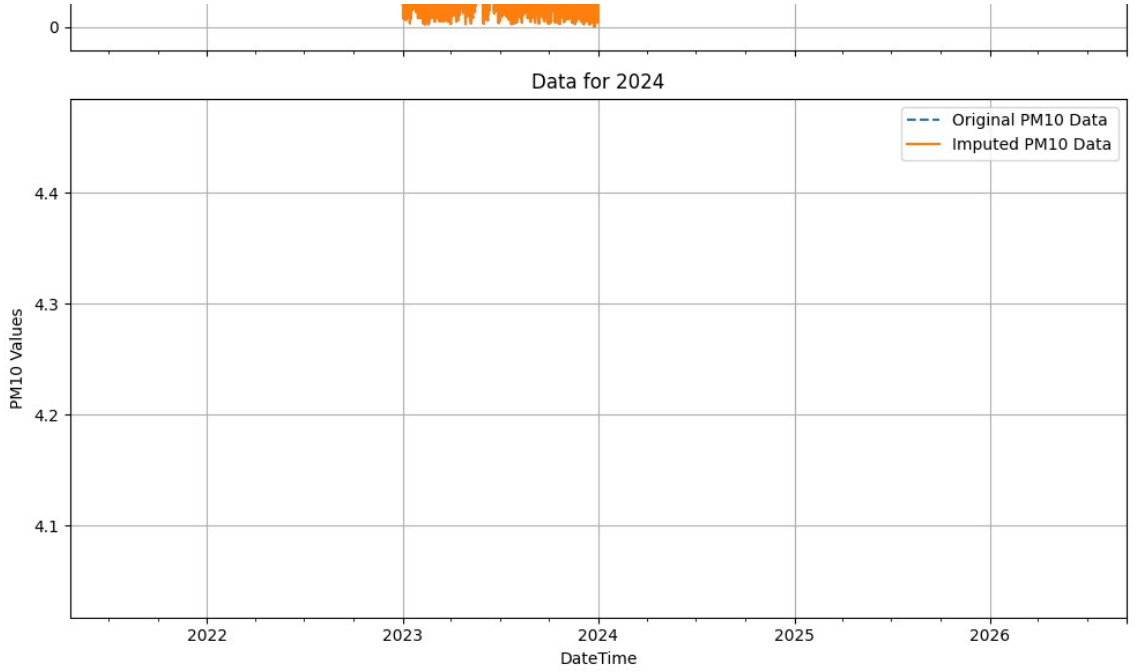
```
Plotting Old Data year: 2019
count    8509.000000
mean        9.535257
std         9.161261
min         0.413000
25%         4.130000
50%         6.337000
75%        11.175000
max        90.981000
Name: PM2.5, dtype: float64
Plotting Imputed Data year: 2019
count    8759.000000
mean        9.471212
std         9.068612
min         0.413000
25%         4.161500
50%         6.349000
75%        10.968500
max        90.981000
Name: PM2.5, dtype: float64
Plotting Old Data year: 2020
count    8681.000000
mean        7.916442
std         7.582349
min         0.319000
25%         3.575000
50%         5.449000
75%         9.155000
max        96.163000
```

```
max        96.163000
Name: PM2.5, dtype: float64
Plotting Imputed Data year: 2020
count    8784.000000
mean        7.966047
std         7.579192
min         0.319000
25%         3.600750
50%         5.483500
75%         9.264000
max        96.163000
Name: PM2.5, dtype: float64
Plotting Old Data year: 2021
count    8703.000000
mean        7.747758
std         6.268321
min         0.280000
25%         3.763000
50%         5.929000
75%         9.600000
max        62.623000
Name: PM2.5, dtype: float64
Plotting Imputed Data year: 2021
count    8760.000000
mean        7.758845
std         6.282386
min         0.280000
25%         3.759750
50%         5.929000
75%         9.612500
max        62.623000
Name: PM2.5, dtype: float64
Plotting Old Data year: 2022
count    8694.000000
mean        7.903948
std         6.798462
min         0.281000
25%         3.841000
50%         5.684500
75%         9.517000
max        74.470000
Name: PM2.5, dtype: float64
Plotting Imputed Data year: 2022
count    8760.000000
mean        7.894994
std         6.778363
min         0.281000
25%         3.851750
50%         5.706000
75%         9.475500
max        74.470000
Name: PM2.5, dtype: float64
Plotting Old Data year: 2023
count    8749.000000
mean        7.135204
std         6.407699
min         0.147000
25%         3.490000
50%         5.291000
```

```
50%         5.291000
75%         8.463000
max        82.144000
Name: PM2.5, dtype: float64
Plotting Imputed Data year: 2023
count    8760.000000
mean        7.132763
std         6.404382
min         0.147000
25%         3.490000
50%         5.288500
75%         8.463000
max        82.144000
Name: PM2.5, dtype: float64
Plotting Old Data year: 2024
count    1.000
mean     4.251
std        NaN
min      4.251
25%      4.251
50%      4.251
75%      4.251
max      4.251
Name: PM2.5, dtype: float64
Plotting Imputed Data year: 2024
count    1.000
mean     4.251
std        NaN
min      4.251
25%      4.251
50%      4.251
75%      4.251
max      4.251
Name: PM2.5, dtype: float64
c:\Users\User\Code\databook\Lib\site-packages\pandas\plotting\_matplotlib\
  ax.set_xlim(left, right)
c:\Users\User\Code\databook\Lib\site-packages\pandas\plotting\_matplotlib\
  ax.set_xlim(left, right)
```




PM10 Values
Data for 2021
Original PM10 Data
Imputed PM10 Data
PM10 Values
Data for 2022
Original PM10 Data
Imputed PM10 Data
PM10 Values
Data for 2023
Original PM10 Data
Imputed PM10 Data
PM10 Values

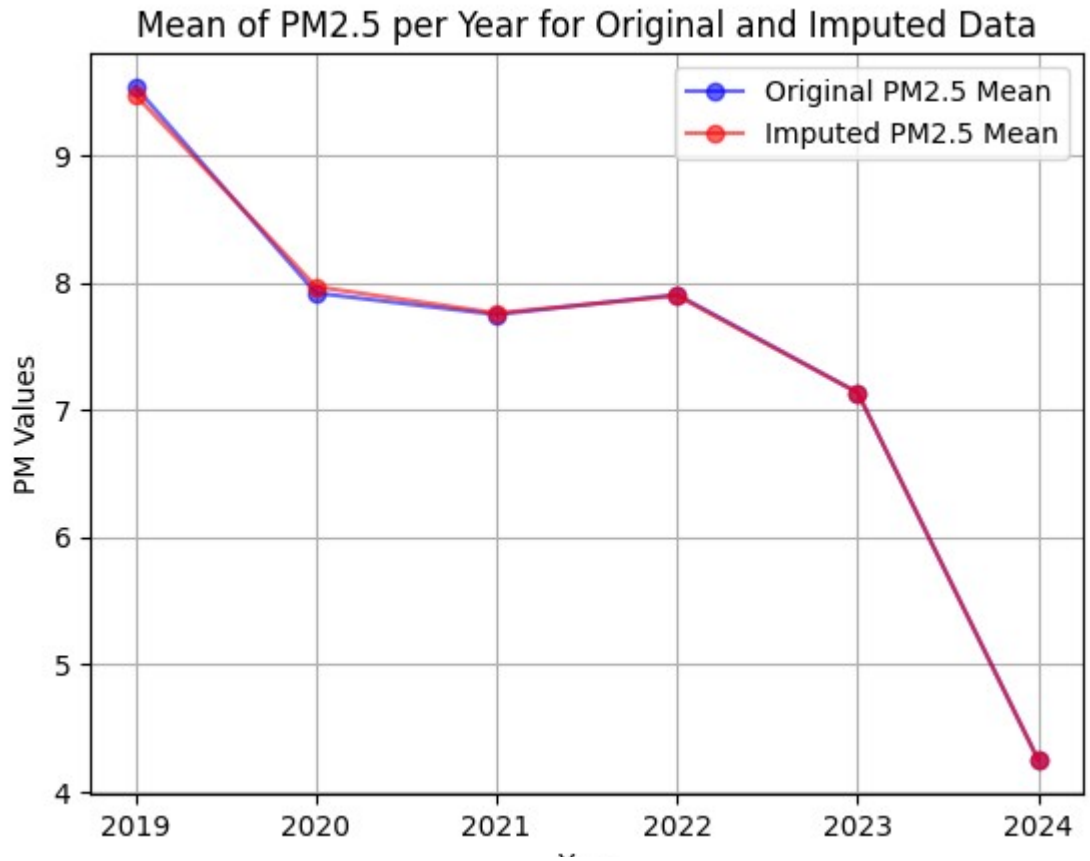

0
Data for 2024
Original PM10 Data
Imputed PM10 Data
4.4
4.3
4.2
4.1
PM10 Values
2022
2023
2024
2025
2026
DateTime

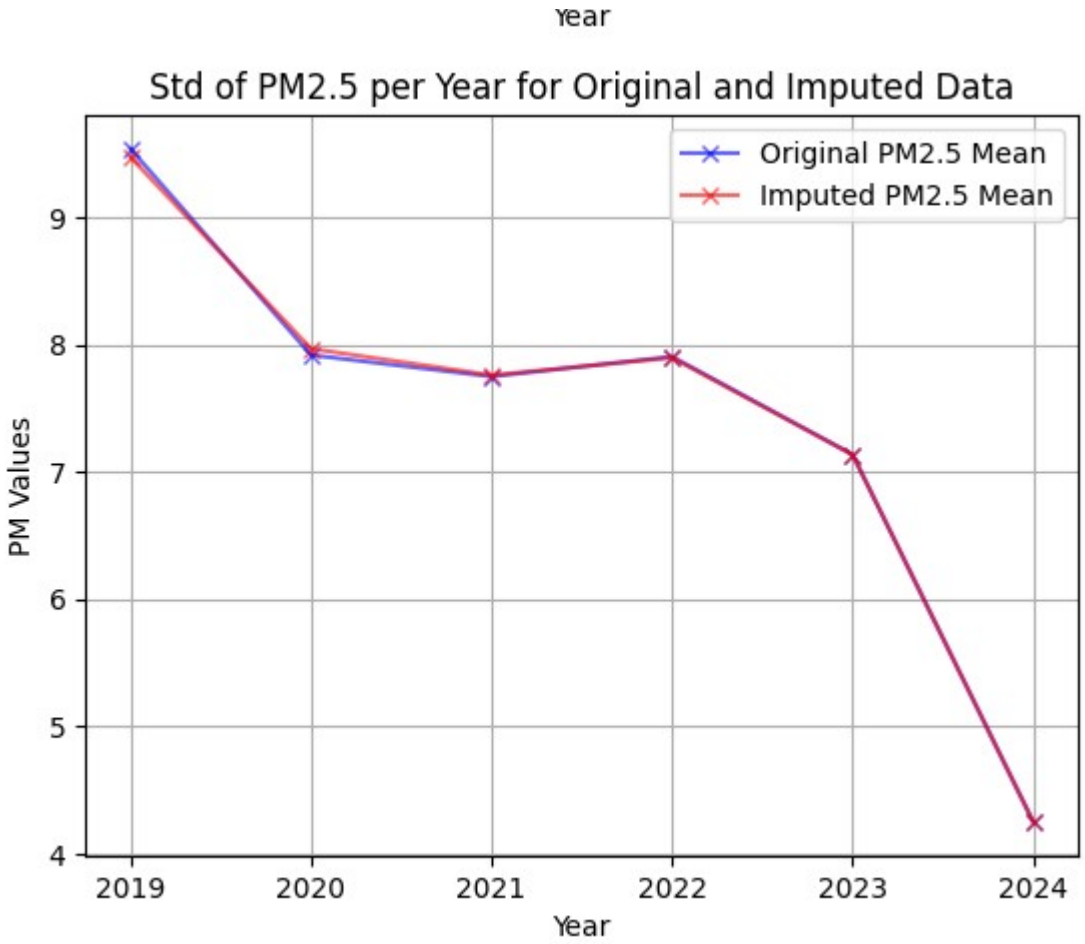
```
#function to print out and plot the statistics of the dataframe
def PM_imputation_stats_calc(df_orginal, df_impute, input_data):

    #Calculate Overall Statistics of original and Imputed Data
    #print(f"Original {input_data} Data Mean = {df_orginal[input_data
    #print(f"Imputed {input_data} Data Mean = {df_impute[input_data].
    print(f"Percentage Change of {input_data} Overall Mean = {(abs(df

    #print(f"Original {input_data} Data Std = {df_orginal[input_data]
    #print(f"Imputed {input_data} Data Std = {df_impute[input_data].s
    print(f"Percentage Change of {input_data} Overall Std = {(abs(df_


    grouped_original = df_orginal.groupby(df_orginal.index.year)
    grouped_imputed = df_impute.groupby(df_impute.index.year)

    #PCalcuate mean and std statistics
    group_original_mean = grouped_original[input_data].mean()
    grouped_imputed_mean = grouped_imputed[input_data].mean()
    group_original_std = grouped_original[input_data].std()
    grouped_imputed_std = grouped_imputed[input_data].std()


```

```
    #Plotting the means
    fig, ax = plt.subplots()

    ax.plot(group_original_mean.index, group_original_mean, color='bl
    ax.plot(grouped_imputed_mean.index, grouped_imputed_mean, color='

    plt.xlabel('Year')
    plt.ylabel('PM Values')
    plt.title(f'Mean of {input_data} per Year for Original and Impute
    plt.legend()
    plt.grid(True)
    plt.show()

    #Plotting the standard deviation
    fig, ax = plt.subplots()

    ax.plot(group_original_mean.index, group_original_mean, color='bl
    ax.plot(grouped_imputed_mean.index, grouped_imputed_mean, color='

    plt.xlabel('Year')
    plt.ylabel('PM Values')
    plt.title(f'Std of {input_data} per Year for Original and Imputed
    plt.legend()
    plt.grid(True)
    plt.show()
```

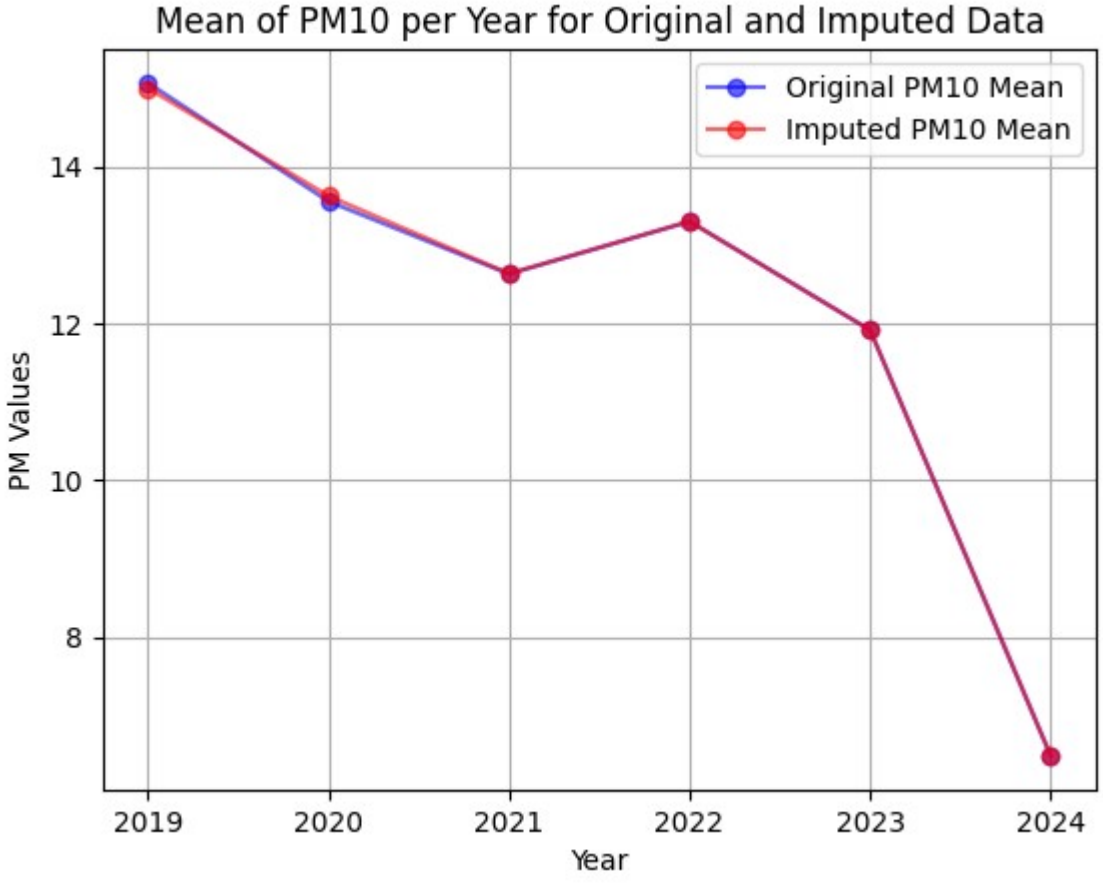
```
PM_imputation_stats_calc(UK_or_df, UK_K_fill, 'PM2.5')
PM_imputation_stats_calc(UK_or_df, UK_K_fill, 'PM10')
```

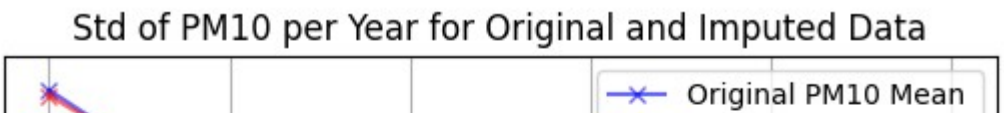
```
Percentage Change of PM2.5 Overall Mean = 0.05605355130964996
Percentage Change of PM2.5 Overall Std = 0.23795888178094704
```

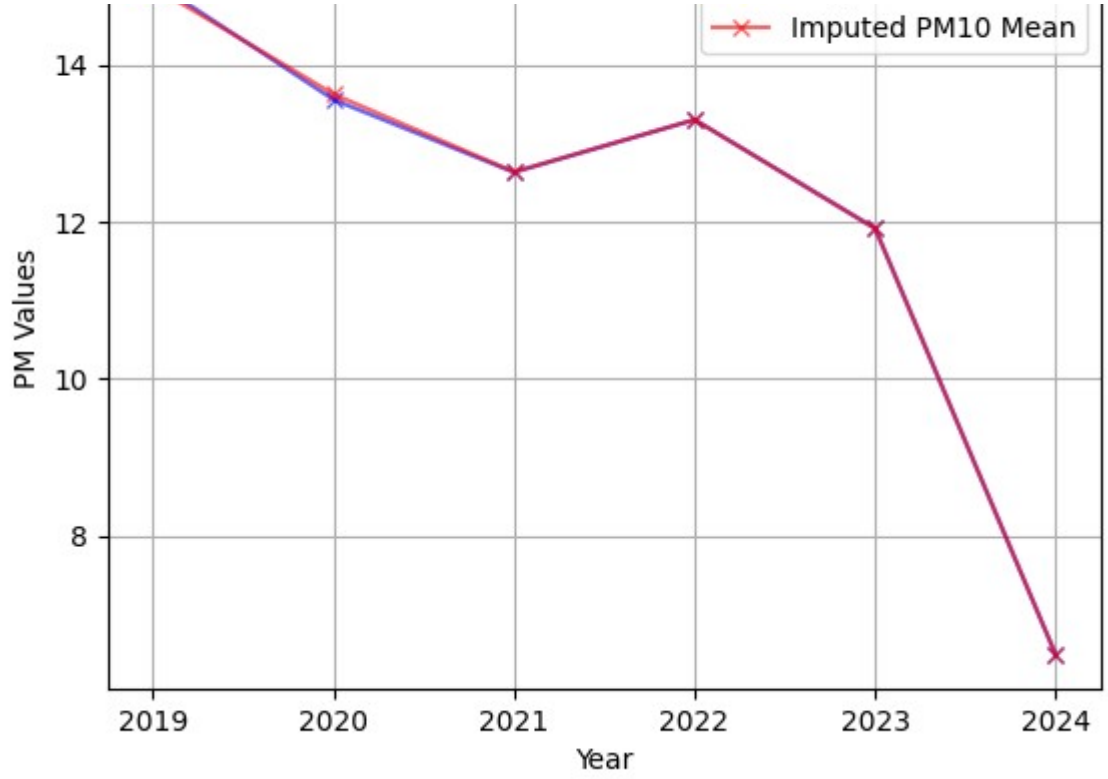





```
Percentage Change of PM10 Overall Mean = 0.08028171681506606
Percentage Change of PM10 Overall Std = 0.32061487495746716
```

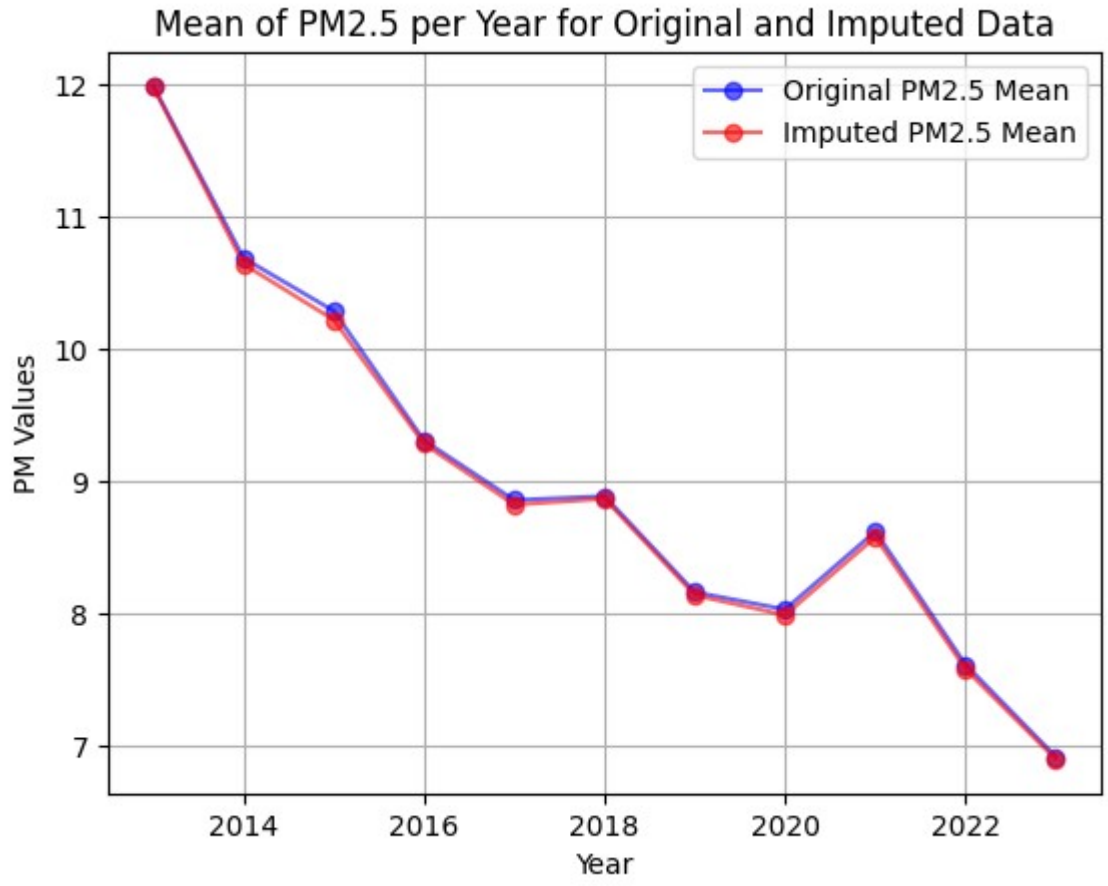


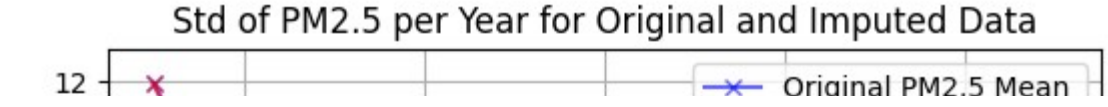


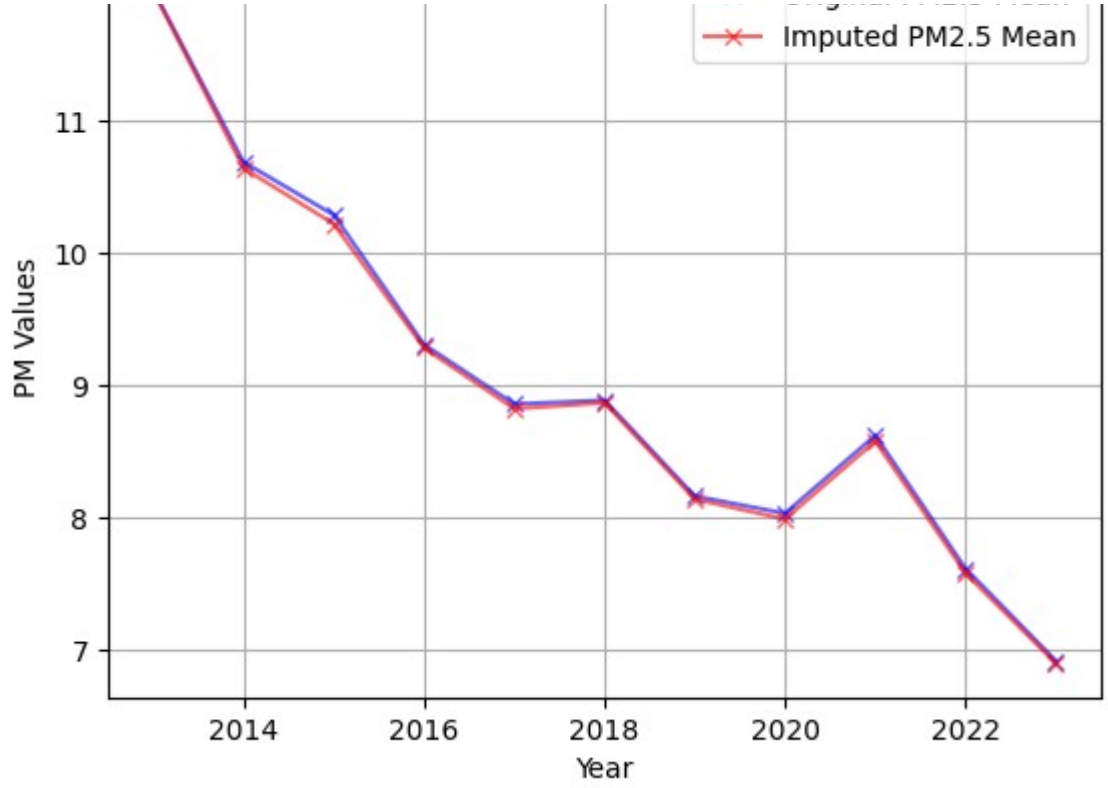



```
1 PM_imputation_stats_calc(US_or_df, US_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(US_or_df, US_K_fill, 'PM10')
```

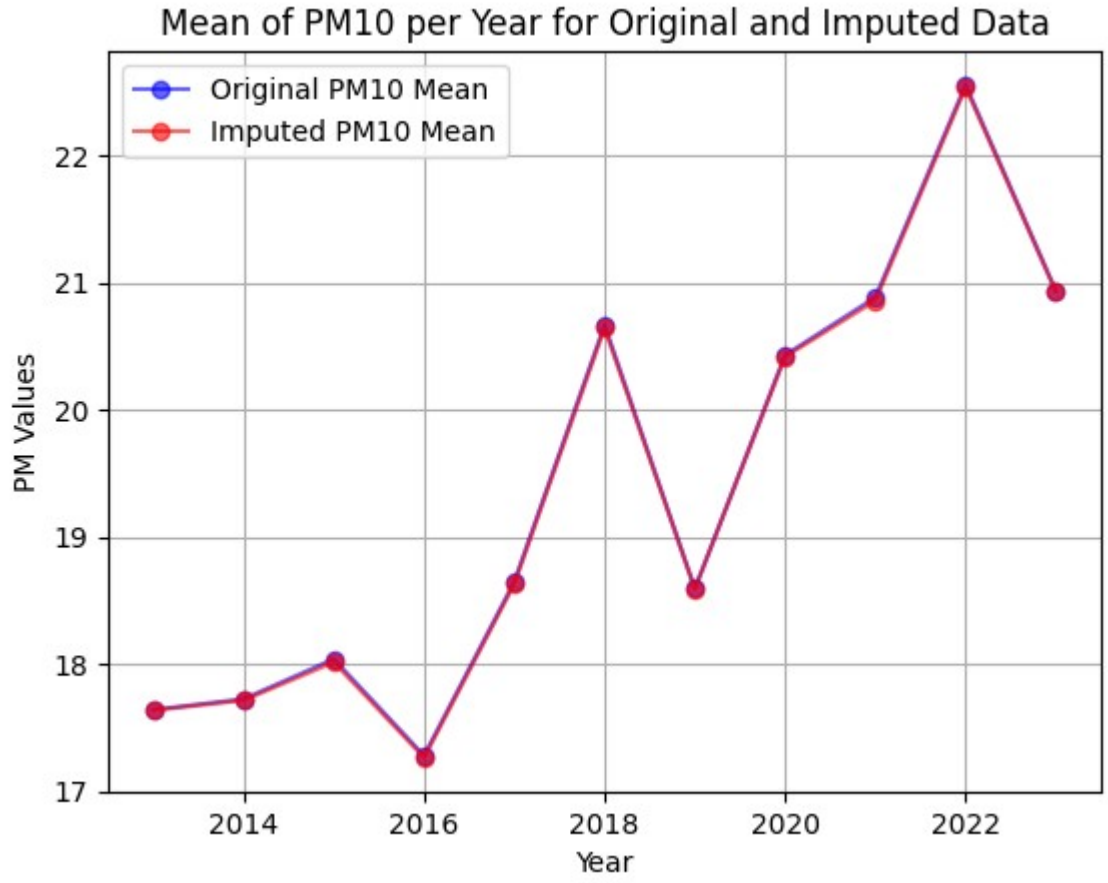
```
Percentage Change of PM2.5 Overall Mean = 0.427891717051312
Percentage Change of PM2.5 Overall Std = 0.5778582720704228
```

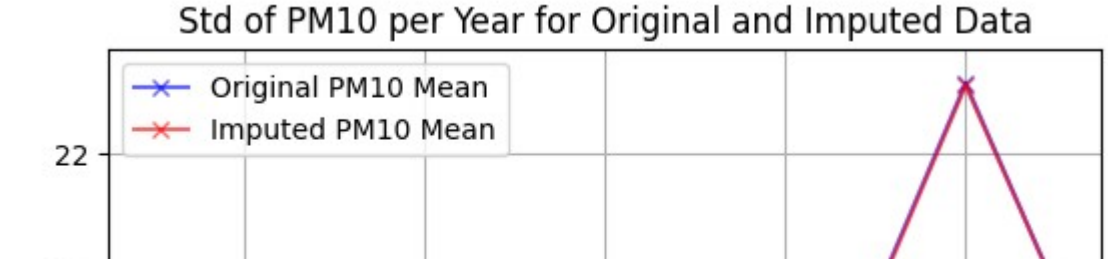


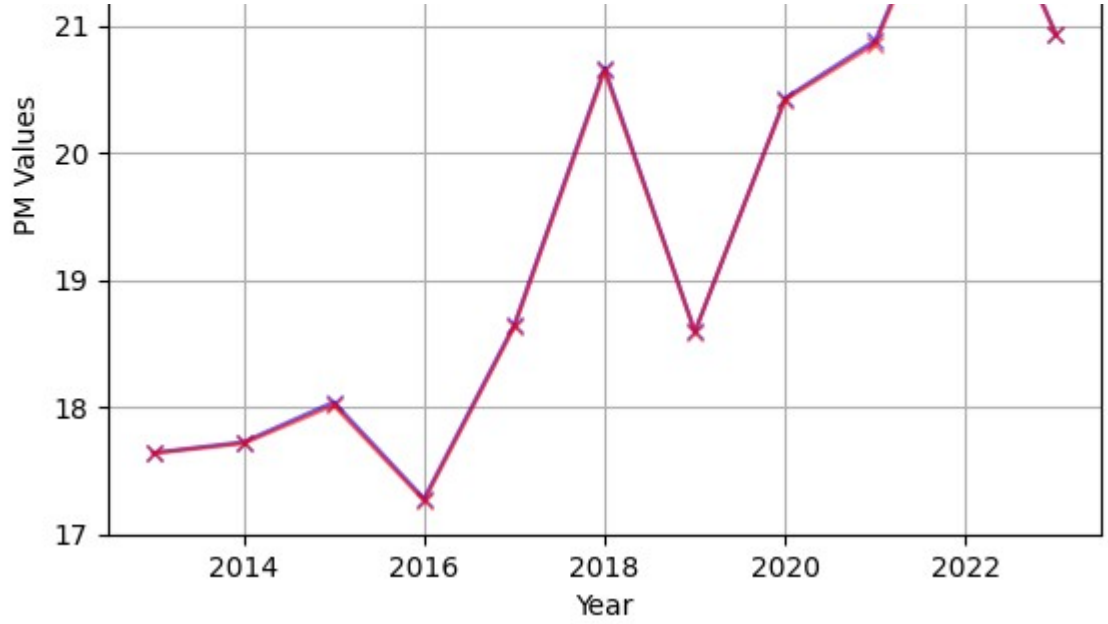





```
Percentage Change of PM10 Overall Mean = 0.0611882319009041
Percentage Change of PM10 Overall Std = 0.1673138123306832
```

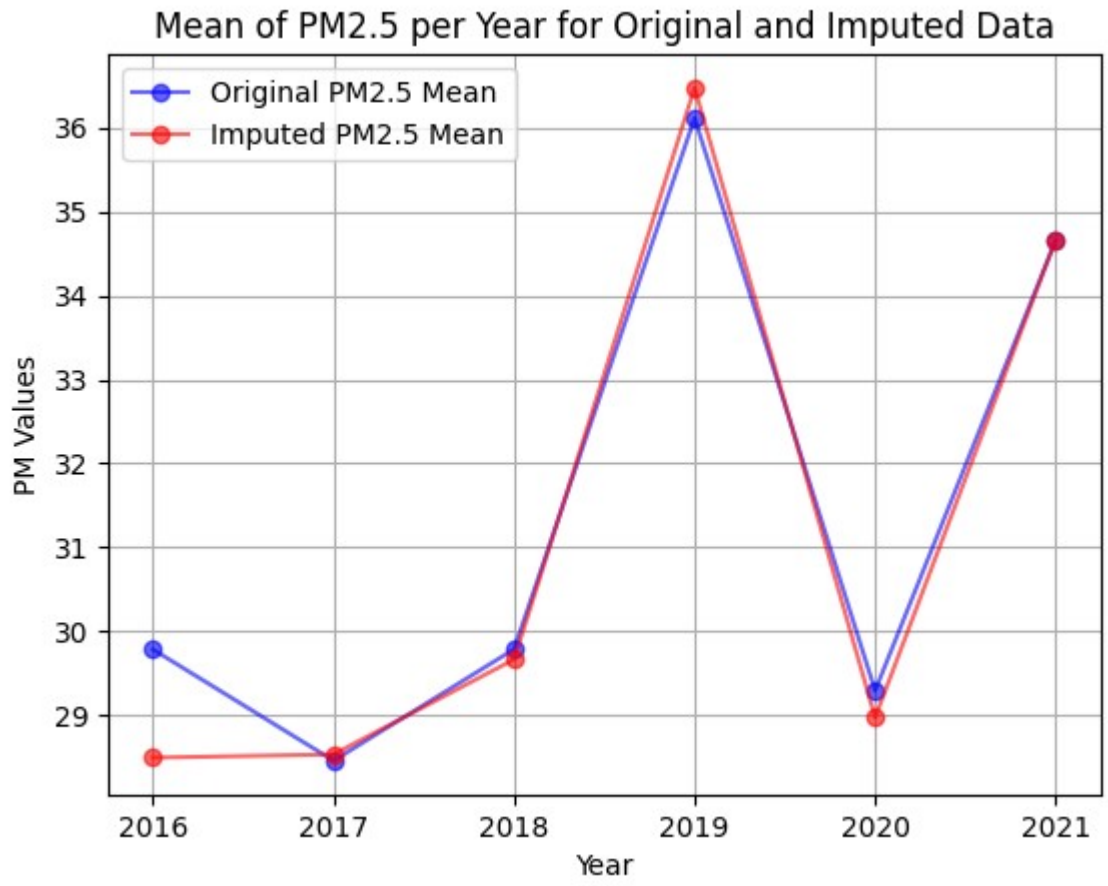


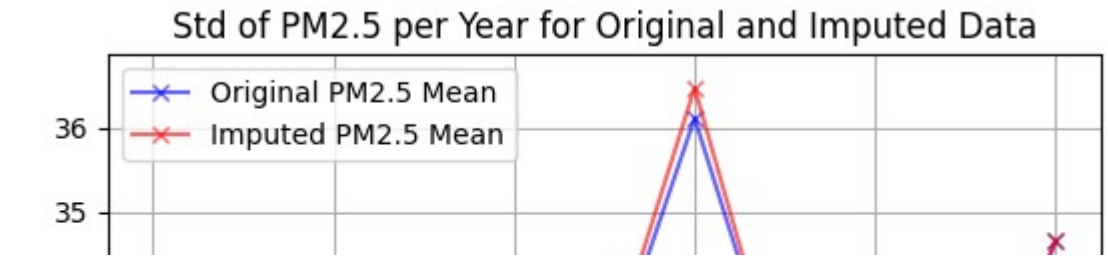


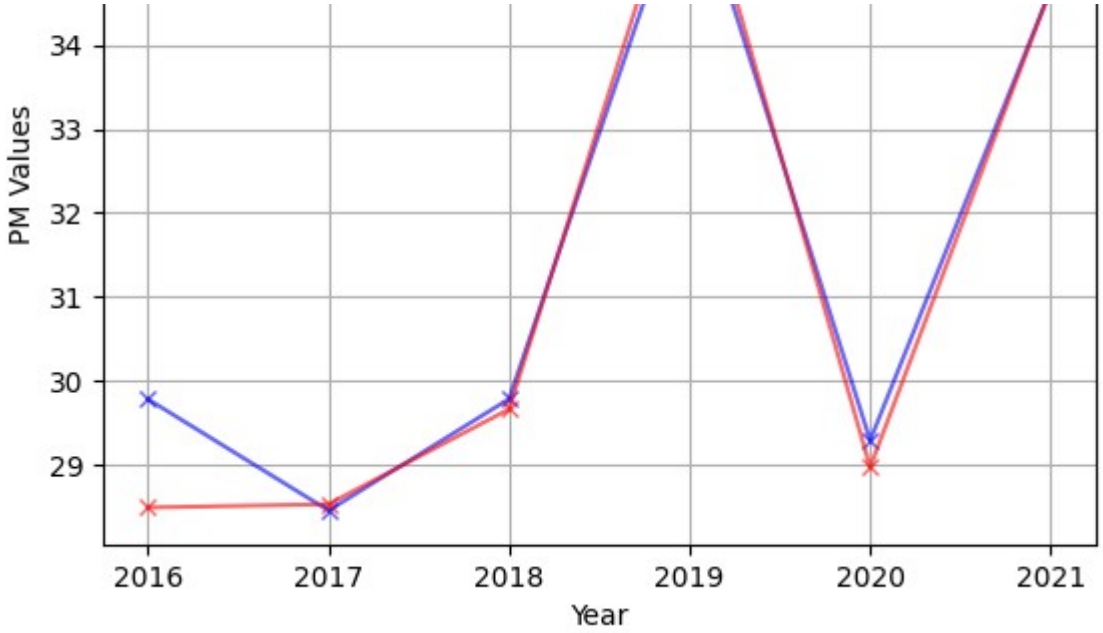



```
1 PM_imputation_stats_calc(Columbia_or_df, Columbia_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(Columbia_or_df, Columbia_K_fill, 'PM10')
```

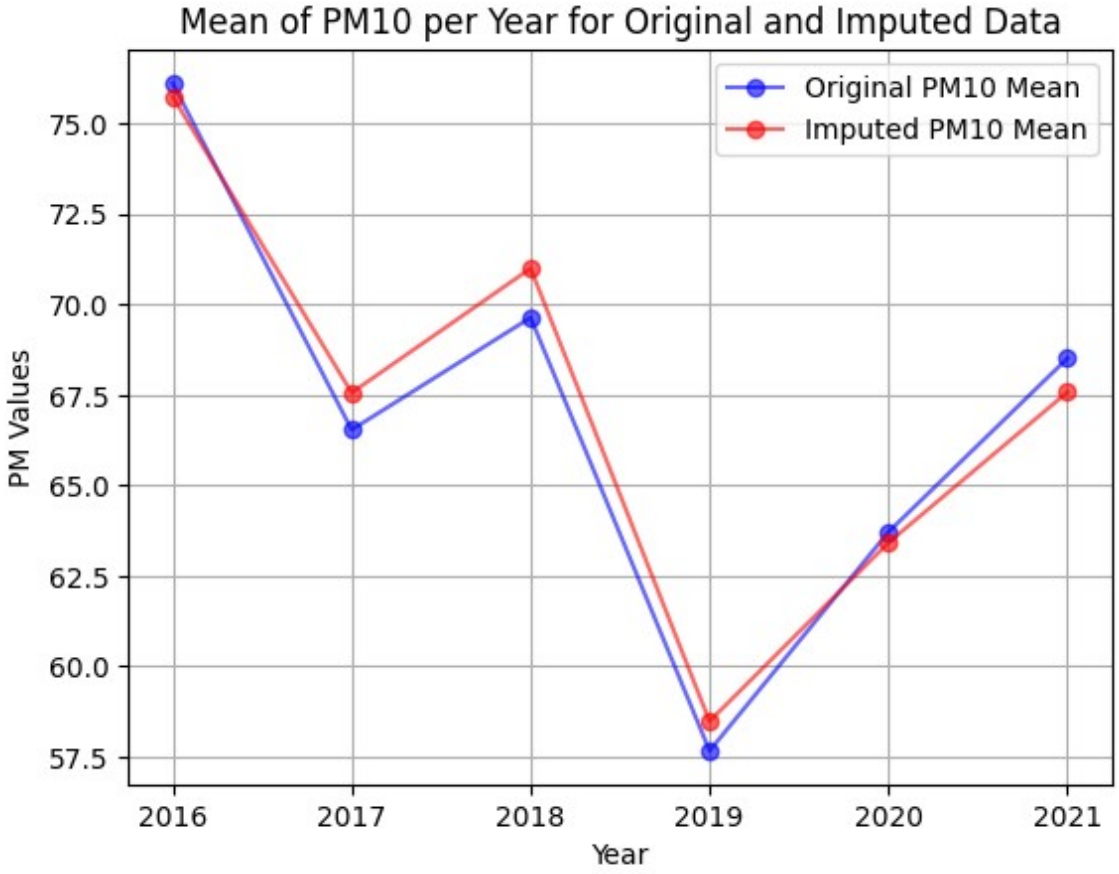
```
Percentage Change of PM2.5 Overall Mean = 0.7321924247741135
Percentage Change of PM2.5 Overall Std = 1.5793313433095268
```

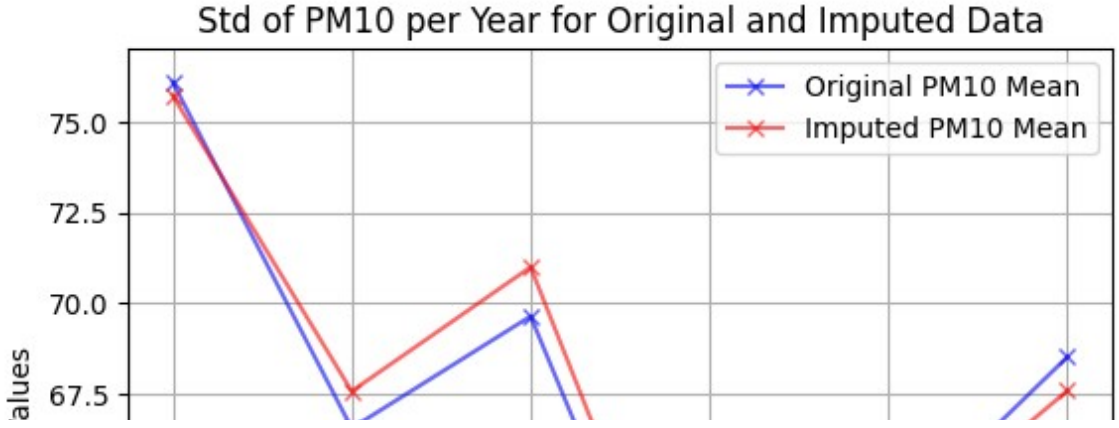


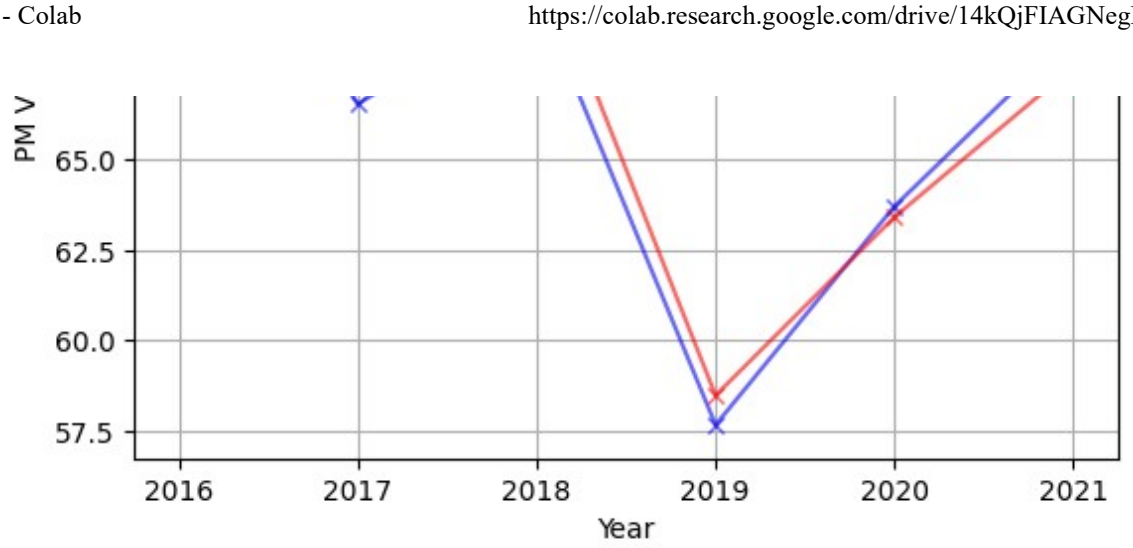





```
Percentage Change of PM10 Overall Mean = 0.4651727282316991
Percentage Change of PM10 Overall Std = 2.281018247802094
```

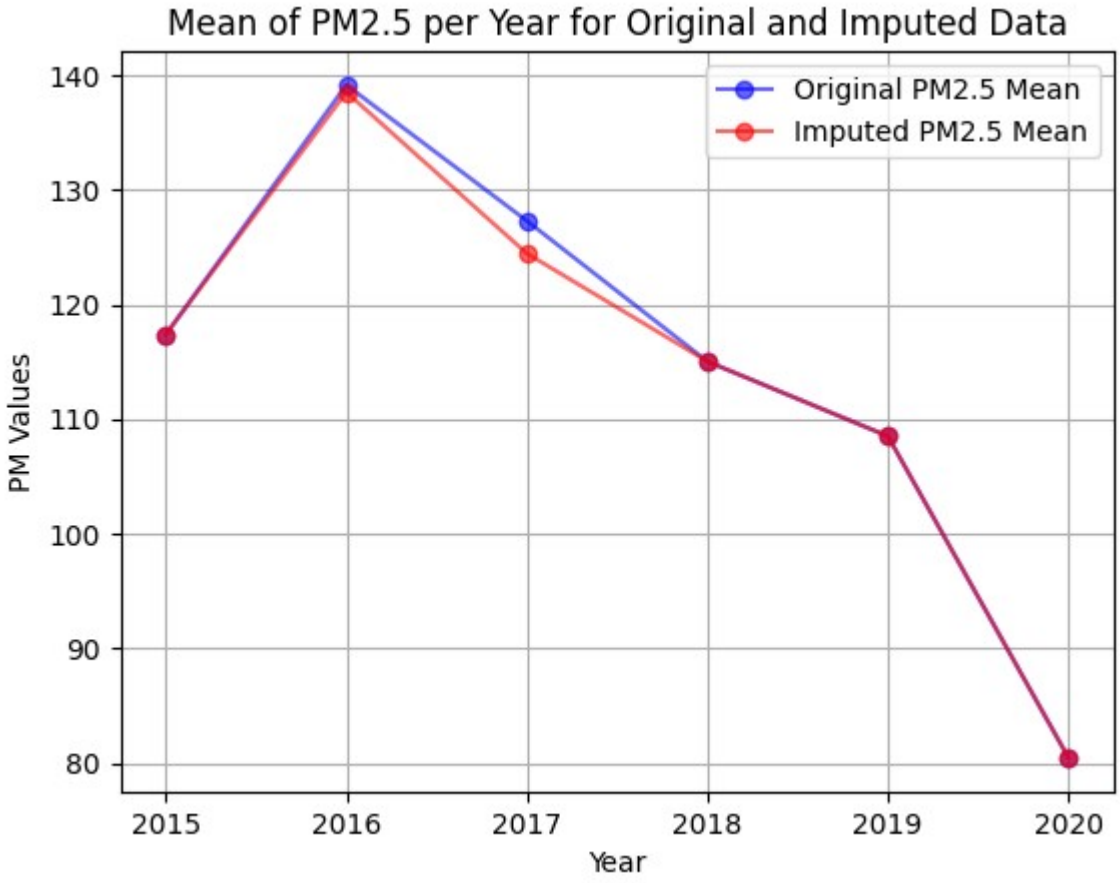


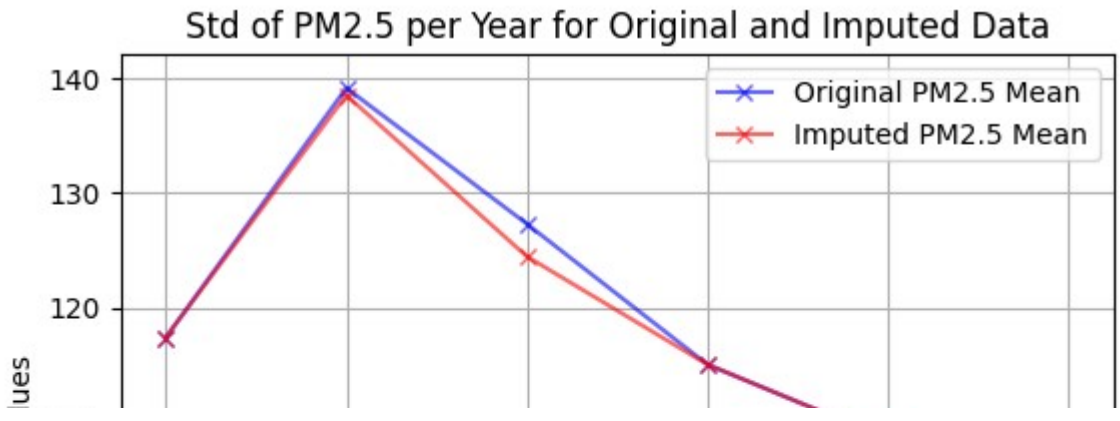


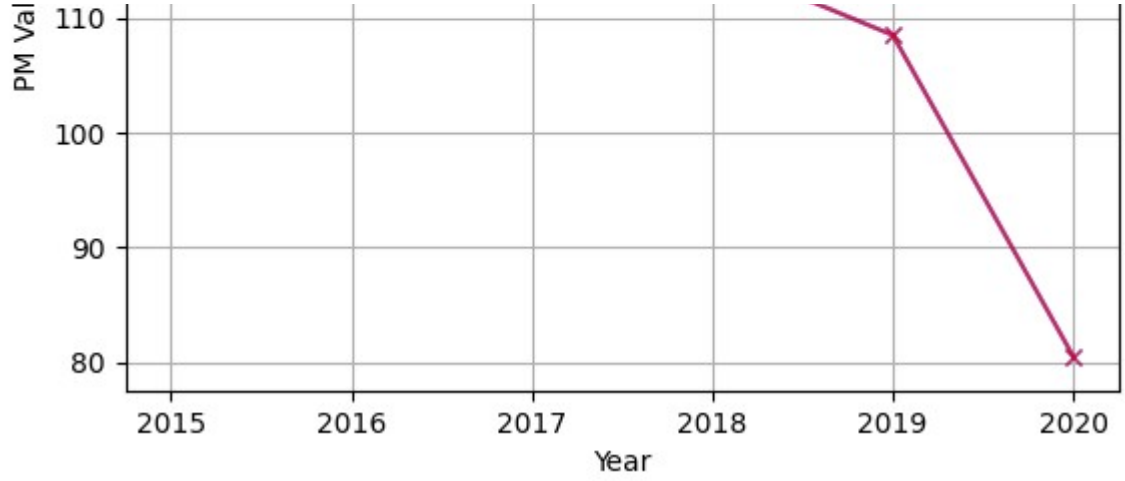



```
1 PM_imputation_stats_calc(India_or_df, India_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(India_or_df, India_K_fill, 'PM10')
```

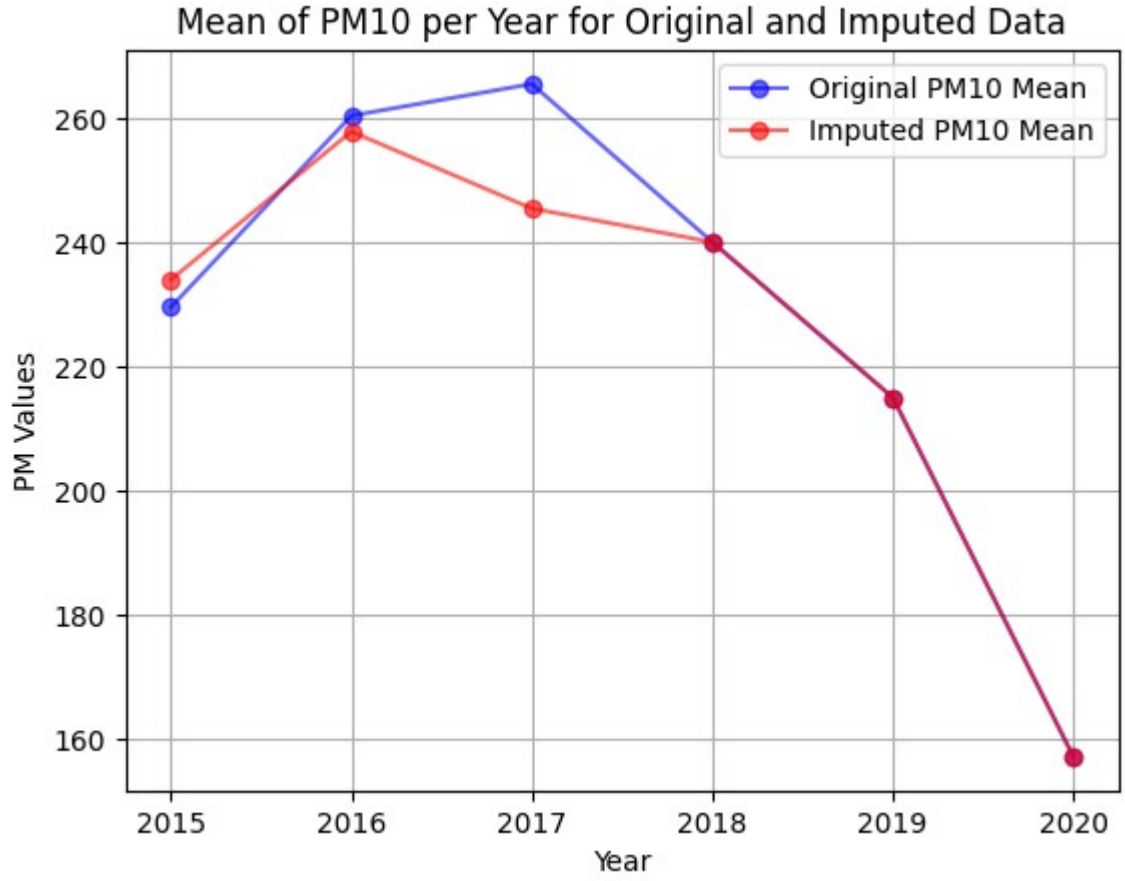
```
Percentage Change of PM2.5 Overall Mean = 0.4604876644418227
Percentage Change of PM2.5 Overall Std = 0.14906950836784796
```

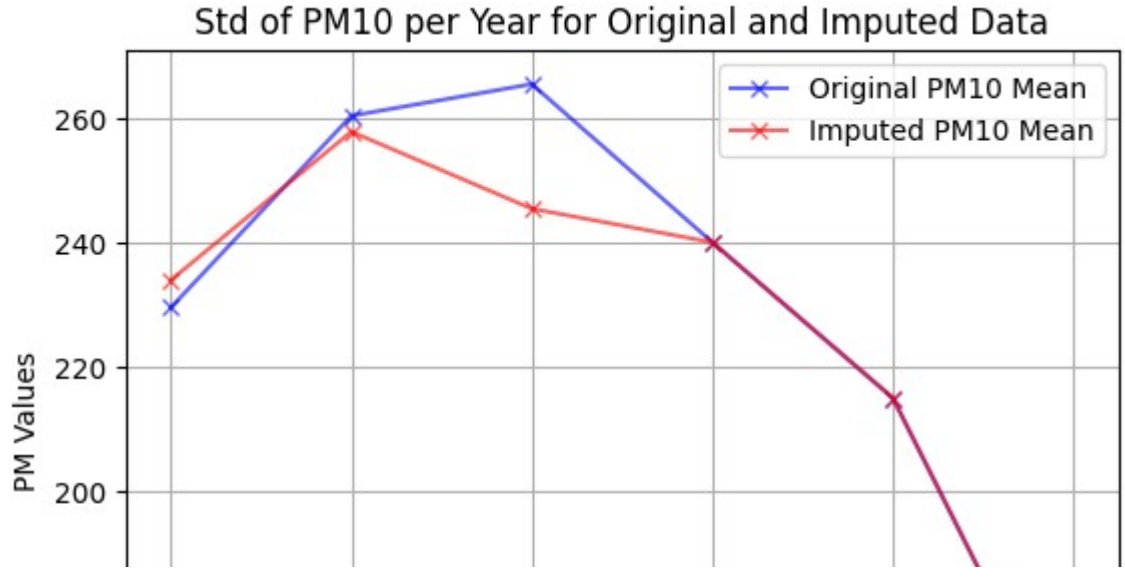


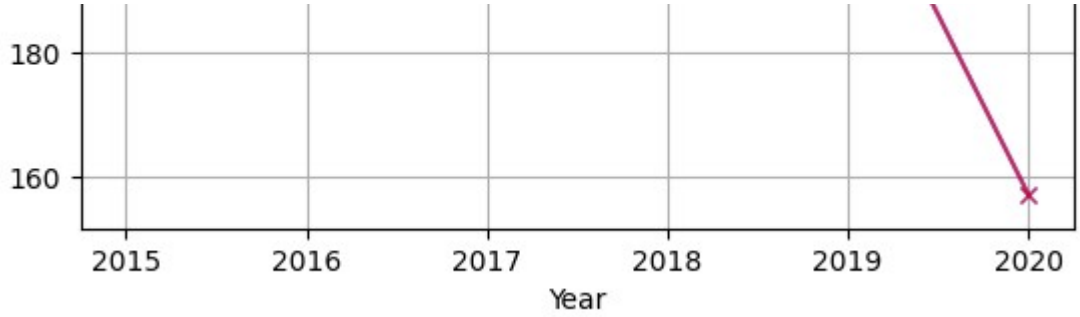





```
Percentage Change of PM10 Overall Mean = 0.8385303779176518
Percentage Change of PM10 Overall Std = 0.5125678445818023
```

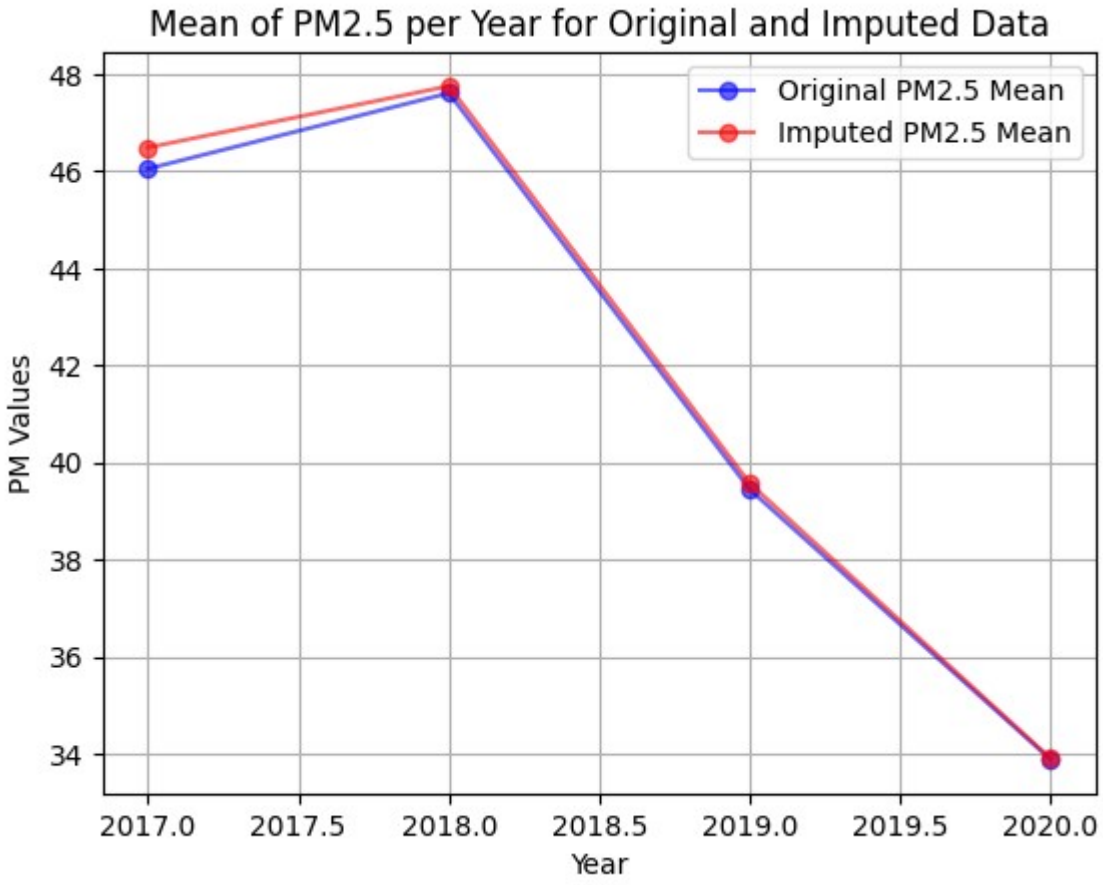


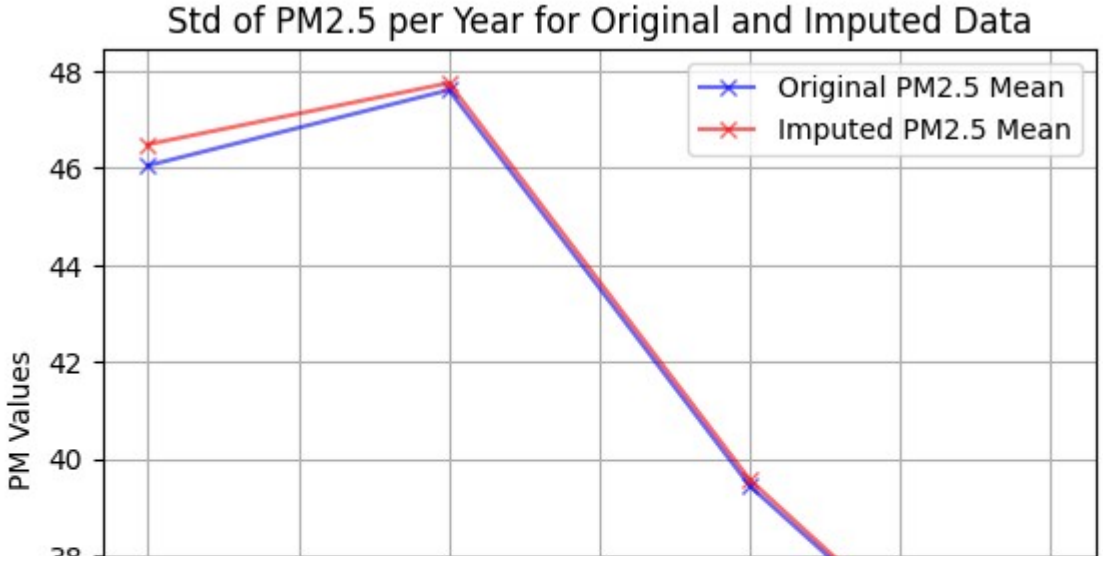


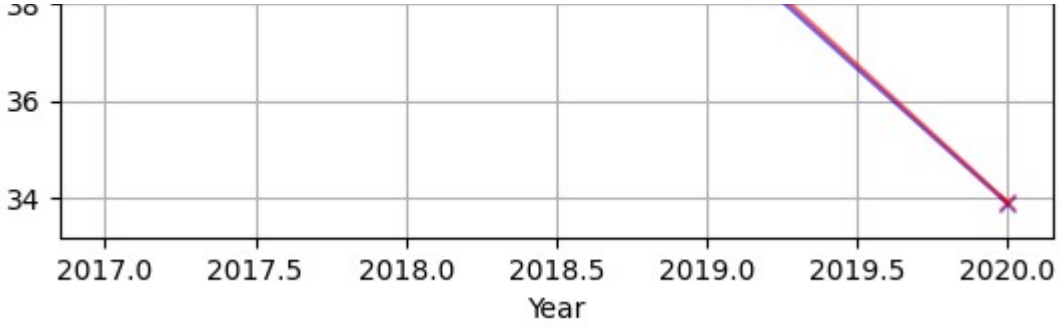



```
1 PM_imputation_stats_calc(China_or_df, China_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(China_or_df, China_K_fill, 'PM10')
```

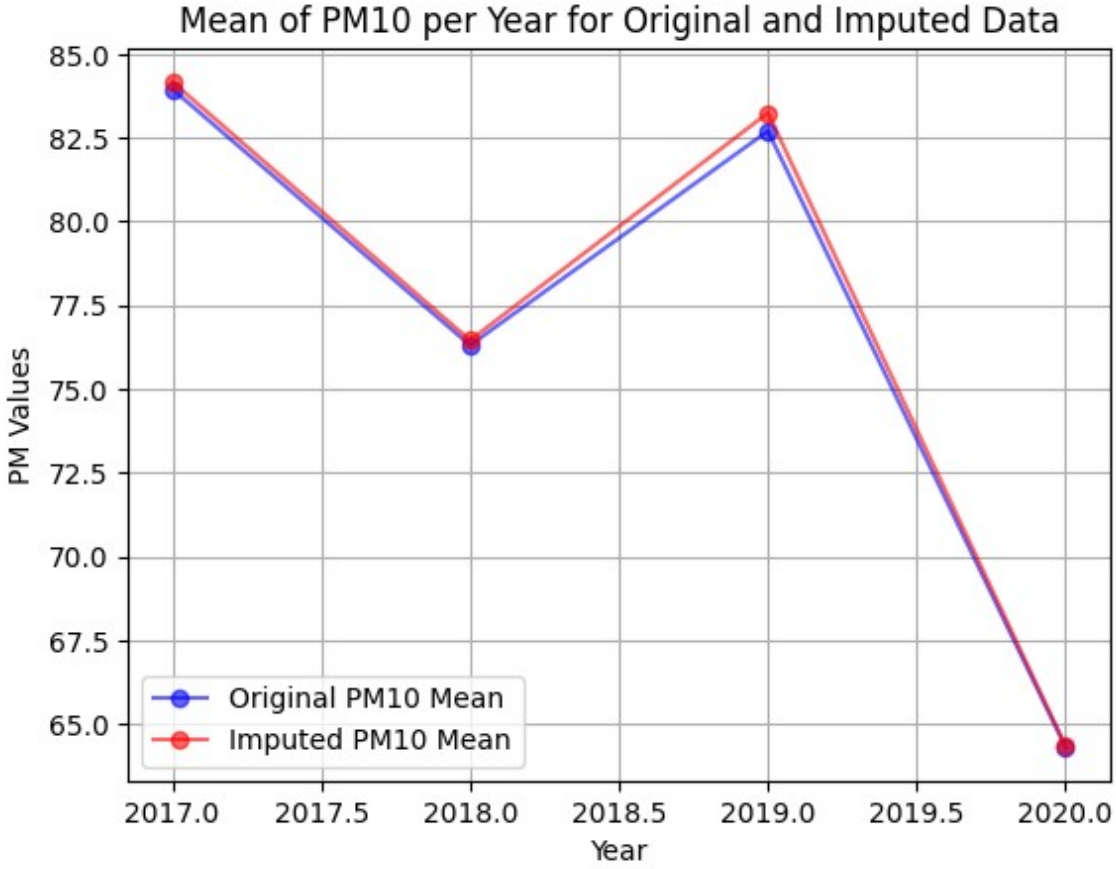
```
Percentage Change of PM2.5 Overall Mean = 0.44918468554038277
Percentage Change of PM2.5 Overall Std = 1.1434616881837634
```

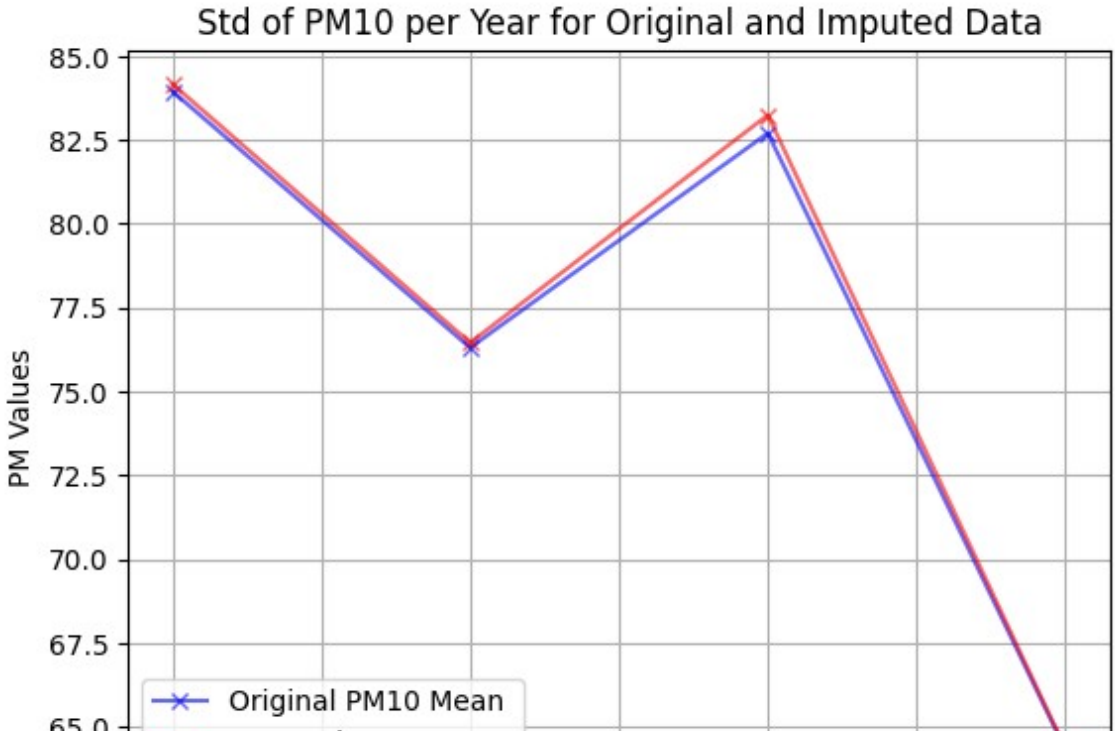


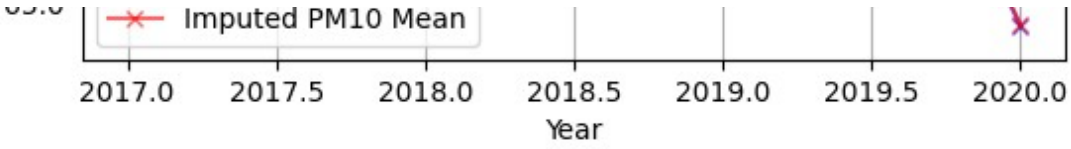





```
Percentage Change of PM10 Overall Mean = 0.31353906839155576
Percentage Change of PM10 Overall Std = 0.7985179262126181
```

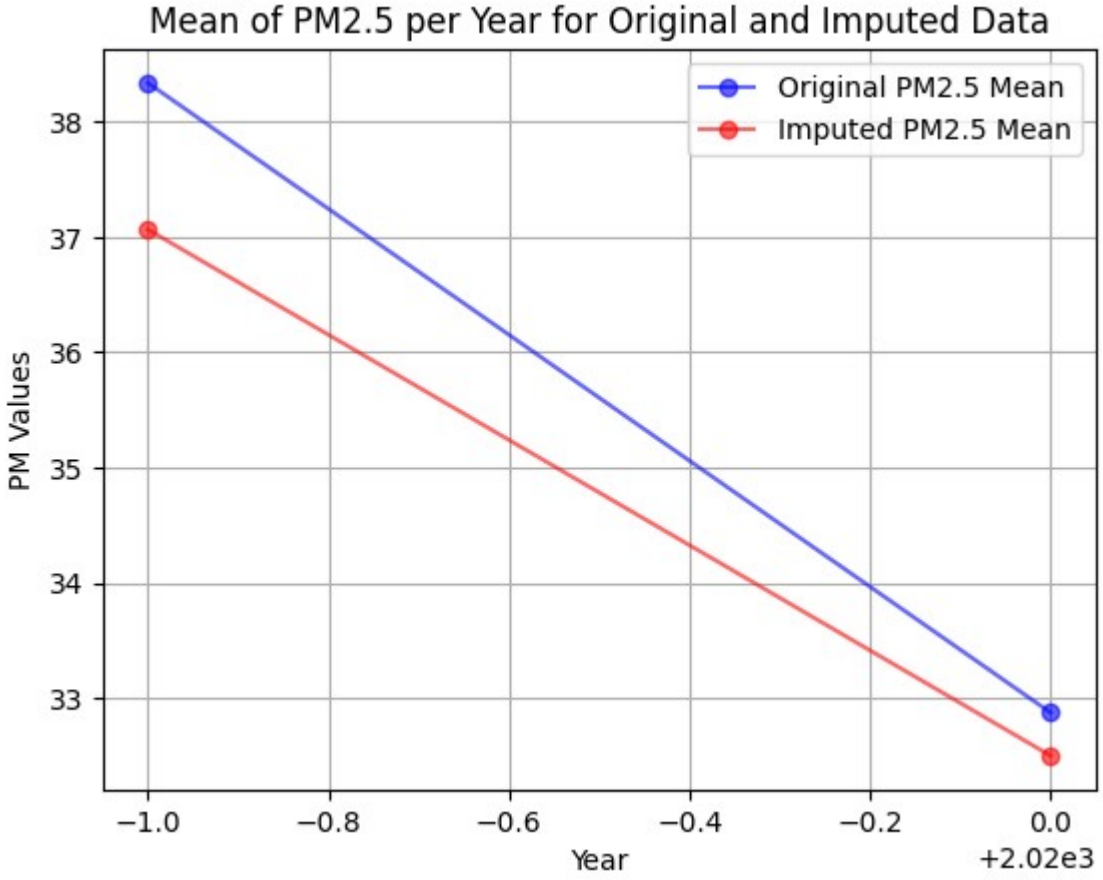


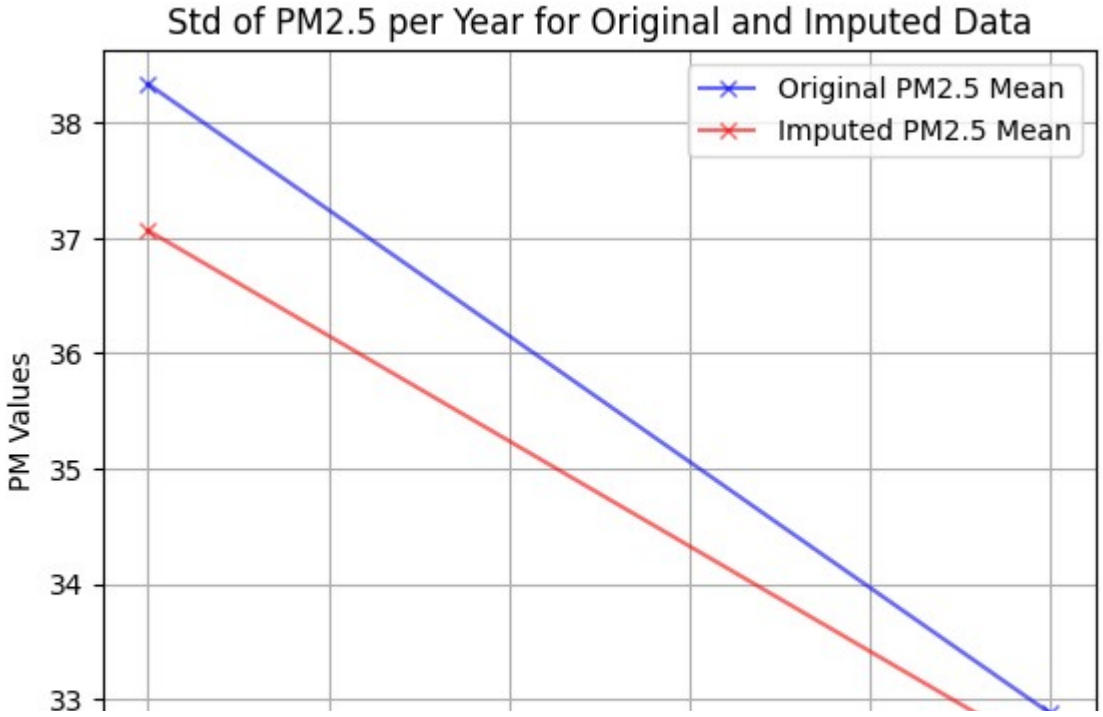


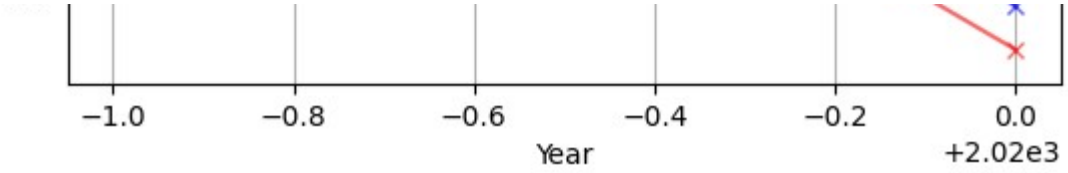



```
1 PM_imputation_stats_calc(Africa_or_df, Africa_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(Africa_or_df, Africa_K_fill, 'PM10')
```

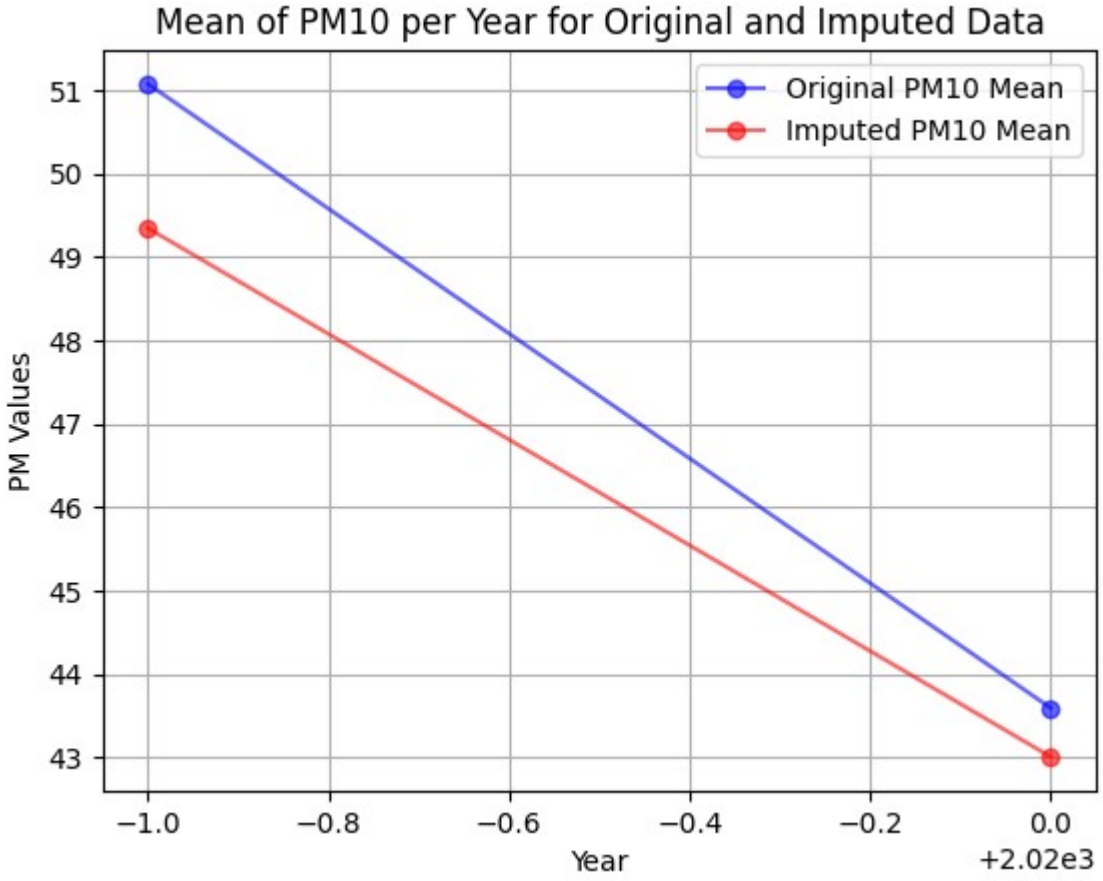
```
Percentage Change of PM2.5 Overall Mean = 1.1964887519314449
Percentage Change of PM2.5 Overall Std = 1.1644457091283062
```

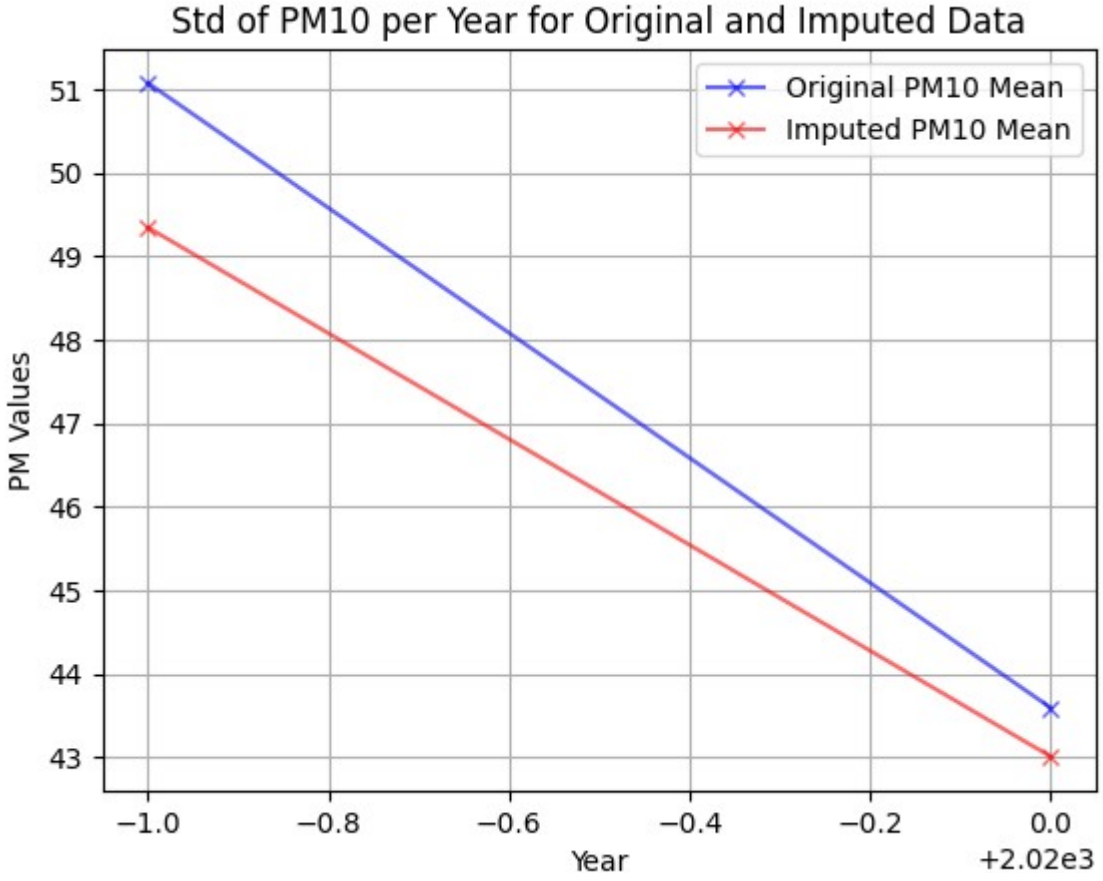






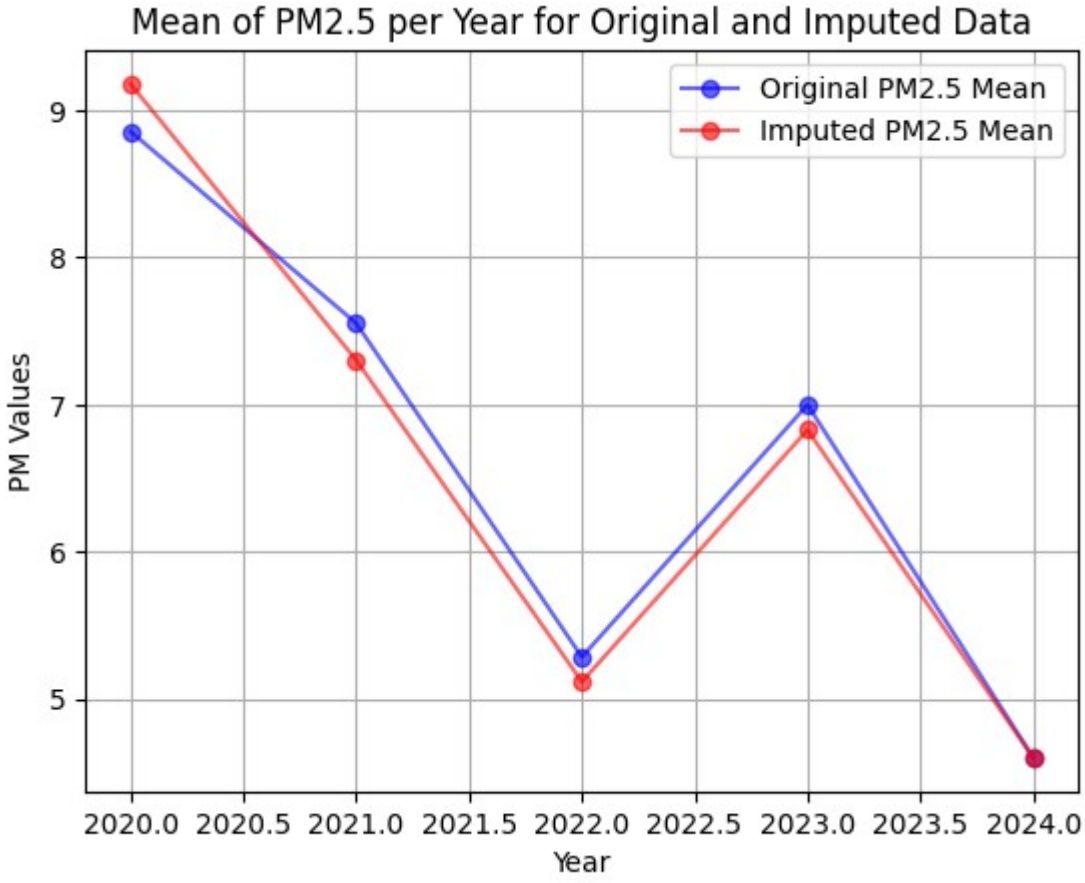

```
Percentage Change of PM10 Overall Mean = 1.3827971946344795
Percentage Change of PM10 Overall Std = 0.9716531546598578
```

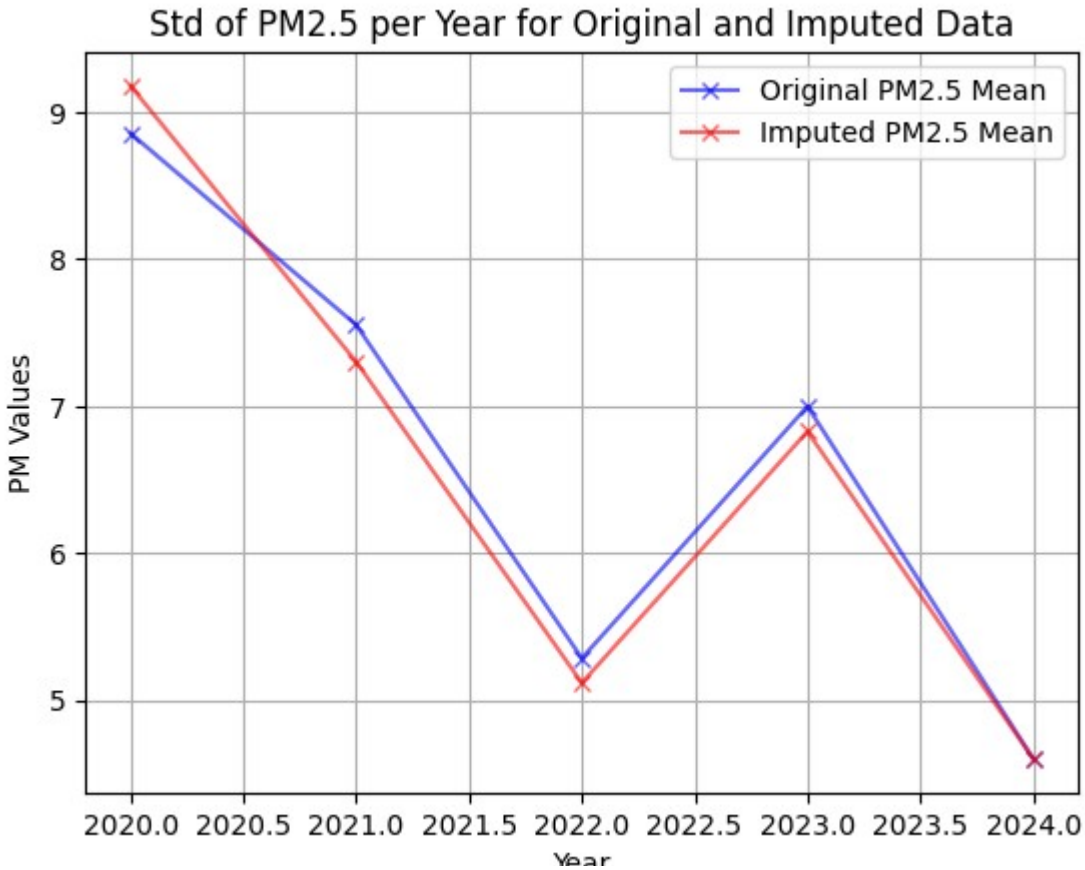




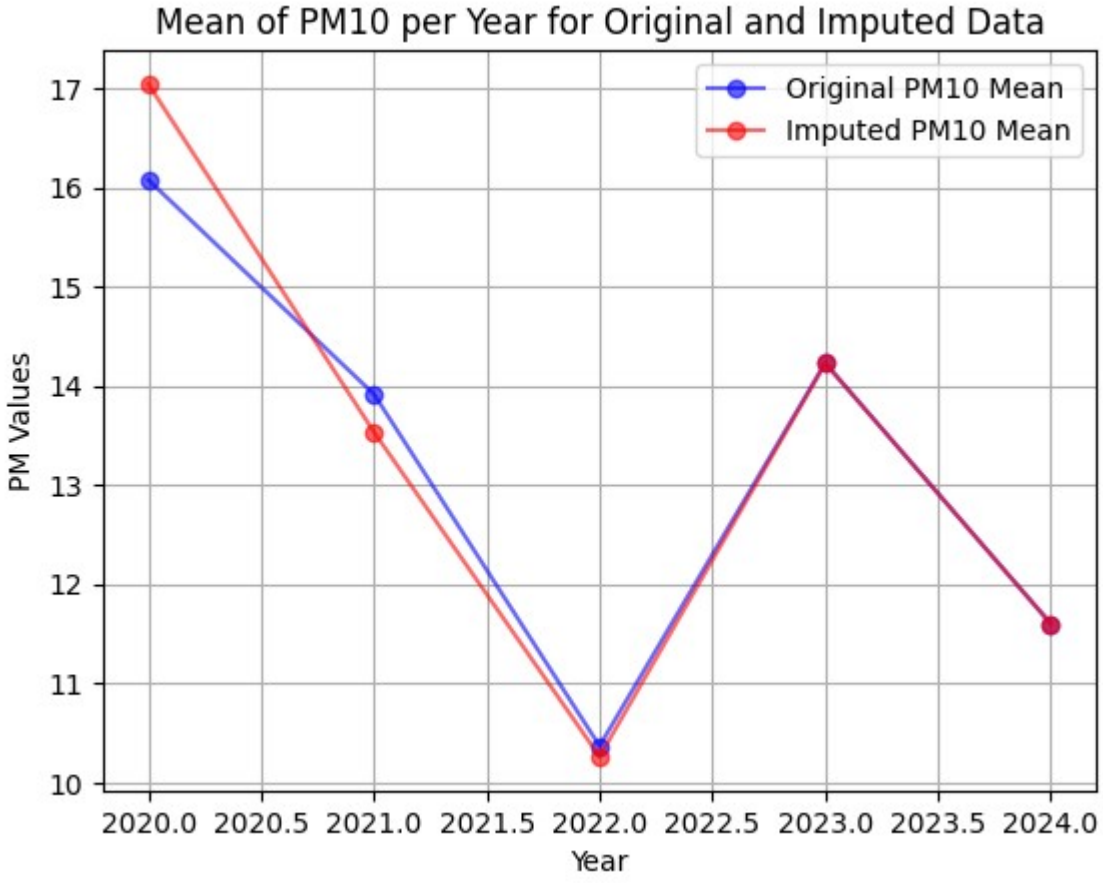
```
1 PM_imputation_stats_calc(Australia_or_df, Australia_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(Australia_or_df, Australia_K_fill, 'PM10')
```

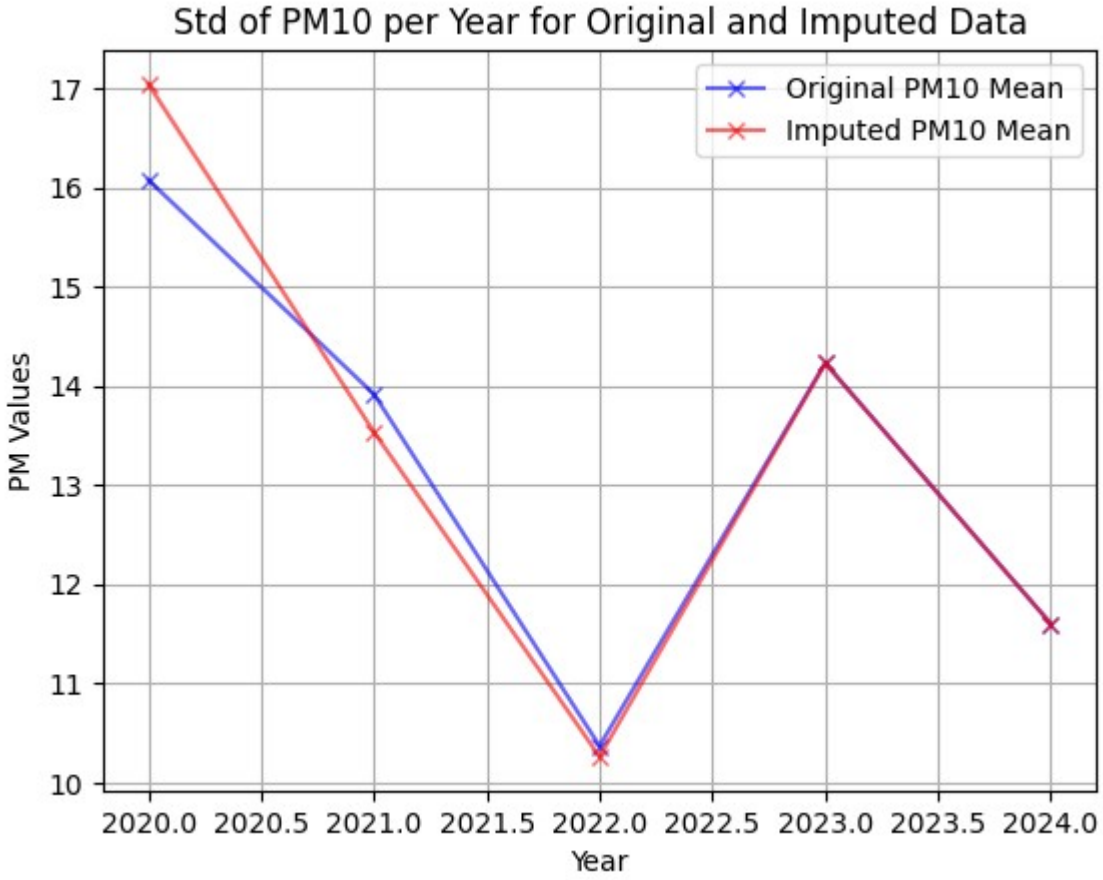
```
Percentage Change of PM2.5 Overall Mean = 0.42956348108133624
Percentage Change of PM2.5 Overall Std = 1.7468248260242891
```





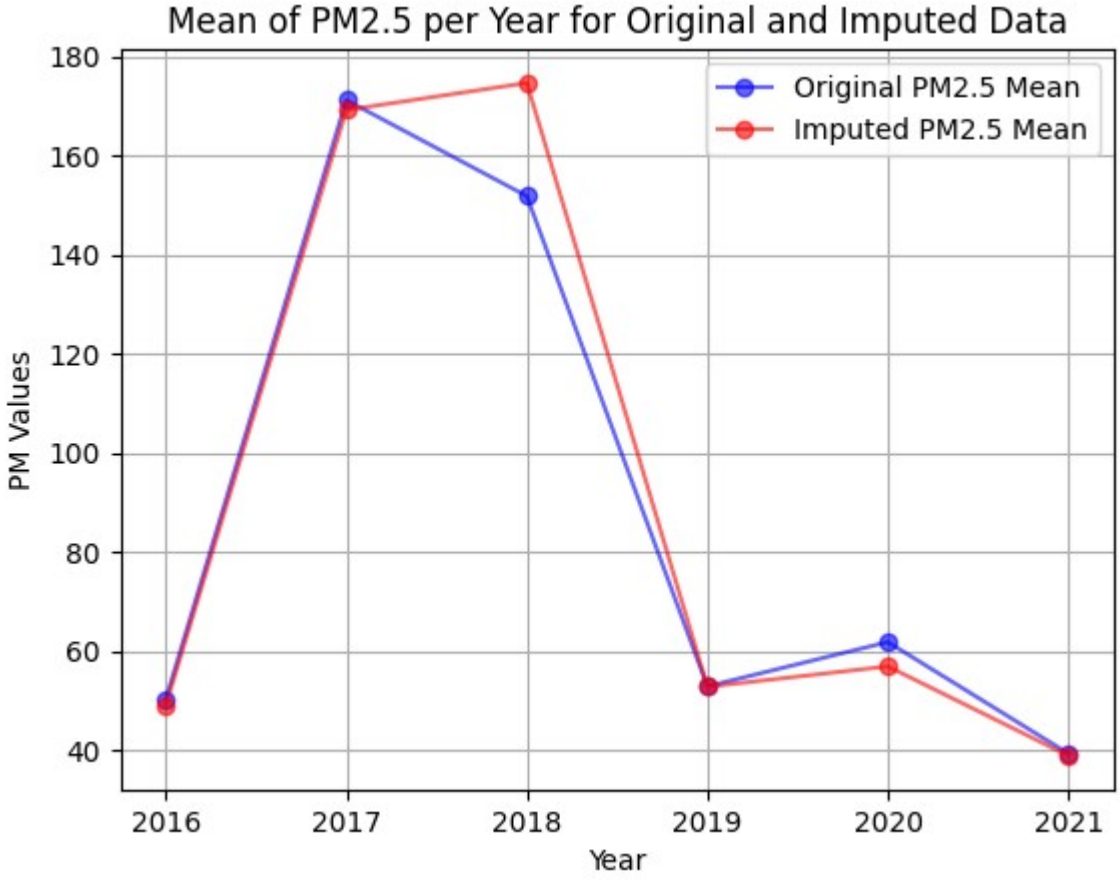
```
Percentage Change of PM10 Overall Mean = 1.3097138800643722
Percentage Change of PM10 Overall Std = 2.4864330336273626
```

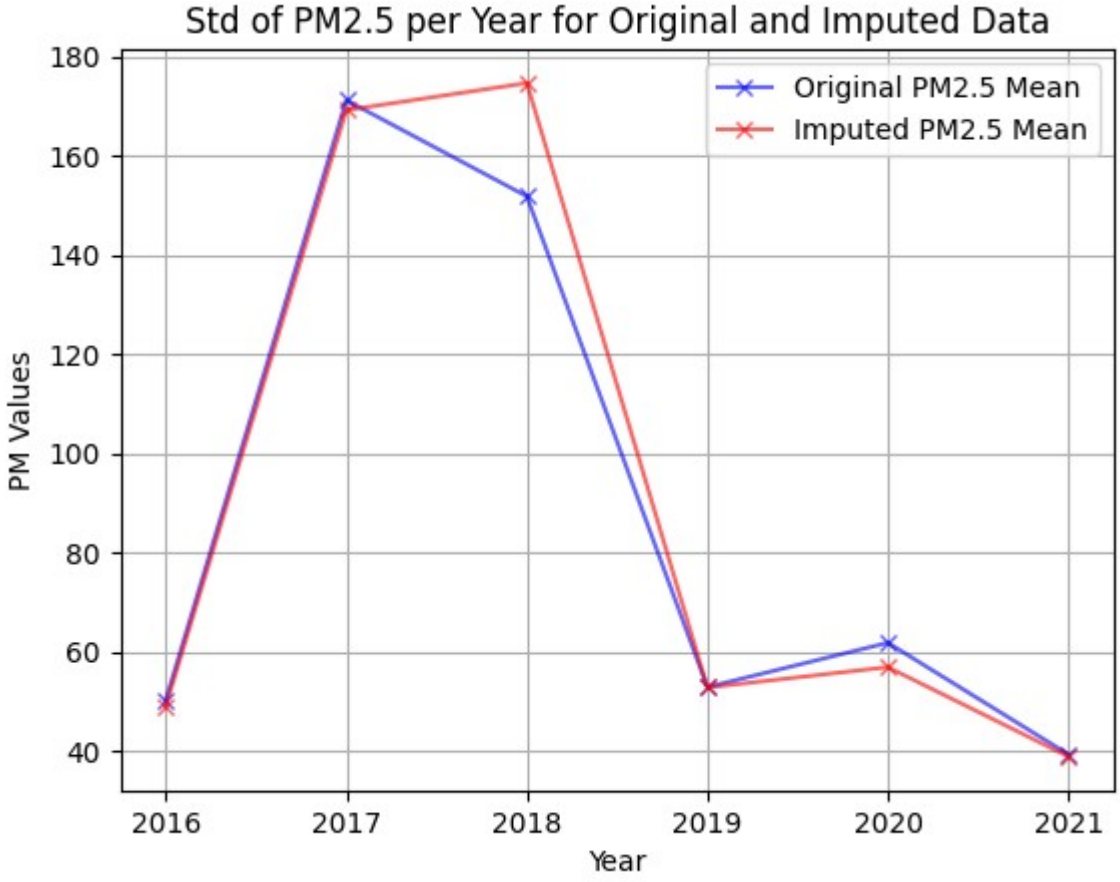





```
1 PM_imputation_stats_calc(Mexico_or_df, Mexico_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(Mexico_or_df, Mexico_K_fill, 'PM10')
```

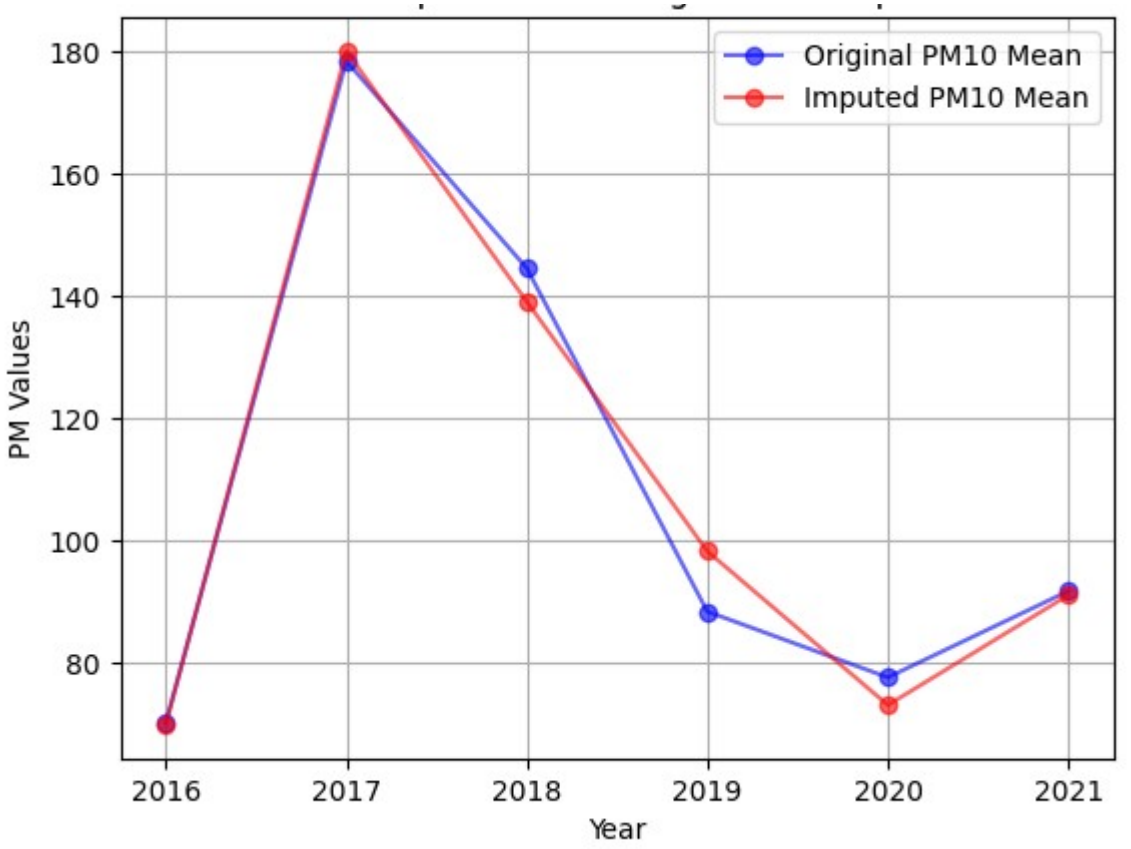
```
Percentage Change of PM2.5 Overall Mean = 2.184979860893057
Percentage Change of PM2.5 Overall Std = 0.9820005967992084
```

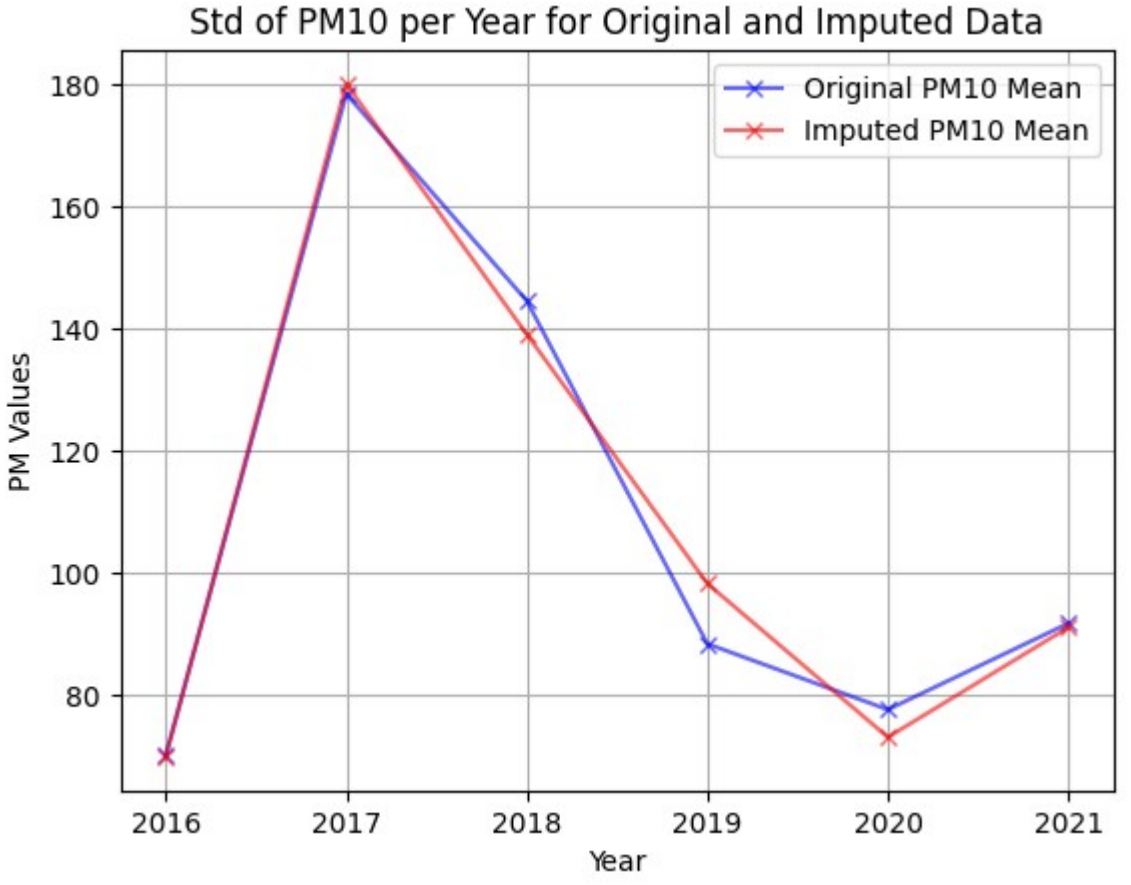





```
Percentage Change of PM10 Overall Mean = 0.7996667665438505
Percentage Change of PM10 Overall Std = 2.329818276887337
```

Mean of PM10 per Year for Original and Imputed Data

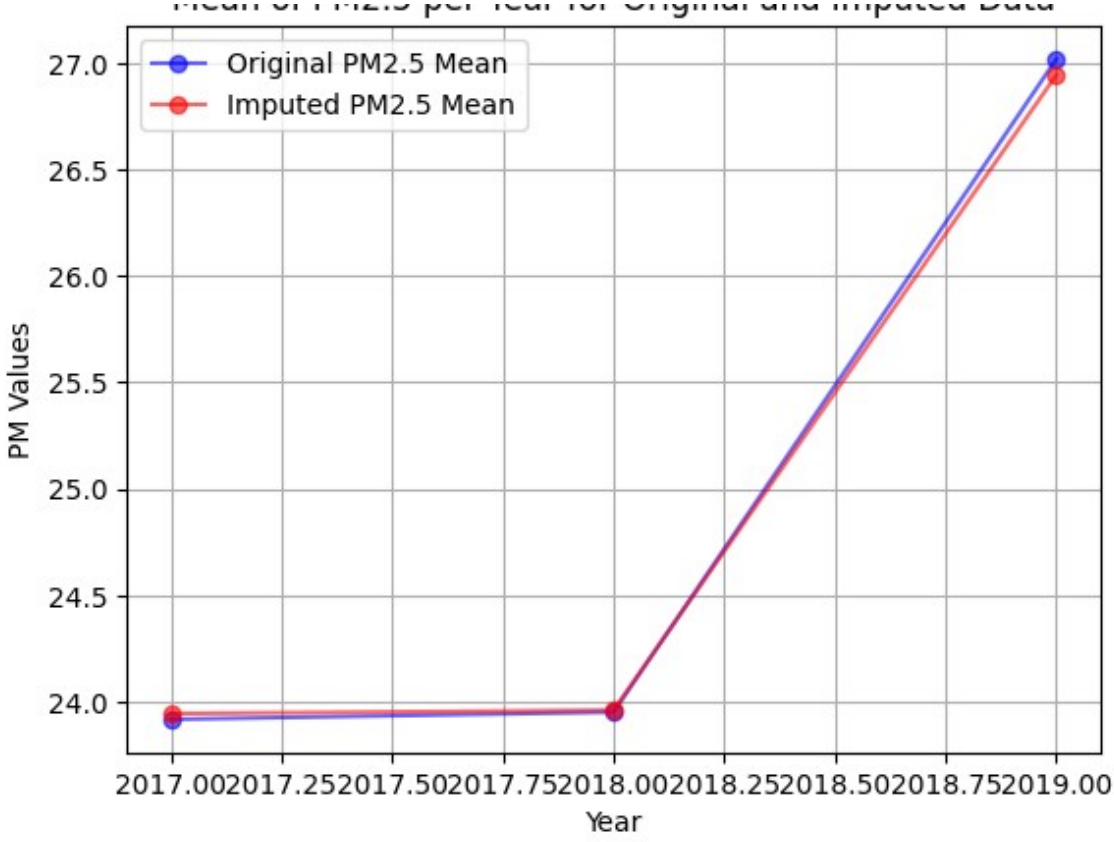


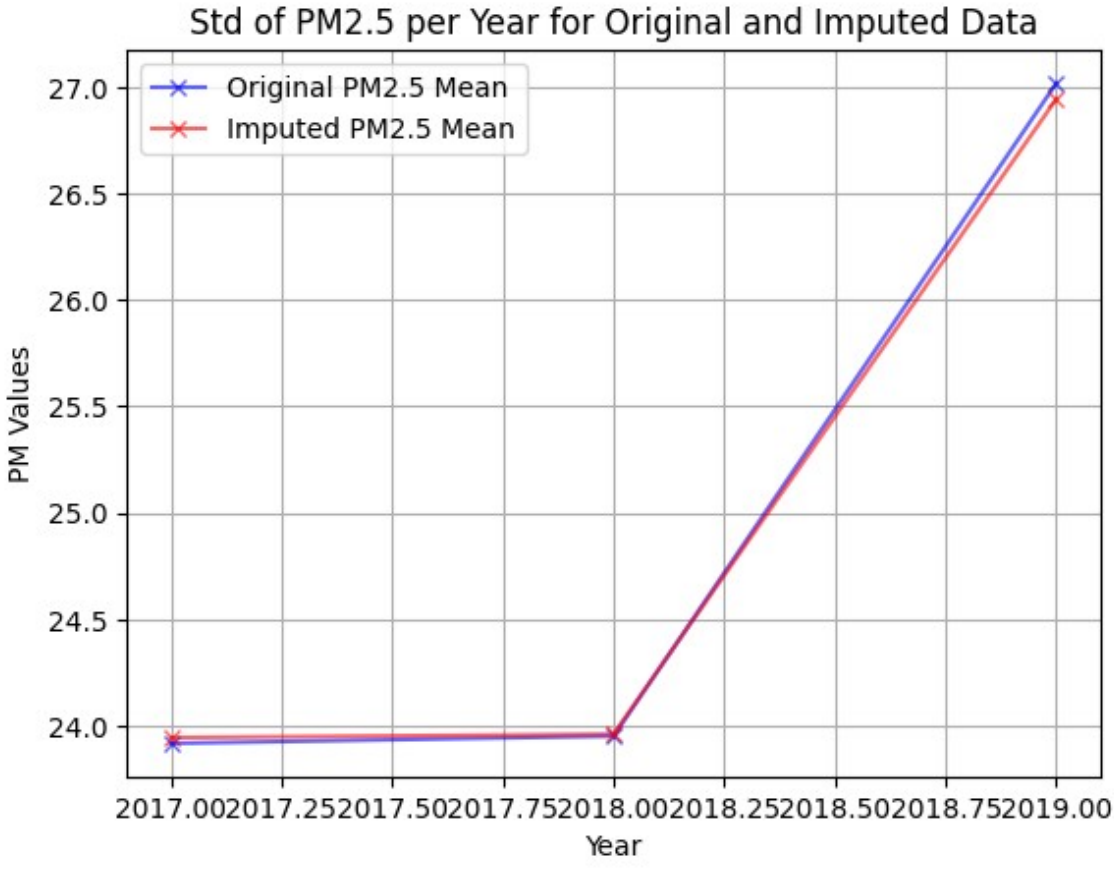



```
1 PM_imputation_stats_calc(Korea_or_df, Korea_K_fill, 'PM2.5')
2 PM_imputation_stats_calc(Korea_or_df, Korea_K_fill, 'PM10')
```

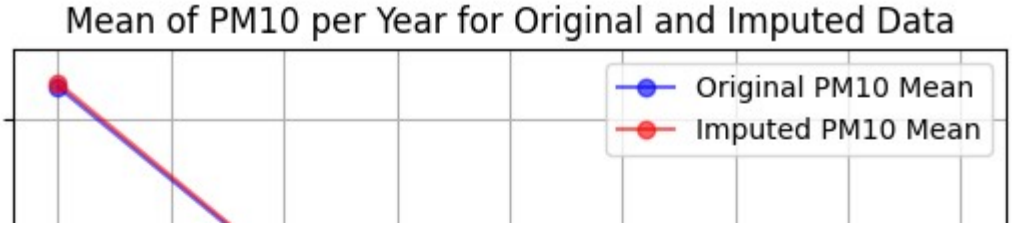
```
Percentage Change of PM2.5 Overall Mean = 0.07101317992289206
Percentage Change of PM2.5 Overall Std = 0.5712802022892692
```

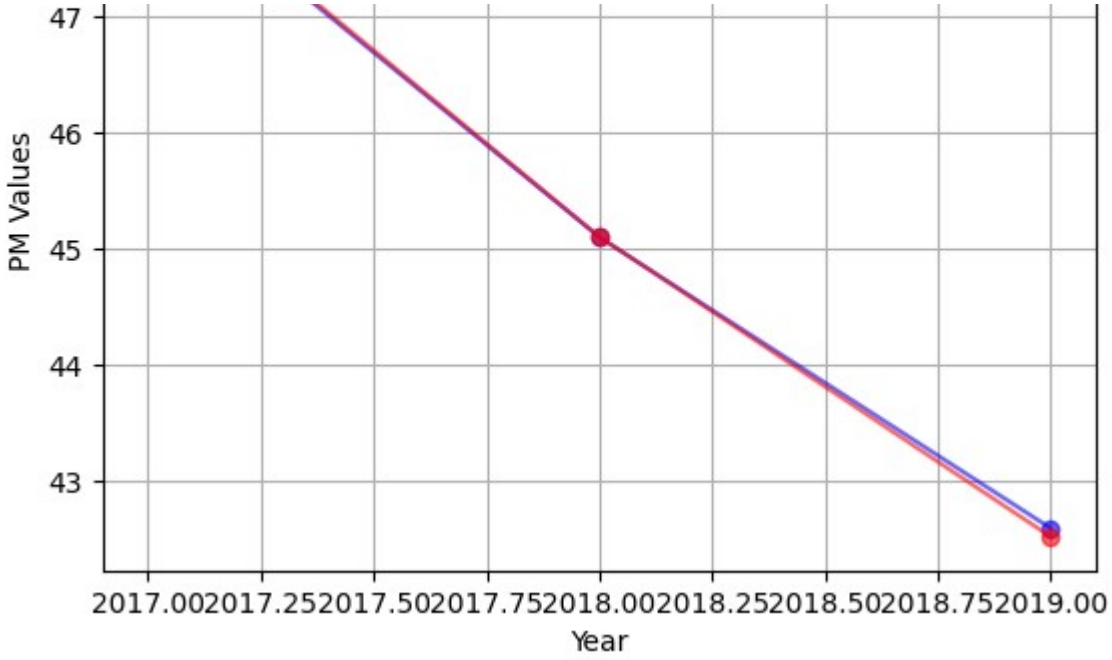
Mean of PM2.5 per Year for Original and Imputed Data

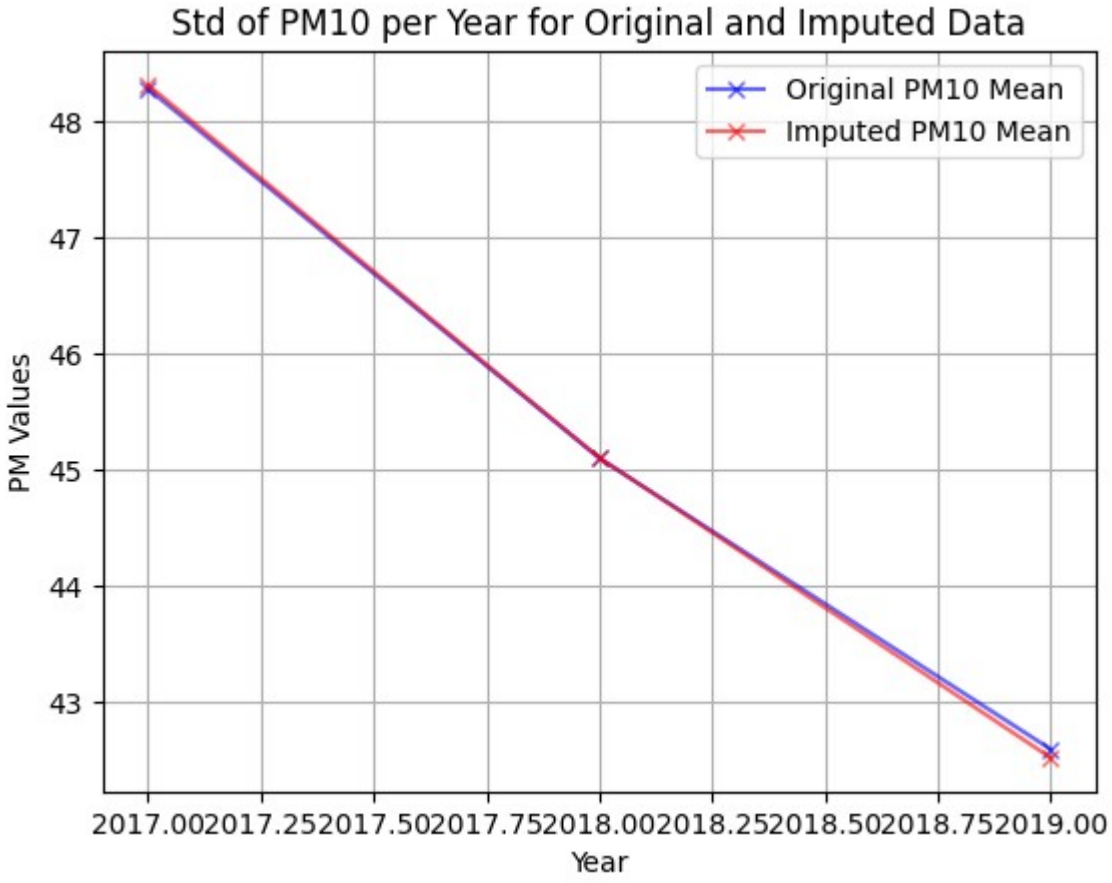





```
Percentage Change of PM10 Overall Mean = 0.11246764906690922
Percentage Change of PM10 Overall Std = 0.5719494283271769
```




47
46
45
44
43
PM Values
2017.002017.252017.502017.752018.002018.252018.502018.752019.00
Year


Std of PM10 per Year for Original and Imputed Data
Original PM10 Mean
Imputed PM10 Mean
48
47
46
45
44
43
PM Values
2017.002017.252017.502017.752018.002018.252018.502018.752019.00
Year