

```
In [ ]: import pandas as pd
from statsmodels.tsa.seasonal import STL
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose
```

1. Wczytanie i przegląd danych

```
In [ ]: df = pd.read_excel('PM_BDG.xlsx', names = ['Date', 'PM25'], skiprows = 5, index_col=0)
df.index = pd.to_datetime(df.index)
```

```
In [ ]: df
```

```
Out[ ]:          PM25
```

Date	PM25
2022-01-01	4.934921
2022-01-02	10.341562
2022-01-03	5.007348
2022-01-04	9.688094
2022-01-05	5.914476
...	...
2022-12-27	2.195314
2022-12-28	4.317930
2022-12-29	5.061757
2022-12-30	5.950761
2022-12-31	8.273009

365 rows × 1 columns

```
In [ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2022-01-01 to 2022-12-31
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   PM25     365 non-null   float64
dtypes: float64(1)
memory usage: 5.7 KB
```

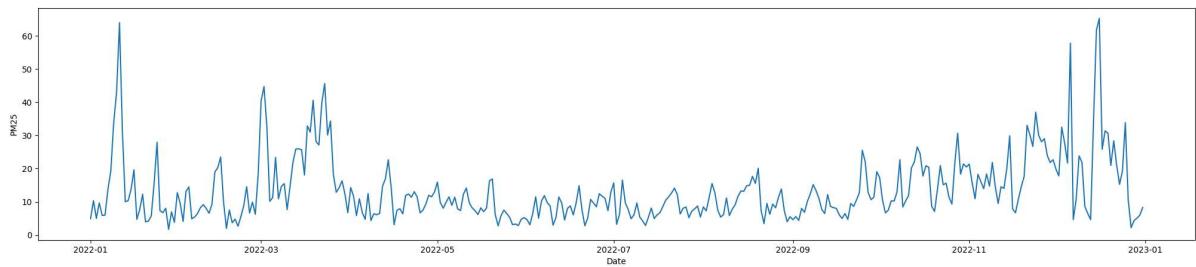
```
In [ ]: df.describe()
```

Out[]:

PM25	
count	365.000000
mean	13.388558
std	9.827103
min	1.705395
25%	6.712919
50%	10.522706
75%	16.328249
max	65.242437

```
In [ ]: fig, ax = plt.subplots(1,1,figsize=(25,5))
sns.lineplot(x=df.index, y=df['PM25'],ax=ax,data=df)

plt.show()
```



2. Feature Engineering

2.1 Rolling Windows

```
In [ ]: stats = ['mean', 'median', 'std']
```

Wybrane statystyki do przedstawienia w oknach to średnia, mediana jako alternatywa dla średniej oraz odchylenie standardowe, które może informować o tym gdzie występują duże wahania amplitudy sygnału.

```
In [ ]: colors = ['red', 'green', 'orange']
windows = [3, 7, 14, 30]
fig, ax = plt.subplots(len(windows), 1, figsize = [25, 14])
for j, window in enumerate(windows):

    sns.lineplot(data = df, x = df.index, y = 'PM25', ax = ax[j], color = 'grey', )
    ax[j].set_title(f'Okno = {window}')
    ax[j].grid()
    # ax[i].set_xlim(np.min(data.index), np.max(data.index))
    ax[j].set_ylabel('PM2.5 [ug/m3]')

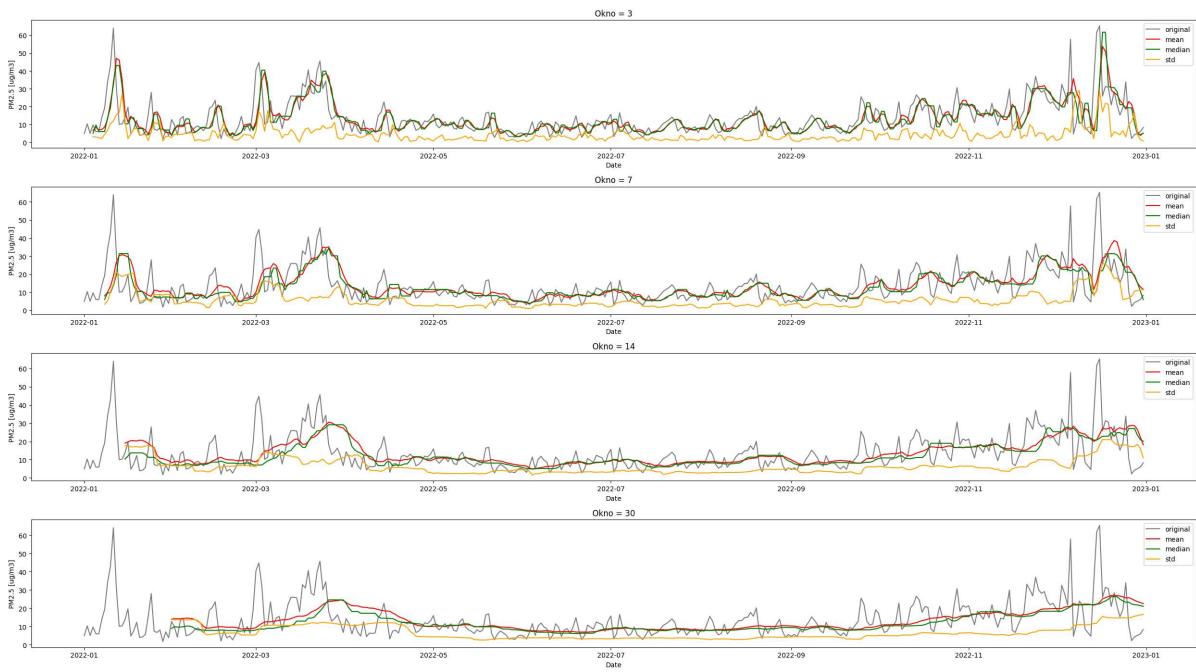
    rolling_df = df.rolling(window=window).agg(stats).shift(1)

    for i, stat in enumerate(stats):

        sns.lineplot(data = rolling_df, x = rolling_df.index, y = ('PM25', stat), ax = ax[j], color = colors[i])
        ax[j].set_title(f'Okno = {window}')
        ax[j].grid()
```

```
# ax[i].set_xlim(np.min(data.index), np.max(data.index))
ax[j].set_ylabel('PM2.5 [ug/m3]')

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



Wnioski:

- Mediana dla wszystkich wykresów wydaje się być gorszą statystyką do wyświetlanego niż średnia, bo w sygnałach mediany występują schodki, średnia jest bardziej wygładzona.
- Średnia i mediana dla okna 3 wyglądają jak wykres lagu z lekkim wygładzeniem sygnału, odchylenie standardowe wydaje się dobrze pokazywać zaburzenia amplitudy sygnału.
- Dla okien 14 i 30 wszystkie pomiary mocno wygładzają sygnał. Przy oknie 30 średnia i mediana działają trochę jak trend. Dodatkowo te okna powodują większe braki na początku zbioru danych.

Po analizie wykresów do zbioru dołączone zostaną okna 3 i 7 dla średniej oraz 3 dla odchylenia standardowego.

```
In [ ]: df['PM_25_rolling_mean_3'] = df.rolling(window=3).agg('mean').shift(1)[ 'PM25']
df['PM_25_rolling_mean_7'] = df.rolling(window=7).agg('mean').shift(1)[ 'PM25']
df['PM_25_rolling_std_3'] = df.rolling(window=3).agg('std').shift(1)[ 'PM25']
```

2.2 Expanding Windows

```
In [ ]: expanding_df = df.expanding().agg(stats)
colors = ['red', 'green', 'orange']

fig, ax = plt.subplots(1, 1, figsize = [25, 5])

sns.lineplot(data = df, x = df.index, y = 'PM25', ax = ax, color = 'grey', label =
ax.set_title(f'Okna typu expanding')
ax.grid()
# ax[i].set_xlim(np.min(data.index), np.max(data.index))
ax.set_ylabel('PM2.5 [ug/m3]')

rolling_df = df[ 'PM25'].expanding().agg(stats).shift(1)
```

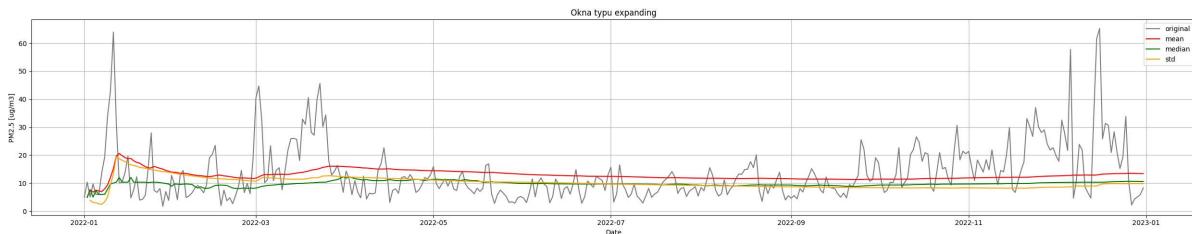
```

for i, stat in enumerate(stats):

    sns.lineplot(data = rolling_df, x = rolling_df.index, y = stat, ax = ax, color
    # kolor linii dla poszczególnych statystyk

plt.tight_layout()
plt.show()

```



w tym przypadku okna typu rolling wydają się nie nieść przydatnych informacji, gdyż od pewnego poziomu są one niemal stałe. Gdyby jednak mogły się przydać do zbioru dołączone zostanie okno expanding dla wartości średniej.

```
In [ ]: df['PM_25_expanding_mean'] = df['PM25'].expanding().agg('mean').shift(1)
```

2.3 Okna czasowe zagnieżdżone

```

In [ ]: nested_windows = [(7, 14), (14, 7), (3, 30), (30,3),(7,3)]

fig, ax = plt.subplots(len(nested_windows), 1, figsize = [25, 16])

i = 0
for outer_window, inner_window in nested_windows:

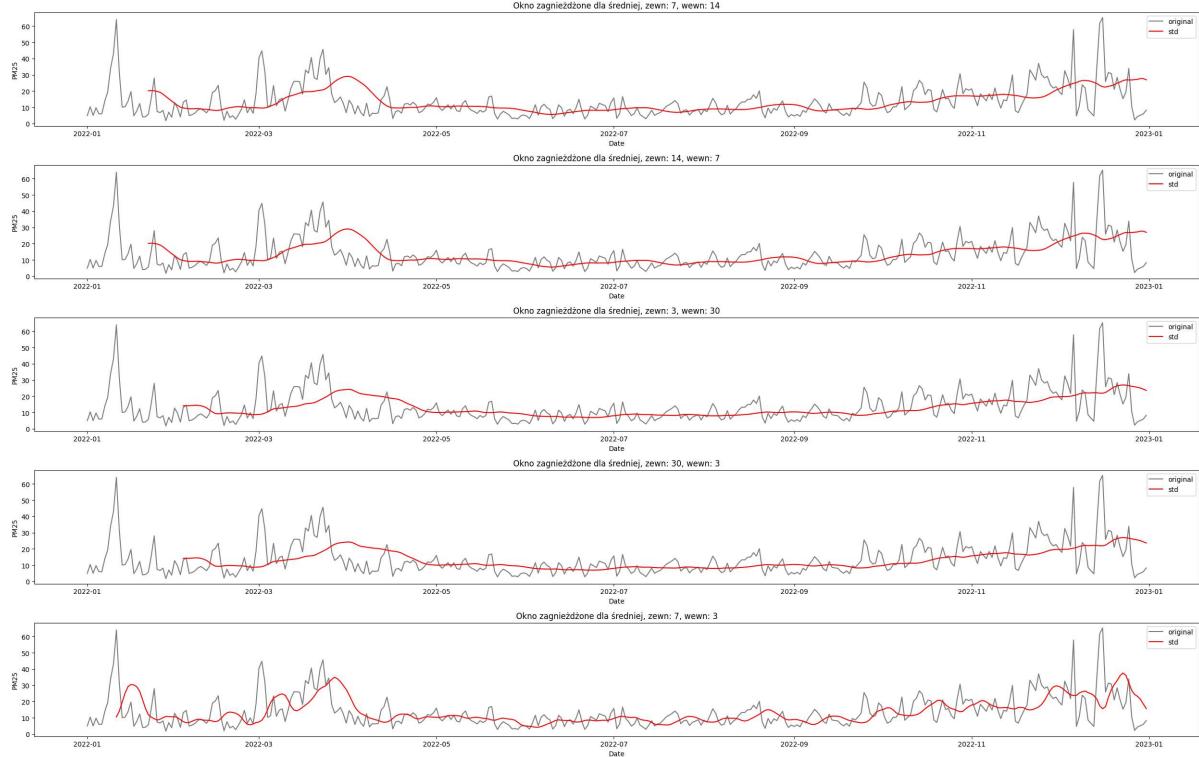
    sns.lineplot(data = df, x = df.index, y = 'PM25', ax = ax[i], color = 'grey', )

    outer_stats = df['PM25'].rolling(window=outer_window).agg('mean').shift(1)
    nested_df = outer_stats.rolling(window=inner_window).agg('mean').shift(1)

    sns.lineplot(data = nested_df, ax = ax[i], color = 'red', label = stat)
    ax[i].set_title(f'Okno zagnieżdżone dla średniej, zewn: {outer_window}, wewn: {inner_window} ')
    i += 1

plt.tight_layout()
plt.show()

```



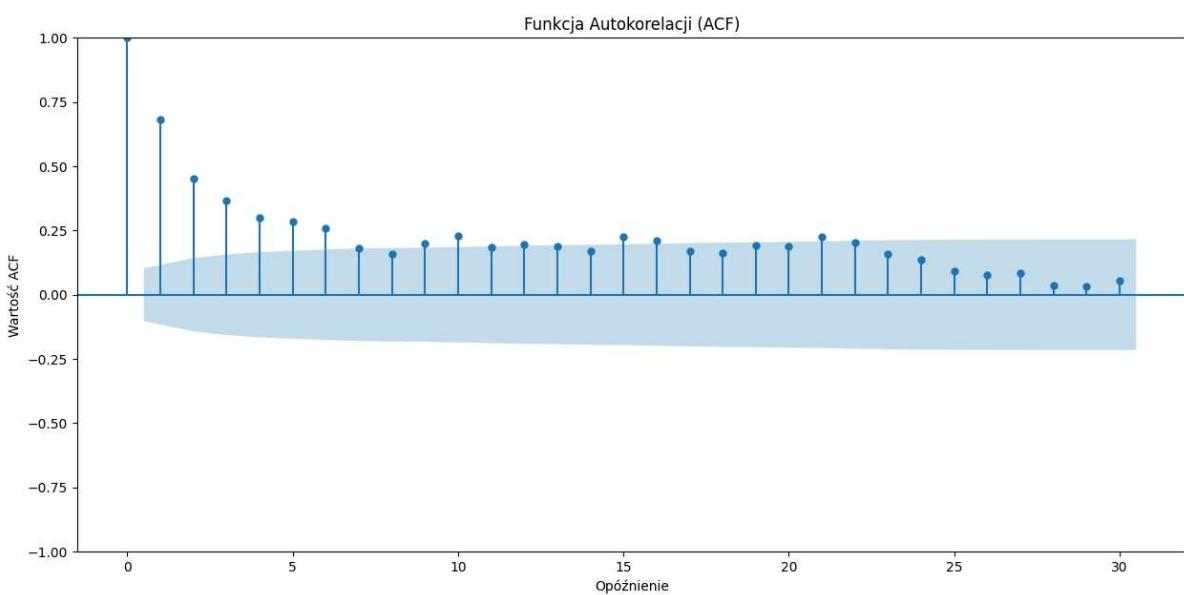
Okna zagnieżdżone o różnych parametrach dla średniej powodują bardziej przesunięte i wygładzone sygnały. Z tego powodu nie zostaną dodane do danych.

2.4 Wartości opóźnione bazując na ACF

```
In [ ]: fig,ax = plt.subplots(1,1, figsize = (15,7))

plot_acf(df['PM25'], lags=30, ax=ax)
plt.title("Funkcja Autokorelacji (ACF)")
plt.xlabel("Opóźnienie")
plt.ylabel("Wartość ACF")
```

```
Out[ ]: Text(0, 0.5, 'Wartość ACF')
```



Na wykresie autokorelacji widoczne są lokalne maksima dla opóźnień 10, 15 i 21, jednak wartość korelacji wychodzi niewiele poza granicę istotności statystycznej. Z tego powodu

lepiej wybrać lag=1, dla którego korelacja z sygnałem jest najwyższa (nie licząc zerowego opóźnienia).

```
In [ ]: df['PM25_lag_1'] = df['PM25'].shift(1)
```

2.5 Wartości wynikające z daty

```
In [ ]: df["day_of_week"] = df.index.dayofweek
df["month"] = df.index.month
df["day_of_year"] = df.index.dayofyear
df['is_weekend'] = df['day_of_week'].apply(lambda day: 1 if day > 4 else 0)
```

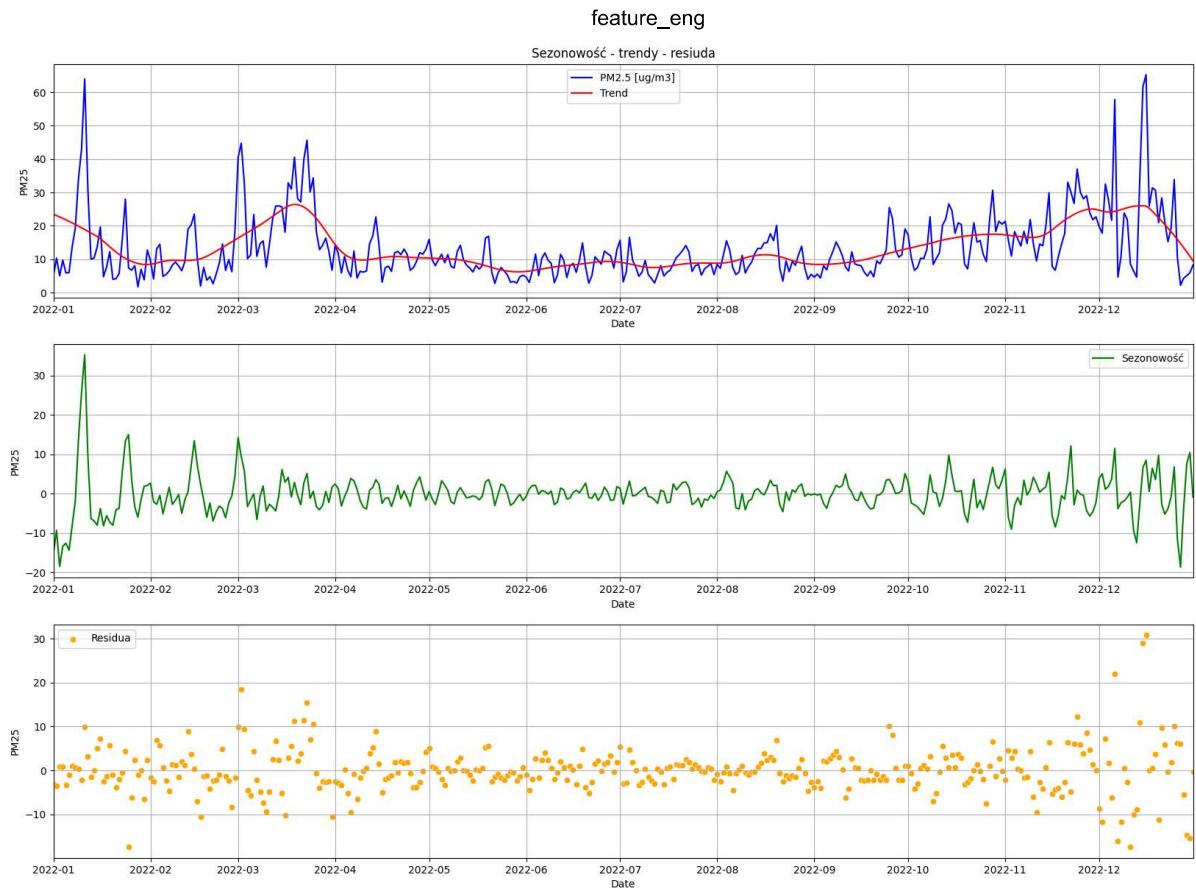
```
In [ ]: result = STL(df['PM25'], period = 14, seasonal = 5).fit()
df['Trend'], df['Seasonal'], df['Resid'] = result.trend, result.seasonal, result.resid

fig, axes = plt.subplots(3, 1, figsize = [20, 14])
sns.lineplot(x = df.index, y = df['PM25'], ax = axes[0], color = 'blue', label = 'PM25')
sns.lineplot(x = df.index, y = df['Trend'], ax = axes[0], color = 'red', label = 'Trend')
axes[0].set_ylabel('PM25')
axes[0].grid()
axes[0].set_title('Sezonowość - trendy - reszta')
axes[0].set_xlim(np.min(df.index), np.max(df.index))

sns.lineplot(x = df.index, y = df['Seasonal'], ax = axes[1], color = 'green', label = 'Seasonal')
axes[1].set_ylabel('PM25')
axes[1].grid()
axes[1].set_xlim(np.min(df.index), np.max(df.index))

sns.scatterplot(x = df.index, y = df['Resid'], ax = axes[2], color = 'orange', label = 'Resid')
axes[2].set_ylabel('PM25')
axes[2].grid()
axes[2].set_xlim(np.min(df.index), np.max(df.index))
```

```
Out[ ]: (18993.0, 19357.0)
```



Cieźko o jeden wyraźny trend sezonowy w prezentowanych danych. Przypuszczalnie roczny trend byłby widoczny najlepiej, jednak dane dostępne są jedynie z jednego roku. Po testach różnych wartości period oraz seasonal, wydaje się, że względnie dobre rezultaty dają parametry:

period = 14

seasonal = 5

Widoczne jest na wykresie resztu, że najlepiej STL poradził sobie w okresie letnim, gdzie oscylują one najbliższej wartości 0.

Zbiór danych uzyskanych w ramach feature engineeringu:

In []: df

Out[]: PM25 PM_25_rolling_mean_3 PM_25_rolling_mean_7 PM_25_rolling_std_3 PM_25_expan

Date	PM25	PM_25_rolling_mean_3	PM_25_rolling_mean_7	PM_25_rolling_std_3	PM_25_expan
2022-01-01	4.934921	NaN	NaN	NaN	NaN
2022-01-02	10.341562	NaN	NaN	NaN	NaN
2022-01-03	5.007348	NaN	NaN	NaN	NaN
2022-01-04	9.688094	6.761277	NaN	3.100829	
2022-01-05	5.914476	8.345668	NaN	2.909474	
...
2022-12-27	2.195314	21.263458	21.348769	11.735045	
2022-12-28	4.317930	15.560667	18.668857	16.395281	
2022-12-29	5.061757	5.714974	15.232136	4.388264	
2022-12-30	5.950761	3.858334	12.943590	1.487463	
2022-12-31	8.273009	5.110149	11.616591	0.817490	

365 rows × 13 columns

3. Przedstaw uzyskany zbiór cech na w przejrzysty sposób na wykresach i opisz co można na ich podstawie zinterpretować.

```
In [ ]: col_list = ['PM_25_rolling_mean_3', 'PM_25_rolling_mean_7',
                 'PM_25_rolling_std_3', 'PM_25_expanding_mean', 'PM25_lag_1', 'Trend',
                 'Seasonal', 'Resid']

fig, ax = plt.subplots(len(col_list), 1, figsize = [25, 35])

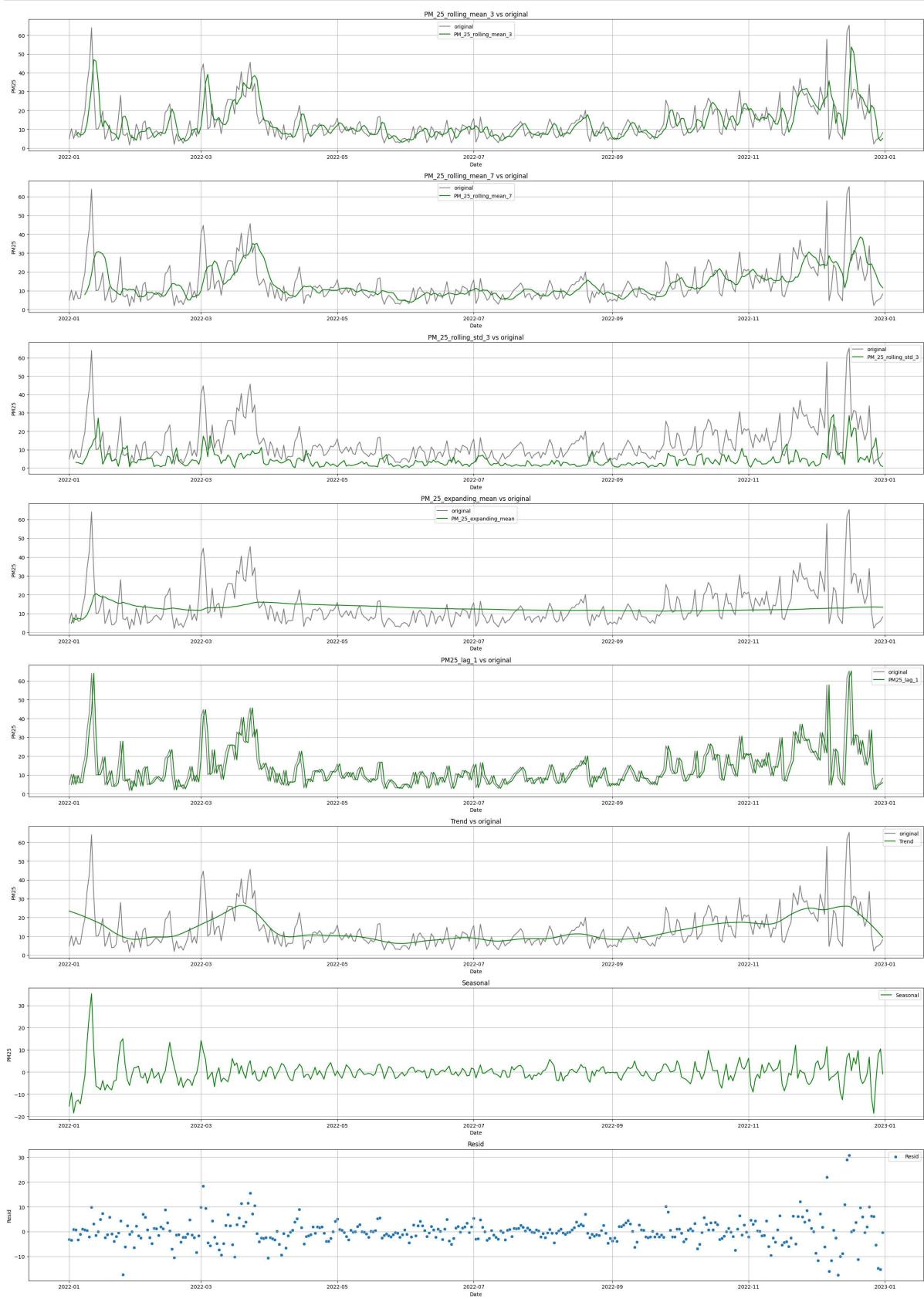
for i, col in enumerate(col_list):
    if(col not in ['Seasonal', 'Resid']):
        sns.lineplot(data = df, x = df.index, y = 'PM25', ax = ax[i], color =
                      ax[i].set_title(col + ' vs original')
    else:
        ax[i].set_title(col)

    if col == 'Resid':
        sns.scatterplot(x = df.index,y=df[col], ax = ax[i], label = col)
    else:
        sns.lineplot(x = df.index, y = df[col], ax = ax[i], color = 'green',
                     ax[i].set_ylabel('PM25')
```

```

        ax[i].grid()

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



Interpretacje dodawanych cech zostały umieszczone bezpośrednio pod odpowiadającym im kodem.