In [ ]:
```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
import numpy as np
from statsmodels.graphics.tsaplots import plot_pacf
from geopy.distance import geodesic
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import TimeSeriesSplit
import matplotlib.pyplot as plt
from collections import defaultdict
```

Projekt - Predykcja zanieczyszczenia PM2.5 - Anna Kaniowska 407334

Część 1 - Przygotowanie danych i inżynieria cech

Przegląd danych

In [ ]:
```
df = pd.read_excel("data/2019_PM25_1g.xlsx", header=1, index_col=0)
```

In [ ]:
```
df = df.drop(
    ["Wskaźnik", "Czas uśredniania", "Jednostka", "Kod stanowiska"]
)  # PM2.5, 1g, ug/m3, <Kod stacji>-PM2.5-1g
df.index = df.index.rename("Date")
df.index = pd.to_datetime(df.index)
df.head()
```

Out[ ]:

Date	DsDusznikMOB	DsJaworMOB	DsJelGorOgin	DsWrocAlWisn	DsWrocWybCon	KpByd
2019-01-01 01:00:00	33.4053	51.3878	118.773	102.09	107.061	
2019-01-01 02:00:00	13.8028	28.4995	110.064	63.6111	55.9187	
2019-01-01 03:00:00	9.94056	11.1206	107.941	48.354	41.3488	
2019-01-01 04:00:00	6.75889	5.57358	94.5489	34.6621	29.8697	
2019-01-01 05:00:00	7.88722	6.56224	67.88	14.287	17.6	

5 rows × 63 columns

In [ ]:
```
# Zdobycie informacji o lokalizacji stacji - plik dostępny na stronie GIOŚ

df_xy = pd.read_excel(
    "data/Metadane oraz kody stacji i stanowisk pomiarowych.xlsx", index_col=0
)
locs = df_xy[df_xy["Kod stacji"].isin(df.columns)].set_index("Kod stacji")
locs = locs[["WGS84 ϕ N", "WGS84 λ E"]]
locs = locs.rename(columns={"WGS84 ϕ N": "X", "WGS84 λ E": "Y"})
locs.head()
```

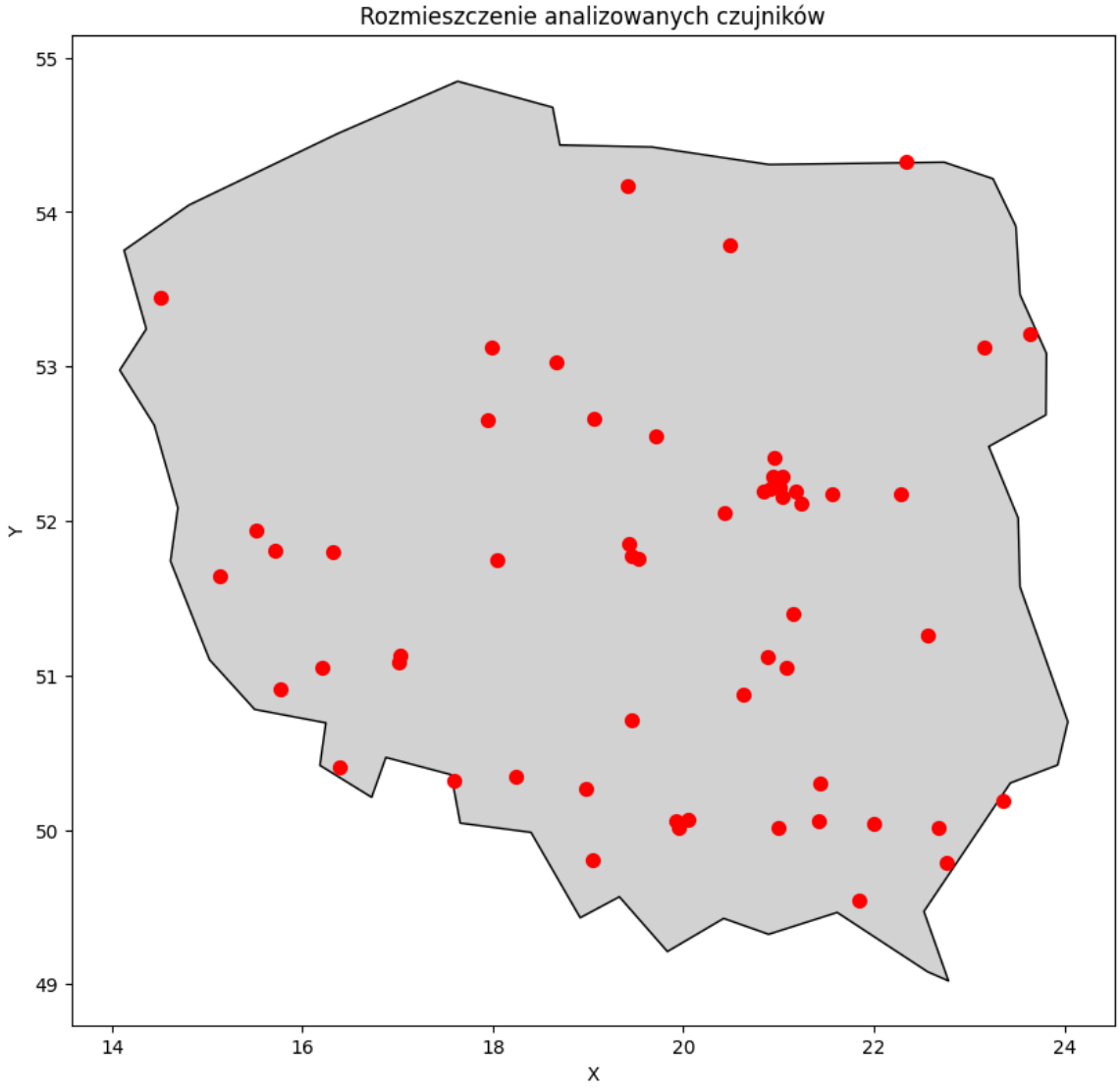
Out[ ]:

Kod stacji	X	Y
DsDusznikMOB	50.402645	16.393319
DsJaworMOB	51.049212	16.202317
DsJelGorOgin	50.913433	15.765608
DsWrocAlWisn	51.086225	17.012689
DsWrocWybCon	51.129378	17.029250

In [ ]:
```
# Wizualizacja czujników

geometry = [Point(xy) for xy in zip(locs["Y"], locs["X"])]
gdf = gpd.GeoDataFrame(locs, geometry=geometry, crs="EPSG:4326")

poland_boundary = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
poland_boundary = poland_boundary[poland_boundary.name == "Poland"]

fig, ax = plt.subplots(figsize=(10, 10))
ax.set_aspect("equal")
poland_boundary.plot(ax=ax, color="lightgrey", edgecolor="black")
```

```
gdf.plot(ax=ax, color="red", markersize=50)
ax.set(title="Rozmieszczenie analizowanych czujników", xlabel="X", ylabel="Y")
plt.show()
```

```
/tmp/ipykernel_2672373/706845106.py:6: FutureWarning: The geopandas.dataset mod
ule is deprecated and will be removed in GeoPandas 1.0. You can get the origina
l 'naturalearth_lowres' data from https://www.naturalearthdata.com/downloads/11
0m-cultural-vectors/.
  poland_boundary = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
```



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

Out[ ]:

	DsDusznikMOB	DsJaworMOB	DsJelGorOgin	DsWrocAlWisn	DsWrocWybCon	KpBydP
count	8646.00000	8711.00000	8498.000	8707.0000	8646.0	791
unique	8315.00000	8641.00000	8199.000	8630.0000	8119.0	125
top	5.68639	8.91834	9.215	12.7177	6.4	1
freq	4.00000	2.00000	3.000	2.0000	23.0	4

4 rows × 63 columns

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

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8760 entries, 2019-01-01 01:00:00 to 2020-01-01 00:00:00
Data columns (total 63 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   DsDusznikMOB      8646 non-null   object
 1   DsJaworMOB        8711 non-null   object
 2   DsJelGorOgin      8498 non-null   object
 3   DsWrocAlWisn      8707 non-null   object
 4   DsWrocWybCon      8646 non-null   object
 5   KpBydPlPozna      7911 non-null   object
 6   KpMogiNowMOB      8319 non-null   object
 7   KpToruDziewu      7213 non-null   object
 8   KpWloclOkrze      8713 non-null   object
 9   LbLubObywate      8412 non-null   object
 10  LbNaleczow        8191 non-null   object
 11  LdLodzCzerni      8646 non-null   object
 12  LdLodzGdansk      8083 non-null   object
 13  LdZgieMielcz      8268 non-null   object
 14  LuNowaSolMOB      8164 non-null   object
 15  LuWsKaziWiel      8263 non-null   object
 16  LuZarySzyman      8496 non-null   object
 17  LuZielKrotka      8138 non-null   object
 18  MpKrakAlKras      8730 non-null   object
 19  MpKrakBujaka      8660 non-null   object
 20  MpKrakBulwar      8683 non-null   object
 21  MpTarRoSitko      8748 non-null   object
 22  MzKonJezMos       8584 non-null   object
 23  MzLegZegrzyn      8637 non-null   object
 24  MzMinMazKaziMOB   8623 non-null   object
 25  MzOtwoBrzozo      8703 non-null   object
 26  MzPiasPulask      8532 non-null   object
 27  MzPlocMiReja      8683 non-null   object
 28  MzRadTochter      8609 non-null   object
 29  MzSiedKonars      8325 non-null   object
 30  MzWarAlNiepo      8555 non-null   object
 31  MzWarBajkowa      8734 non-null   object
 32  MzWarChrosci      8711 non-null   object
 33  MzWarKondrat      8709 non-null   object
 34  MzWarTolstoj      8306 non-null   object
 35  MzWarWokalna      8209 non-null   object
 36  MzZyraRoosev      8528 non-null   object
 37  OpKKozBSmial      8170 non-null   object
 38  OpPrudPodgor      8632 non-null   object
 39  PdBialWaszyn      8543 non-null   object
 40  PdBorsukowiz      8716 non-null   object
 41  PdSuwPulaskp      8487 non-null   object
 42  PkDebiGrottg      8449 non-null   object
 43  PkHorZdrParkMOB   8087 non-null   object
 44  PkJarosPruch      8717 non-null   object
 45  PkMielBierna      8377 non-null   object
 46  PkPrzemGrunw      8245 non-null   object
 47  PkRymZdrPark      8536 non-null   object
 48  PkRzeszPilsu      8749 non-null   object
 49  PmGdaLeczk08      4112 non-null   object
 50  SkKielTargow      8341 non-null   object
 51  SkSkarZielnaMOB   7959 non-null   object
 52  SkStaraZlota      8719 non-null   object
 53  SlBielPartyz      8740 non-null   object
 54  SlKatoKossut      8191 non-null   object
```

```
 55  SlZlotPotLes      8666 non-null   object
 56  WmElbBazynsk      7705 non-null   object
 57  WmGoldUzdrowMOB   8153 non-null   object
 58  WmOlsPuszkin      8758 non-null   object
 59  WpKaliSawick      8548 non-null   object
 60  ZpSzczAndr01      8665 non-null   object
 61  ZpSzczBudzWosMOB  7140 non-null   object
 62  ZpSzczPils02      8735 non-null   object
dtypes: object(63)
memory usage: 4.3+ MB
```

In [ ]:
```
for col in df.columns:
    df[col] = pd.to_numeric(df[col])
```

Obsługa NaNów

In [ ]:
```
df.isna().sum().sort_values()[-10:]  # 8670
```

Out[ ]:
```
WmGoldUzdrowMOB      607
LuZielKrotka         622
PkHorZdrParkMOB      673
LdLodzGdansk         677
SkSkarZielnaMOB      801
KpBydPlPozna         849
WmElbBazynsk        1055
KpToruDziewu        1547
ZpSzczBudzWosMOB    1620
PmGdaLeczk08        4648
dtype: int64
```

In [ ]:
```
# kolumna gdzie mamy 4648 wartości NaN, może zostać usunięta, gdyż jest to ponad
# co więcej z późniejszych analiz wyszło, że ZpSzczBudzWosMOB jest również probl
df = df.drop(columns=["PmGdaLeczk08", "ZpSzczBudzWosMOB"])
```

In [ ]:
```
def fill_missing_values(df):
    window_size = 168
    for column in df.columns:
        for i in range(len(df)):
            if pd.isnull(df.loc[df.index[i], column]):
                start_index = max(0, i - window_size)
                end_index = min(len(df), i + window_size)
                values_in_window = df.loc[df.index[start_index:end_index], colum
                non_null_values = values_in_window.dropna()
                if len(non_null_values) > 0:
                    df.loc[df.index[i], column] = non_null_values.mean()
    return df


df_filled = fill_missing_values(df)
df_filled.head()
```

Out[ ]:

Date	DsDusznikMOB	DsJaworMOB	DsJelGorOgin	DsWrocAlWisn	DsWrocWybCon	KpByd
2019-01-01 01:00:00	33.40530	51.38780	118.7730	102.0900	107.0610	
2019-01-01 02:00:00	13.80280	28.49950	110.0640	63.6111	55.9187	
2019-01-01 03:00:00	9.94056	11.12060	107.9410	48.3540	41.3488	
2019-01-01 04:00:00	6.75889	5.57358	94.5489	34.6621	29.8697	
2019-01-01 05:00:00	7.88722	6.56224	67.8800	14.2870	17.6000	

5 rows × 61 columns

In [ ]:
```
na_cols = []
for col in df_filled.columns:
    if df_filled[col].isna().sum() != 0:
        print(col, df_filled[col].isna().sum())
        na_cols.append(col)

df_filled[na_cols]
```

```
KpMogiNowMOB 60
LbNaleczow 350
```

Out[ ]:

Date	KpMogiNowMOB	LbNaleczow
2019-01-01 01:00:00	NaN	NaN
2019-01-01 02:00:00	NaN	NaN
2019-01-01 03:00:00	NaN	NaN
2019-01-01 04:00:00	NaN	NaN
2019-01-01 05:00:00	NaN	NaN
...	...	...
2019-12-31 20:00:00	12.0	5.42388
2019-12-31 21:00:00	8.9	5.25952
2019-12-31 22:00:00	8.5	5.17734
2019-12-31 23:00:00	8.4	5.42388
2020-01-01 00:00:00	8.5	4.19118

8760 rows × 2 columns

In [ ]:
```
df_filled = df_filled.drop(columns=["KpMogiNowMOB", "LbNaleczow"])
```

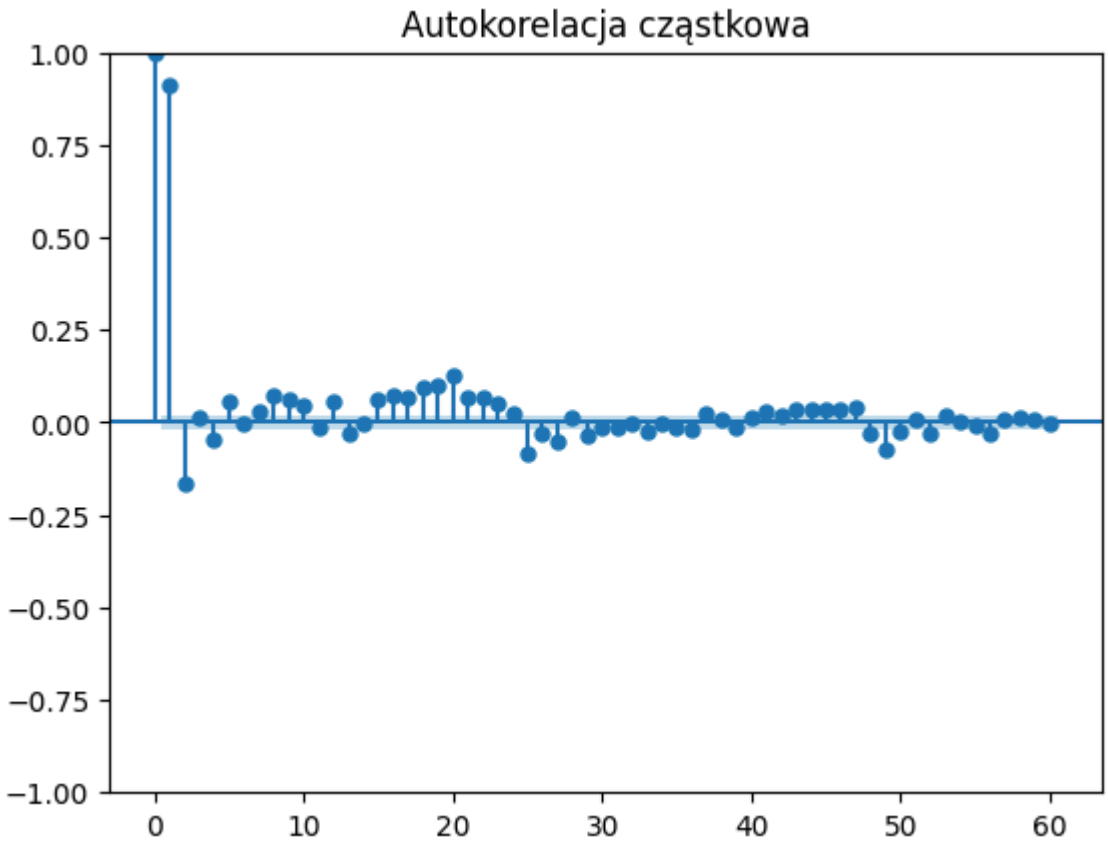
In [ ]:
```
df_filled.isna().sum().sum()  # sprawdzenie
```

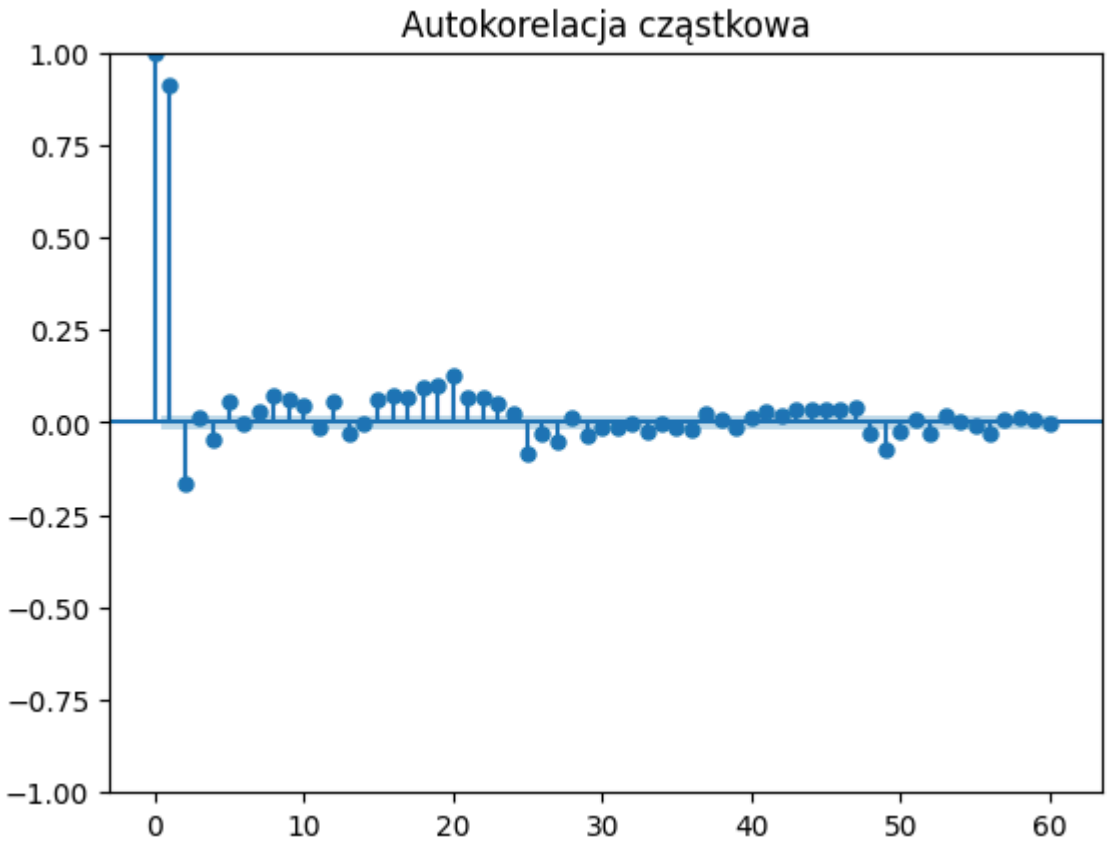
Out[ ]:
```
0
```

Tworzenie cech

In [ ]:
```
plot_pacf(df["DsDusznikMOB"], title="Autokorelacja cząstkowa", lags=60)
```

Out[ ]:





In [ ]:
```python
def create_features(df):
    lagged_features = pd.DataFrame()

    lag_hours = [1, 2, 24, 48]
    for lag_hour in lag_hours:
        shifted_df = df.shift(lag_hour)
        shifted_df.columns = [f"{col}_lag_{lag_hour}h" for col in df.columns]
        lagged_features = pd.concat([lagged_features, shifted_df], axis=1)

    df_with_features = pd.concat([df, lagged_features], axis=1)

    window_sizes = [
        24,
        168,
    ]  # Okna czasowe: 24 godziny (1 dzień) i 168 godzin (tydzień)
    for window_size in window_sizes:
        rolling_features = df.rolling(window=window_size).mean()
        df_with_features = pd.concat(
            [df_with_features, rolling_features.add_suffix(f"_rolling_{window_si
            axis=1,
        )

    df_with_features["hour"] = df_with_features.index.hour
    df_with_features["day_of_week"] = df_with_features.index.dayofweek
    df_with_features["month"] = df_with_features.index.month

    df_with_features["season"] = 0
    for date in df_with_features.index:
        month = date.month
        if 2 < month < 6:
            df_with_features.loc[date, "season"] = 1  # wiosna
        elif 5 < month < 9:
            df_with_features.loc[date, "season"] = 2  # lato
```

```
        elif 8 < month < 12:
            df_with_features.loc[date, "season"] = 3  # jesień
        else:
            df_with_features.loc[date, "season"] = 4  # zima

    return df_with_features


df_with_features = create_features(df_filled)
```

In [ ]:
```
# czynnik przestrzenny - być może się przyda ?


def calculate_distances(df, locs):
    num_stations = len(locs)
    distance_matrix = pd.DataFrame(index=locs.index, columns=locs.index)

    for i in locs.index:
        for j in locs.index:
            if i == j:
                distance_matrix.loc[i, j] = 0.0
            else:
                coord_i = (locs.loc[i, "Y"], locs.loc[i, "X"])
                coord_j = (locs.loc[j, "Y"], locs.loc[j, "X"])
                distance_matrix.loc[i, j] = geodesic(coord_i, coord_j).kilometer

    return distance_matrix


distance_matrix = calculate_distances(df, locs)
print(distance_matrix.head())
```

```
Kod stacji   DsDusznikMOB DsJaworMOB DsJelGorOgin DsWrocAlWisn DsWrocWybCon  \
Kod stacji
DsDusznikMOB          0.0  72.261741    88.383282   100.066391   104.688183
DsJaworMOB      72.261741        0.0    50.463872    89.766786    91.911544
DsJelGorOgin    88.383282  50.463872          0.0   139.234331   141.727927
DsWrocAlWisn   100.066391  89.766786   139.234331          0.0     4.946732
DsWrocWybCon   104.688183  91.911544   141.727927     4.946732          0.0

Kod stacji   KpBydPlPozna KpMogiNowMOB KpToruDziewu KpWloclOkrze LbLubObywate
\
Kod stacji
DsDusznikMOB   338.825453   295.249925   375.509078    379.86708   689.539025
DsJaworMOB      296.14695   258.206307   344.333909   359.362388   705.136736
DsJelGorOgin    340.36181   304.840168   391.975716   408.955894   753.994384
DsWrocAlWisn   241.615496   196.412421   275.459087   281.108604   615.378895
DsWrocWybCon    236.68862   191.548158    270.82111   276.934057   613.429901

Kod stacji    ... SkSkarZielnaMOB SkStaraZlota SlBielPartyz SlKatoKossut  \
Kod stacji    ...
DsDusznikMOB  ...      502.440798   523.701945   300.704055   286.110661
DsJaworMOB    ...      517.884779   540.364963   341.685158   317.970184
DsJelGorOgin  ...      566.579847   588.868714   382.043506   361.826616
DsWrocAlWisn  ...      428.165088   450.700661   263.179327   233.973029
DsWrocWybCon  ...      426.317433   448.928801   264.011807   234.021339

Kod stacji   SlZlotPotLes WmElbBazynsk WmGoldUzdrowMOB WmOlsPuszkin  \
Kod stacji
DsDusznikMOB   340.854025   520.263904      775.932538     577.1761
DsJaworMOB     362.202322   485.175476      760.978039   555.551826
DsJelGorOgin   409.313206   531.086744      810.772114   604.592554
DsWrocAlWisn   273.637284   420.348771       679.86924   478.561483
DsWrocWybCon   272.527455   415.643643      676.020046   474.379215

Kod stacji   WpKaliSawick ZpSzczBudzWosMOB
Kod stacji
DsDusznikMOB   232.487226       387.651152
DsJaworMOB     217.484821       318.497031
DsJelGorOgin   267.887484       305.812896
DsWrocAlWisn   134.510147       375.280353
DsWrocWybCon   130.585708       373.540022

[5 rows x 57 columns]
```

W procesie tworzenia cech dla modelu predykcyjnego wykorzystano różnorodne podejścia, mające na celu uwzględnienie historii pomiarów pyłu PM2.5 oraz informacji związanych z czasem i przestrzenią.

1. **Cechy oparte na opóźnieniach wartości PM2.5:**
   Stworzono cechy opóźnione o różne liczby godzin na podstawie autokorelacji (1h, 2h, 24h, 48h), co pozwala uwzględnić wpływ poprzednich pomiarów na aktualną wartość.

2. **Cechy oparte na oknach czasowych:**
   Utworzono cechy oparte na średnich kroczących w oknach czasowych o długościach 24 i 168 godzin (odpowiednio 1 dzień i 1 tydzień).

3. **Cechy oparte na informacjach z datatime oraz oparte na sezonach:** Utworzono cechy dotyczące informacji czasowych, takie jak godzina dnia, dzieńi tygodni(rok jest wszędzie ten sam)esiąc i rok. Dodanie tych cech pozwala uwzględnić sezonowe oraz cykliczne wzorce w danych, co może mieć istotny wpływ na jakość predykcji. Dodatkowo, stworzono cechę określającą sezon roku (wiosna, lato, jesień, zima) na podstawie miesiąca.

Wszystkie powyższe cechy zostały wybrane z uwzględnieniem potrzeb modelu predykcyjnego oraz dostępnych danych. Wykorzystanie opóźnień, okien czasowych oraz informacji z datatime pozwal na uwzględniienie zarówno krótko- jak i długoterminowych wzorców w danych, co może znacząco poprawić jakość predykcji. Dodatkowo, uwzględnienie sezonów pozwala na adaptację modelu do zmieniających się warunków atmosferycznych w u roku.

Uwzględnienie czynnika przestrzennego

W procesie tworzenia cech nie został bezpośrednio uwzględniony czynnik przestrzenny, czyli geograficzne położenie stacji pomiarowych. Jednakże, możliwe jest uwzględnienie tego czynnika poprzez analizę odległości między stacjami pomiarowymi oraz ich wzajemne oddziaływanie. Funkcja `calculate_distances` oblicza odległość między stacjami pomiarowymi, co może być wykorzystane do stworzenia dodatkowych cech uwzględniających przestrzenne zależności w danych. leżności w danych.

Podział zbioru danych oraz tworzenie wstępnych modeli do predykcji

In [ ]:
```
models = {}  # słownik na modele dla każdej stacji

for station in [col for col in df_with_features.columns if col in locs.index]:
    station_features = [col for col in df_with_features.columns if station in co
    station_features += ["hour", "day_of_week", "month", "season"]

    X_station = df_with_features[station_features]
    y_station = df_with_features[station]

    X_train_station, X_test_station, y_train_station, y_test_station = train_tes
        X_station, y_station, test_size=0.2, random_state=42, shuffle=False
    )  # shuffle False, bo chronologia

    xgb_model_station = XGBRegressor()
    xgb_model_station.fit(X_train_station, y_train_station)

    models[station] = xgb_model_station
```

Decyzja o tworzeniu modeli osobno dla każdej stacji wynika z faktu, że pomiary pyłu PM2.5 mogą być silnie zależne od lokalnych warunków środowiskowych, takich jak lokalizacja geograficzna, warunki atmosferyczne, rodzaj działalności przemysłowej, itp. Każda stacja pomiarowa może reprezentować unikalne warunki, dlatego też osobne modele dla poszczególnych stacji mogą lepiej odzwierciedlać te różnice i dostosować się do lokalnych wzorców zmienności pyłu PM2.5.

Tworzenie modeli indywidualnie dla każdej stacji umożliwia również dostosowanie hiperparametrów modelu do specyficznych cech danych pomiarowych dla danej lokalizacji. Ponadto, pozwala to na bardziej elastyczne zarządzanie danymi, takimi jak usuwanie lub dodawanie cech specyficznych dla danej stacji.

W rezultacie, decyzja o tworzeniu osobnych modeli dla każdej stacji pomiarowej pozwala na lepsze dopasowanie modeli do lokalnych warunków, co może poprawić jakość predykcji wartości pyłu PM2.5 dla poszczególnych stacji.

Część 2 - Tworzenie modelu do predykcji

Plusy i minusy wykorzystania algorytmu XGBoost w kontekście predykcji szeregów czasowych

Plusy	Minusy
Bardzo wysoka wydajność i skuteczność	Wrażliwość na overfitting, szczególnie w przypadku dużych zestawów danych
Możliwość obsługi różnorodnych typów danych	Wymaga dobrego strojenia hiperparametrów
Elastyczność w obsłudze różnych funkcji straty	Wymaga większej ilości danych w stosunku do innych algorytmów
Automatyczna obsługa brakujących wartości	Skomplikowane interpretowanie wyników predykcji
Wydajne przetwarzanie równoległe	Wymaga przetwarzania równoległego na wielu rdzeniach CPU/GPU

Tuning hiperparametrów

In [ ]:
```
param_grid = {
    "n_estimators": [100, 200, 300],
    "max_depth": [3, 4, 5],
    "learning_rate": [0.01, 0.05, 0.1],
    "subsample": [0.8, 0.9, 1.0],
    "colsample_bytree": [0.8, 0.9, 1.0],
}
best_models = {}

for station in [col for col in df_with_features.columns if col in locs.index]:

    station_features = [col for col in df_with_features.columns if station in co
    station_features += ["hour", "day_of_week", "month", "season"]

    X_station = df_with_features[station_features]
    y_station = df_with_features[station]
    X_train_station, X_test_station, y_train_station, y_test_station = train_tes
        X_station, y_station, test_size=0.2, random_state=42, shuffle=False
    )

    xgb_model_station = XGBRegressor()
    random_search_station = RandomizedSearchCV(
        estimator=xgb_model_station,
```

```
        param_distributions=param_grid,
        n_iter=50,
        cv=5,
        verbose=0,
        random_state=42,
        n_jobs=-1,
    )
    random_search_station.fit(X_train_station, y_train_station)

    print(
        "Najlepsze parametry dla stacji",
        station,
        ":",
        random_search_station.best_params_,
    )
    best_model_station = random_search_station.best_estimator_
    best_models[station] = best_model_station
```

```
Najlepsze parametry dla stacji DsDusznikMOB : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji DsJaworMOB : {'subsample': 0.8, 'n_estimators':
300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji DsJelGorOgin : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji DsWrocAlWisn : {'subsample': 1.0, 'n_estimator
s': 300, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 0.9}
Najlepsze parametry dla stacji DsWrocWybCon : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji KpBydPlPozna : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji KpToruDziewu : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji KpWloclOkrze : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LbLubObywate : {'subsample': 0.9, 'n_estimator
s': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LdLodzCzerni : {'subsample': 0.9, 'n_estimator
s': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LdLodzGdansk : {'subsample': 0.9, 'n_estimator
s': 100, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LdZgieMielcz : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LuNowaSolMOB : {'subsample': 0.9, 'n_estimator
s': 100, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LuWsKaziWiel : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LuZarySzyman : {'subsample': 0.9, 'n_estimator
s': 100, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji LuZielKrotka : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 0.9}
Najlepsze parametry dla stacji MpKrakAlKras : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MpKrakBujaka : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MpKrakBulwar : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MpTarRoSitko : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzLegZegrzyn : {'subsample': 0.9, 'n_estimator
s': 100, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzMinMazKaziMOB : {'subsample': 0.8, 'n_estimato
rs': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzOtwoBrzozo : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzPiasPulask : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzPlocMiReja : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzRadTochter : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 0.8}
Najlepsze parametry dla stacji MzSiedKonars : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzWarAlNiepo : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzWarBajkowa : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzWarChrosci : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
```

```
Najlepsze parametry dla stacji MzWarKondrat : {'subsample': 0.9, 'n_estimator
s': 100, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzWarTolstoj : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzWarWokalna : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji MzZyraRoosev : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji OpKKozBSmial : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji OpPrudPodgor : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PdBialWaszyn : {'subsample': 1.0, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PdBorsukowiz : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkDebiGrottg : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkHorZdrParkMOB : {'subsample': 0.8, 'n_estimato
rs': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkJarosPruch : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkMielBierna : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkPrzemGrunw : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkRymZdrPark : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji PkRzeszPilsu : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 0.9}
Najlepsze parametry dla stacji SkKielTargow : {'subsample': 0.8, 'n_estimator
s': 300, 'max_depth': 4, 'learning_rate': 0.1, 'colsample_bytree': 0.9}
Najlepsze parametry dla stacji SkSkarZielnaMOB : {'subsample': 0.9, 'n_estimato
rs': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji SkStaraZlota : {'subsample': 1.0, 'n_estimator
s': 100, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji SlBielPartyz : {'subsample': 0.9, 'n_estimator
s': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji SlKatoKossut : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji SlZlotPotLes : {'subsample': 1.0, 'n_estimator
s': 300, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji WmElbBazynsk : {'subsample': 0.9, 'n_estimator
s': 100, 'max_depth': 3, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji WmGoldUzdrowMOB : {'subsample': 1.0, 'n_estimato
rs': 100, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
Najlepsze parametry dla stacji WmOlsPuszkin : {'subsample': 1.0, 'n_estimator
s': 200, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 0.9}
Najlepsze parametry dla stacji WpKaliSawick : {'subsample': 0.9, 'n_estimator
s': 300, 'max_depth': 3, 'learning_rate': 0.05, 'colsample_bytree': 1.0}
```

Hiperparametry są parametrami, które kontrolują proces uczenia się algorytmu uczenia maszynowego, ale nie są one bezpośrednio uczane z danymi treningowymi. W przeciwieństwie do parametrów modelu, które są uczane podczas procesu uczenia, hiperparametry muszą być ustalone przed rozpoczęciem procesu uczenia i mają wpływ na zachowanie i wydajność modelu.

Przykłady hiperparametrów obejmują:

- Liczbę drzew w modelach opartych na drzewach decyzyjnych (np. w algorytmie XGBoost)
- Maksymalną głębokość drzewa
- Szybkość uczenia (learning rate)
- Liczbę cech branych pod uwagę podczas budowy każdego drzewa
- Liczbę iteracji lub epok w algorytmach uczenia głębokiego

Tuning hiperparametrów jest procesem wybierania najlepszych wartości dla tych parametrów, które optymalizują wydajność i skuteczność modelu na danych treningowych oraz umożliwiają generalizację do danych testowych. Wymaga to eksperymentowania z różnymi zestawami wartości hiperparametrów i oceny wyników za pomocą odpowiednich metryk ewaluacyjnych. Tuning hiperparametrów ma na celu zbalansowanie pomiędzy nadmiernym dopasowaniem (overfitting) a niedopasowaniem (underfitting) modelu do danych treningowych oraz zapewnienie najlepszej możliwej wydajności modelu na nowych, nieznanych danych.

Część 3 - Backtesting

Podział na okna czasowe

In [ ]:
```
tscv = TimeSeriesSplit(n_splits=5)

mse_scores = []

for train_index, test_index in tscv.split(df_with_features):
    X_train = df_with_features.iloc[train_index]
    X_test = df_with_features.iloc[test_index]

    for station, model in models.items():
        station_features = [col for col in X_train.columns if station in col]
        station_features += ["hour", "day_of_week", "month", "season"]

        X_train_station = X_train[station_features]
        X_test_station = X_test[station_features]
        y_train_station = X_train[station]
        y_test_station = X_test[station]

        y_pred_test_station = model.predict(X_test_station)

        mse_test_station = mean_squared_error(y_test_station, y_pred_test_statio
        mse_scores.append(mse_test_station)

mean_mse = np.mean(mse_scores)
print("Średni błąd średniokwadratowy dla wszystkich okien czasowych:", mean_mse)
```

```
Średni błąd średniokwadratowy dla wszystkich okien czasowych: 0.869834857773346
6
```

Roll-Forward Validation

W tej metodologii:

- Tworzymy wiele okien czasowych, gdzie każde kolejne okno jest przesunięciem w przyszłość względem poprzedniego okna.
- Model jest trenowany na danych historycznych do momentu, w którym kończy się dane w danym oknie czasowym.
- Następnie model jest testowany na danych z kolejnego okna czasowego.
- Ten proces jest powtarzany dla każdego okna czasowego, a wyniki są zbierane i oceniane, aby zrozumieć wydajność modelu na przyszłych danych.

Głównym celem tego sposobu walidacji jest uzyskanie rzeczywistej oceny wydajności modelu na przyszłych danych. W przeciwieństwie do tradycyjnego podziału na zbiór treningowy i testowy, gdzie dane testowe są zwykle wyznaczone na końcu zbioru danych, walidacja za pomocą okien czasowych pozwala na ocenę modelu na danych, które są bardziej zbliżone do tych, które zostaną napotkane w przyszłości. Jest to szczególnie istotne w przypadku szeregów czasowych, gdzie zależy nam na przewidywaniu przyszłych wartości na podstawie historii. Dlatego właśnie ten sposób walidacji jest często stosowany w analizie szeregów czasowych.

Ocena wydajności

In [ ]:
```
mse_scores_per_station = {}

for station, model in best_models.items():
    station_features = [col for col in df_with_features.columns if station in co
    station_features += ["hour", "day_of_week", "month", "season"]

    X_station = df_with_features[station_features]
    y_station = df_with_features[station]
    X_train_station, X_test_station, y_train_station, y_test_station = train_tes
        X_station, y_station, test_size=0.2, random_state=42, shuffle=False
    )
    y_pred_test_station = model.predict(X_test_station)
    mse_test_station = mean_squared_error(y_test_station, y_pred_test_station)
    mse_scores_per_station[station] = mse_test_station

for station, mse in mse_scores_per_station.items():
    print("Błąd średniokwadratowy dla stacji", station, ":", mse)
```

```
Błąd średniokwadratowy dla stacji DsDusznikMOB : 1.345556432903394
Błąd średniokwadratowy dla stacji DsJaworMOB : 0.6696523209490601
Błąd średniokwadratowy dla stacji DsJelGorOgin : 0.9608649061639052
Błąd średniokwadratowy dla stacji DsWrocAlWisn : 0.2660882376642321
Błąd średniokwadratowy dla stacji DsWrocWybCon : 0.15753398066628774
Błąd średniokwadratowy dla stacji KpBydPlPozna : 4.574708874529287
Błąd średniokwadratowy dla stacji KpToruDziewu : 0.4468912396635128
Błąd średniokwadratowy dla stacji KpWloclOkrze : 15.914220974175265
Błąd średniokwadratowy dla stacji LbLubObywate : 7.145174240833071
Błąd średniokwadratowy dla stacji LdLodzCzerni : 0.005198768939541813
Błąd średniokwadratowy dla stacji LdLodzGdansk : 15.508351070420604
Błąd średniokwadratowy dla stacji LdZgieMielcz : 8.352345113289637
Błąd średniokwadratowy dla stacji LuNowaSolMOB : 12.663832307555387
Błąd średniokwadratowy dla stacji LuWsKaziWiel : 11.81907820695338
Błąd średniokwadratowy dla stacji LuZarySzyman : 2.909389415462004
Błąd średniokwadratowy dla stacji LuZielKrotka : 0.5021255835608365
Błąd średniokwadratowy dla stacji MpKrakAlKras : 0.9983760628252935
Błąd średniokwadratowy dla stacji MpKrakBujaka : 0.4981495345694086
Błąd średniokwadratowy dla stacji MpKrakBulwar : 0.34089688470481444
Błąd średniokwadratowy dla stacji MpTarRoSitko : 1.2399177082534165
Błąd średniokwadratowy dla stacji MzLegZegrzyn : 25.960476342517897
Błąd średniokwadratowy dla stacji MzMinMazKaziMOB : 1.2299365612304407
Błąd średniokwadratowy dla stacji MzOtwoBrzozo : 3.9396412523527524
Błąd średniokwadratowy dla stacji MzPiasPulask : 0.5579260346451357
Błąd średniokwadratowy dla stacji MzPlocMiReja : 0.13058823445448298
Błąd średniokwadratowy dla stacji MzRadTochter : 3.0218127155880663
Błąd średniokwadratowy dla stacji MzSiedKonars : 5.1008180329957
Błąd średniokwadratowy dla stacji MzWarAlNiepo : 0.027646886444682425
Błąd średniokwadratowy dla stacji MzWarBajkowa : 2.352495178130097
Błąd średniokwadratowy dla stacji MzWarChrosci : 0.0717506056626298
Błąd średniokwadratowy dla stacji MzWarKondrat : 0.14395308978968022
Błąd średniokwadratowy dla stacji MzWarTolstoj : 0.04425163344287872
Błąd średniokwadratowy dla stacji MzWarWokalna : 0.034153054756790056
Błąd średniokwadratowy dla stacji MzZyraRoosev : 4.964416364594697
Błąd średniokwadratowy dla stacji OpKKozBSmial : 0.08526862365664101
Błąd średniokwadratowy dla stacji OpPrudPodgor : 5.584097880823327
Błąd średniokwadratowy dla stacji PdBialWaszyn : 0.059243622563333556
Błąd średniokwadratowy dla stacji PdBorsukowiz : 0.032629111911529625
Błąd średniokwadratowy dla stacji PkDebiGrottg : 0.6539806769946986
Błąd średniokwadratowy dla stacji PkHorZdrParkMOB : 1.0346394774849688
Błąd średniokwadratowy dla stacji PkJarosPruch : 3.5682369328871624
Błąd średniokwadratowy dla stacji PkMielBierna : 2.4697578883129645
Błąd średniokwadratowy dla stacji PkPrzemGrunw : 0.13537959359638535
Błąd średniokwadratowy dla stacji PkRymZdrPark : 0.5419022424769233
Błąd średniokwadratowy dla stacji PkRzeszPilsu : 0.6615384083851352
Błąd średniokwadratowy dla stacji SkKielTargow : 2.265991515354574
Błąd średniokwadratowy dla stacji SkSkarZielnaMOB : 2.094345034722579
Błąd średniokwadratowy dla stacji SkStaraZlota : 15.236584282125161
Błąd średniokwadratowy dla stacji SlBielPartyz : 5.373464539437636
Błąd średniokwadratowy dla stacji SlKatoKossut : 1.1560471345606045
Błąd średniokwadratowy dla stacji SlZlotPotLes : 0.026646200692995413
Błąd średniokwadratowy dla stacji WmElbBazynsk : 0.013105251276995378
Błąd średniokwadratowy dla stacji WmGoldUzdrowMOB : 0.3351088207631546
Błąd średniokwadratowy dla stacji WmOlsPuszkin : 0.13602264288770233
Błąd średniokwadratowy dla stacji WpKaliSawick : 0.7579302496630648
```

1. **Wariancja błędów w pomiarach**: Błędy średniokwadratowe (MSE) różnią się znacząco między różnymi stacjami pomiarowymi. Na przykład, stacja DsWrocAlWisn ma stosunkowo niski błąd MSE wynoszący 0.27, podczas gdy stacja MzLegZegrzyn

ma wysoki błąd MSE na poziomie 25.96. To sugeruje, że niektóre stacje mają lepszą dokładność w pomiarach niż inne.

2. **Zróżnicowanie jakości danych**: Błędy MSE mogą odzwierciedlać różnice w jakości danych między różnymi stacjami. Stacje o niższych błędach MSE mogą być lepiej skonfigurowane, lepiej utrzymane lub znajdować się w miejscach, gdzie pomiary są łatwiejsze do przeprowadzenia. Stacje o wyższych błędach MSE mogą mieć problemy techniczne, być niewłaściwie kalibrowane lub znajdować się w miejscach, gdzie warunki pomiarowe są trudniejsze.

3. **Wartość odniesienia**: Stacje z bardzo niskimi błędami MSE, takie jak LdLodzCzerni (0.005) i SlZlotPotLes (0.027), sugerują, że pomiary na tych stacjach są bardzo dokładne i mogą być używane jako punkty odniesienia do porównywania dokładności innych pomiarów.

Backtesting modelu:

Backtesting to metoda oceny wydajności modelu na danych historycznych, które nie zostały użyte do trenowania. Jest to często stosowana praktyka w finansach i inwestycjach, gdzie model jest testowany na danych historycznych, aby sprawdzić, jak dobrze działa na przeszłych danych. W przypadku analizy szeregów czasowych backtesting pozwala na ocenę, jak dobrze model radzi sobie z przewidywaniem przyszłych wartości na podstawie danych historycznyc h.

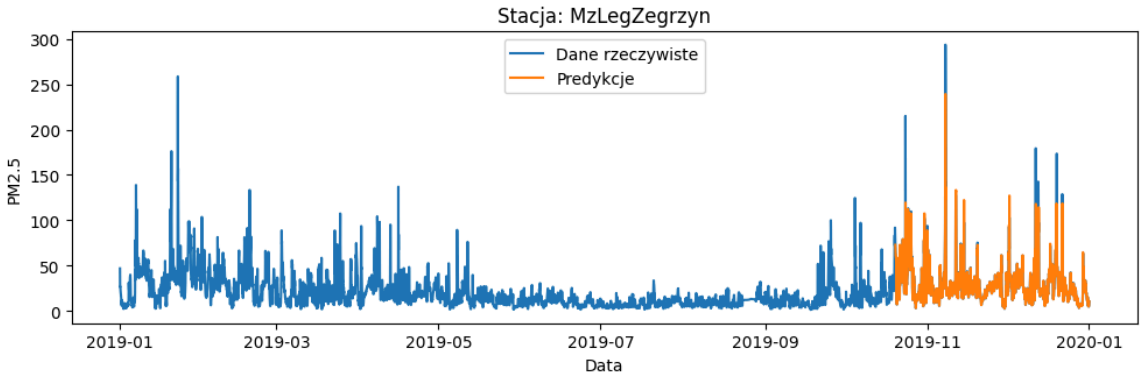
Różnica między backtestingiem a testowaniem na danych niebędących szeregami czasowymi:

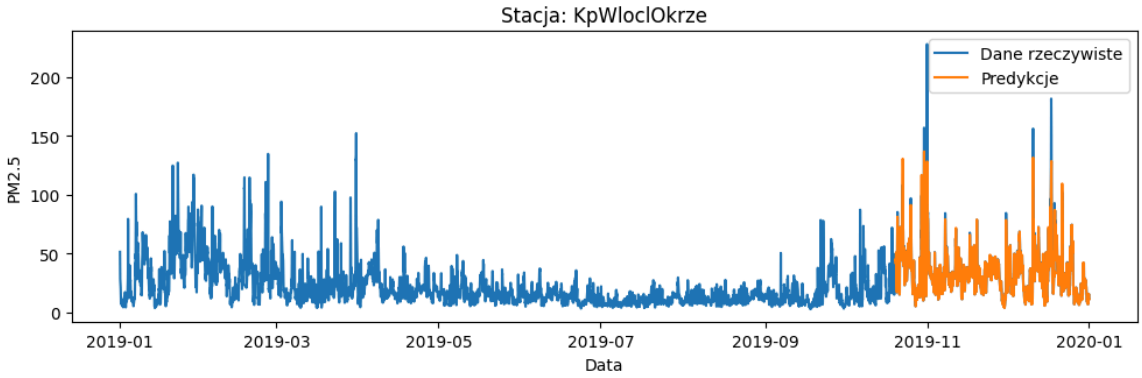
Backtesting ma na celu ocenę wydajności modelu na danych historycznych, które są zorganizowane w szereg czasowy, aby sprawdzić jego zdolność do przewidywania przyszłych wartości na podstawie przeszłych danych.

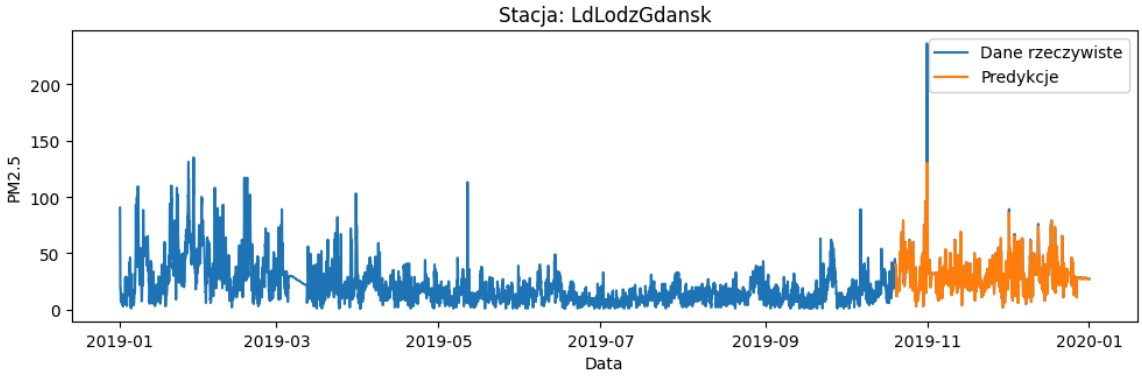
Testowanie wydajności modelu dla danych niebędących szeregami czasowymi może obejmować różne typy danych, ale zazwyczaj nie są one zorganizowane w szereg czasowy. Celem jest zwykle sprawdzenie ogólnej zdolności modelu do generalizacji na nowe, nieznane dane, bez uwzględniania ich sekwencyjnej struktury czaasowej.

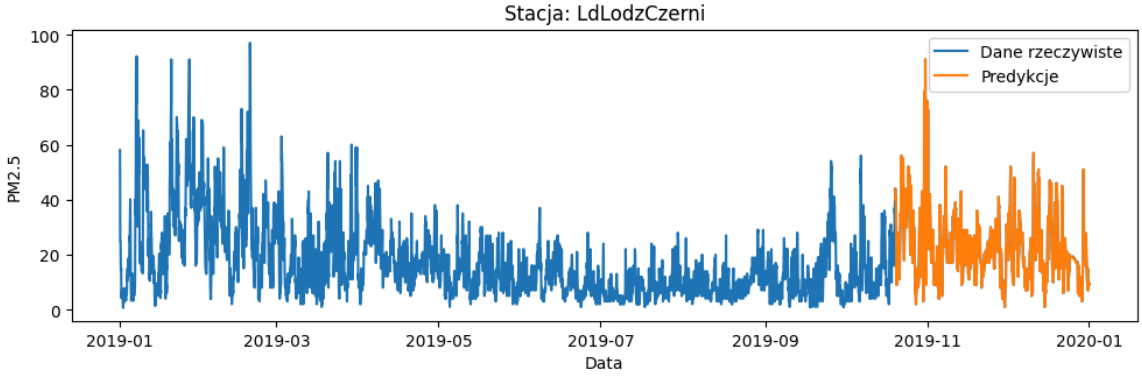
In [ ]:
```
highest_mse_stations = sorted(
    mse_scores_per_station, key=mse_scores_per_station.get, reverse=True
)[
    :3
]  # trzy stacje z najwyższym MSE
lowest_mse_stations = sorted(mse_scores_per_station, key=mse_scores_per_station.
    :3
]  # trzy stacje z najniższym MSE
sorted_stations = (
    highest_mse_stations + lowest_mse_stations
)  # sorted(mse_scores_per_station, key=mse_scores_per_station.get)[:6]

fig, axs = plt.subplots(6, 1, figsize=(10, 20))
axs = axs.ravel()
```

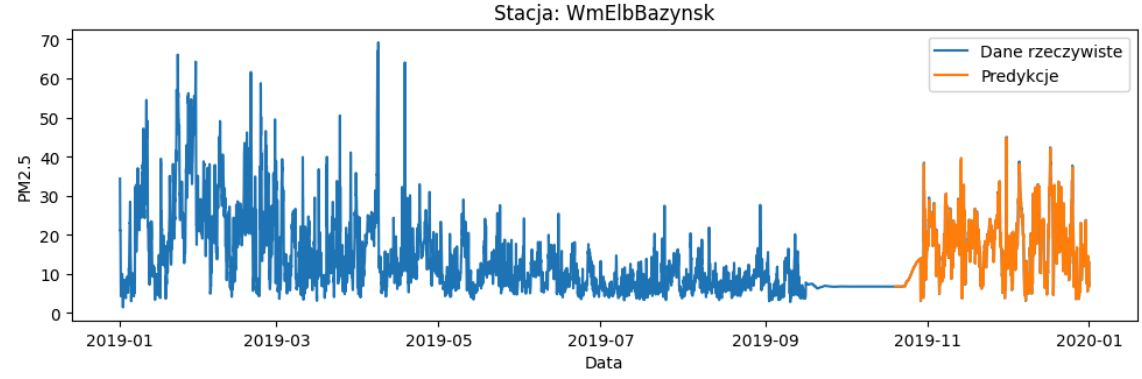
```
for i, station in enumerate(sorted_stations):
    station_features = [col for col in df_with_features.columns if station in co
    station_features += ["hour", "day_of_week", "month", "season"]

    X_station = df_with_features[station_features]
    y_station = df_with_features[station]

    X_train_station, X_test_station, y_train_station, y_test_station = train_tes
        X_station, y_station, test_size=0.2, random_state=42, shuffle=False
    )

    y_pred_test_station = best_models[station].predict(X_test_station)

    axs[i].plot(y_station.index, y_station, label="Dane rzeczywiste")
    axs[i].plot(y_test_station.index, y_pred_test_station, label="Predykcje")
    axs[i].set_title(f"Stacja: {station}")
    axs[i].set_xlabel("Data")
    axs[i].set_ylabel("PM2.5")
    axs[i].legend()

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

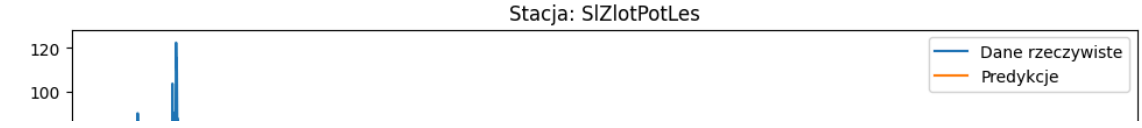

Stacja: MzLegZegrzyn
Dane rzeczywiste
Predykcje
PM2.5
Data
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01

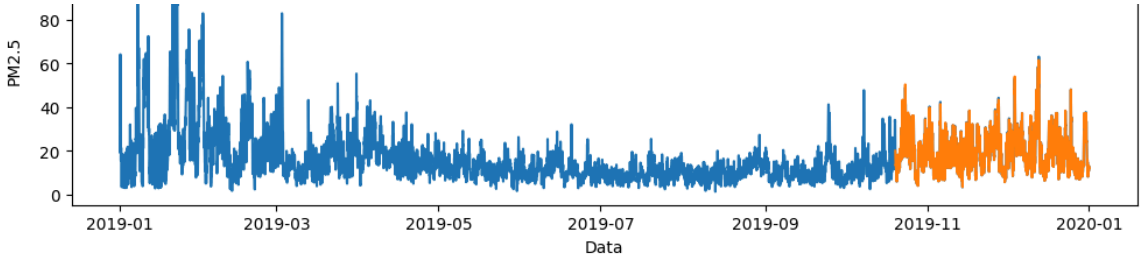

Stacja: KpWloclOkrze
Dane rzeczywiste
Predykcje
PM2.5
Data
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01


Stacja: LdLodzGdansk
Dane rzeczywiste
Predykcje
PM2.5
Data
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01


Stacja: LdLodzCzerni
Dane rzeczywiste
Predykcje
PM2.5
Data
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01


Stacja: WmElbBazynsk
Dane rzeczywiste
Predykcje
PM2.5
Data
2019-01
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01


Stacja: SlZlotPotLes
Dane rzeczywiste
Predykcje



Powyższy kod generuje wykresy, na których porównywane są rzeczywiste i przewidywane wartości PM2.5 dla trzech stacji z najwyższym oraz trzech stacji z najniższym błędem średniokwadratowym (MSE). Stacje zostały wybrane na podstawie wyników MSE dla każdej stacji. Widać, że dla najgorszych wyników predykcje nie pokrywają się z prawdziwymi wartościami, zwłaszcza na pikach, a dla top 3 predykcje wyglądają praktycznie jak wartości rzeczywiste. Mimo wszystko nawet najgorsze 3 modele pod względem MSE nie wyglądają źle - można predykować PM2.5 przy użyciu stworzonych modeli dla każdej ze stacji.

Część 4 - Analiza istotności cech

In [ ]:
```
importances = {}

for station, model in best_models.items():
    station_features = [col for col in df_with_features.columns if station in co
    station_features += ["hour", "day_of_week", "month", "season"]
    importances[station] = model.feature_importances_
```

In [ ]:
```
feature_importances_with_names = {}

for station, importances_array in importances.items():
    station_features = [col for col in df_with_features.columns if station in co
    station_features += ["hour", "day_of_week", "month", "season"]
    features_with_importance = [
        (feature, importance)
        for feature, importance in zip(station_features, importances_array)
    ]
    feature_importances_with_names[station] = features_with_importance

print(feature_importances_with_names)
```

{'DsDusznikMOB': [('DsDusznikMOB', 0.94675136), ('DsDusznikMOB_lag_1h', 0.00614
3732), ('DsDusznikMOB_lag_2h', 0.003886389), ('DsDusznikMOB_lag_24h', 0.0085302
99), ('DsDusznikMOB_lag_48h', 0.004096906), ('DsDusznikMOB_rolling_24h', 0.0071
76677), ('DsDusznikMOB_rolling_168h', 0.0049393685), ('hour', 0.00808442), ('da
y_of_week', 0.010390798), ('month', 0.0), ('season', 0.0)], 'DsJaworMOB': [('Ds
JaworMOB', 0.79871464), ('DsJaworMOB_lag_1h', 0.018121867), ('DsJaworMOB_lag_2
h', 0.008997144), ('DsJaworMOB_lag_24h', 0.010596619), ('DsJaworMOB_lag_48h',
0.011284), ('DsJaworMOB_rolling_24h', 0.012860041), ('DsJaworMOB_rolling_168h',
0.009522753), ('hour', 0.021761505), ('day_of_week', 0.07960955), ('month', 0.0
076776496), ('season', 0.020854177)], 'DsJelGorOgin': [('DsJelGorOgin', 0.97926
49), ('DsJelGorOgin_lag_1h', 0.0026700606), ('DsJelGorOgin_lag_2h', 0.000837077
8), ('DsJelGorOgin_lag_24h', 0.0017308359), ('DsJelGorOgin_lag_48h', 0.00325703
76), ('DsJelGorOgin_rolling_24h', 0.006459928), ('DsJelGorOgin_rolling_168h',
0.0012897578), ('hour', 0.00079656666), ('day_of_week', 0.0027260254), ('mont
h', 0.0003550598), ('season', 0.0006127189)], 'DsWrocAlWisn': [('DsWrocAlWisn',
0.73401916), ('DsWrocAlWisn_lag_1h', 0.24356961), ('DsWrocAlWisn_lag_2h', 0.002
902099), ('DsWrocAlWisn_lag_24h', 0.0028460468), ('DsWrocAlWisn_lag_48h', 0.003
700395), ('DsWrocAlWisn_rolling_24h', 0.0034788602), ('DsWrocAlWisn_rolling_168
h', 0.0047439435), ('hour', 0.0037457831), ('day_of_week', 0.0003761734), ('mon
th', 0.00050684524), ('season', 0.000111042)], 'DsWrocWybCon': [('DsWrocWybCo
n', 0.9399087), ('DsWrocWybCon_lag_1h', 0.01841489), ('DsWrocWybCon_lag_2h', 0.
0017383924), ('DsWrocWybCon_lag_24h', 0.0013354042), ('DsWrocWybCon_lag_48h',
0.00459521), ('DsWrocWybCon_rolling_24h', 0.007737776), ('DsWrocWybCon_rolling_
168h', 0.011290406), ('hour', 0.0058035324), ('day_of_week', 0.0022314952), ('m
onth', 0.006944343), ('season', 0.0)], 'KpBydPlPozna': [('KpBydPlPozna', 0.9258
0205), ('KpBydPlPozna_lag_1h', 0.0073247077), ('KpBydPlPozna_lag_2h', 0.0014301
788), ('KpBydPlPozna_lag_24h', 0.0043589943), ('KpBydPlPozna_lag_48h', 0.003433
4094), ('KpBydPlPozna_rolling_24h', 0.009898998), ('KpBydPlPozna_rolling_168h',
0.010754986), ('hour', 0.001980375), ('day_of_week', 0.033677097), ('month', 0.
0013391612), ('season', 0.0)], 'KpToruDziewu': [('KpToruDziewu', 0.9806877),
('KpToruDziewu_lag_1h', 0.003907532), ('KpToruDziewu_lag_2h', 0.0008467992),
('KpToruDziewu_lag_24h', 0.0008783674), ('KpToruDziewu_lag_48h', 0.004396358),
('KpToruDziewu_rolling_24h', 0.0014243703), ('KpToruDziewu_rolling_168h', 0.000
5715308), ('hour', 0.0059808446), ('day_of_week', 0.000765757), ('month', 0.000
54074475), ('season', 0.0)], 'KpWloclOkrze': [('KpWloclOkrze', 0.9644188), ('Kp
WloclOkrze_lag_1h', 0.001614822), ('KpWloclOkrze_lag_2h', 0.00072387693), ('KpW
loclOkrze_lag_24h', 0.0013973562), ('KpWloclOkrze_lag_48h', 0.0014938707), ('Kp
WloclOkrze_rolling_24h', 0.00053784385), ('KpWloclOkrze_rolling_168h', 0.001626
6038), ('hour', 0.0008881877), ('day_of_week', 0.0011215283), ('month', 0.00417
11195), ('season', 0.022006072)], 'LbLubObywate': [('LbLubObywate', 0.9584723),
('LbLubObywate_lag_1h', 0.004790927), ('LbLubObywate_lag_2h', 0.0023603684),
('LbLubObywate_lag_24h', 0.0044788755), ('LbLubObywate_lag_48h', 0.0024232338),
('LbLubObywate_rolling_24h', 0.0046135415), ('LbLubObywate_rolling_168h', 0.011
486182), ('hour', 0.010525437), ('day_of_week', 0.00031221955), ('month', 0.000
36529268), ('season', 0.00017159678)], 'LdLodzCzerni': [('LdLodzCzerni', 0.9999
803), ('LdLodzCzerni_lag_1h', 1.5134194e-06), ('LdLodzCzerni_lag_2h', 3.1316763
e-06), ('LdLodzCzerni_lag_24h', 3.7162101e-06), ('LdLodzCzerni_lag_48h', 2.9312
7e-06), ('LdLodzCzerni_rolling_24h', 2.378237e-06), ('LdLodzCzerni_rolling_168
h', 1.8860208e-06), ('hour', 2.6660498e-06), ('day_of_week', 9.3436626e-07),
('month', 3.844655e-07), ('season', 1.3014846e-07)], 'LdLodzGdansk': [('LdLodzG
dansk', 0.97601396), ('LdLodzGdansk_lag_1h', 0.002371138), ('LdLodzGdansk_lag_2
h', 0.0009677587), ('LdLodzGdansk_lag_24h', 0.0025269536), ('LdLodzGdansk_lag_4
8h', 0.0024966148), ('LdLodzGdansk_rolling_24h', 0.0013602253), ('LdLodzGdansk_
rolling_168h', 0.0027860366), ('hour', 0.0071181343), ('day_of_week', 0.0037810
653), ('month', 0.000578225), ('season', 0.0)], 'LdZgieMielcz': [('LdZgieMielc
z', 0.9150083), ('LdZgieMielcz_lag_1h', 0.007814613), ('LdZgieMielcz_lag_2h',
0.0018782455), ('LdZgieMielcz_lag_24h', 0.012580559), ('LdZgieMielcz_lag_48h',
0.0072656274), ('LdZgieMielcz_rolling_24h', 0.008750638), ('LdZgieMielcz_rollin
g_168h', 0.0039668847), ('hour', 0.004431846), ('day_of_week', 0.008294881),
('month', 0.0051133092), ('season', 0.02489513)], 'LuNowaSolMOB': [('LuNowaSolM

OB', 0.8190225), ('LuNowaSolMOB_lag_1h', 0.010856209), ('LuNowaSolMOB_lag_2h', 0.0043292763), ('LuNowaSolMOB_lag_24h', 0.012703693), ('LuNowaSolMOB_lag_48h', 0.01731155), ('LuNowaSolMOB_rolling_24h', 0.013277695), ('LuNowaSolMOB_rolling_168h', 0.006780615), ('hour', 0.032467633), ('day_of_week', 0.07139958), ('month', 0.011851218), ('season', 0.0)], 'LuWsKaziWiel': [('LuWsKaziWiel', 0.9324442), ('LuWsKaziWiel_lag_1h', 0.0069482694), ('LuWsKaziWiel_lag_2h', 0.009939348), ('LuWsKaziWiel_lag_24h', 0.0058434075), ('LuWsKaziWiel_lag_48h', 0.005702173), ('LuWsKaziWiel_rolling_24h', 0.0026488213), ('LuWsKaziWiel_rolling_168h', 0.008669759), ('hour', 0.007791686), ('day_of_week', 0.012393746), ('month', 0.007618562), ('season', 0.0)], 'LuZarySzyman': [('LuZarySzyman', 0.87216383), ('LuZarySzyman_lag_1h', 0.022428097), ('LuZarySzyman_lag_2h', 0.007035869), ('LuZarySzyman_lag_24h', 0.009543649), ('LuZarySzyman_lag_48h', 0.016905382), ('LuZarySzyman_rolling_24h', 0.0075500864), ('LuZarySzyman_rolling_168h', 0.012188609), ('hour', 0.008870358), ('day_of_week', 0.02811875), ('month', 0.012534201), ('season', 0.0026611036)], 'LuZielKrotka': [('LuZielKrotka', 0.6153453), ('LuZielKrotka_lag_1h', 0.22787839), ('LuZielKrotka_lag_2h', 0.016075442), ('LuZielKrotka_lag_24h', 0.011455066), ('LuZielKrotka_lag_48h', 0.026066048), ('LuZielKrotka_rolling_24h', 0.06860388), ('LuZielKrotka_rolling_168h', 0.01984083), ('hour', 0.010489283), ('day_of_week', 0.004245782), ('month', 0.0), ('season', 0.0)], 'MpKrakAlKras': [('MpKrakAlKras', 0.98674875), ('MpKrakAlKras_lag_1h', 0.0026853366), ('MpKrakAlKras_lag_2h', 0.00062535406), ('MpKrakAlKras_lag_24h', 0.0011791224), ('MpKrakAlKras_lag_48h', 0.0032629117), ('MpKrakAlKras_rolling_24h', 0.00068129756), ('MpKrakAlKras_rolling_168h', 0.0006627303), ('hour', 0.000815177 77), ('day_of_week', 0.0028638788), ('month', 0.00042345645), ('season', 5.1935687e-05)], 'MpKrakBujaka': [('MpKrakBujaka', 0.978321), ('MpKrakBujaka_lag_1h', 0.004205574), ('MpKrakBujaka_lag_2h', 0.0017638265), ('MpKrakBujaka_lag_24h', 0.0012568836), ('MpKrakBujaka_lag_48h', 0.0012153791), ('MpKrakBujaka_rolling_24h', 0.00089790754), ('MpKrakBujaka_rolling_168h', 0.005783477), ('hour', 0.000841004), ('day_of_week', 0.0052797073), ('month', 0.0004353061), ('season', 0.0)], 'MpKrakBulwar': [('MpKrakBulwar', 0.99275), ('MpKrakBulwar_lag_1h', 0.0014579258), ('MpKrakBulwar_lag_2h', 0.00023643862), ('MpKrakBulwar_lag_24h', 0.00026291586), ('MpKrakBulwar_lag_48h', 0.00084540545), ('MpKrakBulwar_rolling_24h', 0.0003236071), ('MpKrakBulwar_rolling_168h', 0.000751205), ('hour', 0.001014738 9), ('day_of_week', 0.0009281044), ('month', 0.0014296656), ('season', 0.0)], 'MpTarRoSitko': [('MpTarRoSitko', 0.9686086), ('MpTarRoSitko_lag_1h', 0.0026367484), ('MpTarRoSitko_lag_2h', 0.002473947), ('MpTarRoSitko_lag_24h', 0.0024352742), ('MpTarRoSitko_lag_48h', 0.0045441473), ('MpTarRoSitko_rolling_24h', 0.003950301), ('MpTarRoSitko_rolling_168h', 0.0028624132), ('hour', 0.0041622887), ('day_of_week', 0.005198397), ('month', 0.0031278548), ('season', 0.0)], 'MzLegZegrzyn': [('MzLegZegrzyn', 0.8549873), ('MzLegZegrzyn_lag_1h', 0.004432824), ('MzLegZegrzyn_lag_2h', 0.0087712845), ('MzLegZegrzyn_lag_24h', 0.026463442), ('MzLegZegrzyn_lag_48h', 0.018861363), ('MzLegZegrzyn_rolling_24h', 0.010764113), ('MzLegZegrzyn_rolling_168h', 0.023456885), ('hour', 0.0011489625), ('day_of_week', 0.038305957), ('month', 0.008591806), ('season', 0.004216073)], 'MzMinMazKaziMOB': [('MzMinMazKaziMOB', 0.8667383), ('MzMinMazKaziMOB_lag_1h', 0.013693207), ('MzMinMazKaziMOB_lag_2h', 0.0056505767), ('MzMinMazKaziMOB_lag_24h', 0.010504707), ('MzMinMazKaziMOB_lag_48h', 0.01779407), ('MzMinMazKaziMOB_rolling_24h', 0.0070219357), ('MzMinMazKaziMOB_rolling_168h', 0.0149019975), ('hour', 0.017243864), ('day_of_week', 0.027746346), ('month', 0.018704966), ('season', 0.0)], 'MzOtwoBrzozo': [('MzOtwoBrzozo', 0.97762656), ('MzOtwoBrzozo_lag_1h', 0.0019678357), ('MzOtwoBrzozo_lag_2h', 0.0012336257), ('MzOtwoBrzozo_lag_24h', 0.0019158192), ('MzOtwoBrzozo_lag_48h', 0.0020859353), ('MzOtwoBrzozo_rolling_24h', 0.0018140597), ('MzOtwoBrzozo_rolling_168h', 0.0014734478), ('hour', 0.0018002279), ('day_of_week', 0.002014186), ('month', 0.0077824593), ('season', 0.00028586134)], 'MzPiasPulask': [('MzPiasPulask', 0.97584397), ('MzPiasPulask_lag_1h', 0.00078313734), ('MzPiasPulask_lag_2h', 0.0012607226), ('MzPiasPulask_lag_24h', 0.0041138614), ('MzPiasPulask_lag_48h', 0.00048848696), ('MzPiasPulask_rolling_24h', 0.0005984949), ('MzPiasPulask_rolling_168h', 0.015930133), ('hour', 0.0003212932), ('day_of_week', 0.00026431255), ('month', 0.00032124517), ('season', 7.438061e-05)], 'MzPlocMiReja': [('MzPlocMiReja', 0.9909829),

('MzPlocMiReja_lag_1h', 0.001672735), ('MzPlocMiReja_lag_2h', 0.00040714451), ('MzPlocMiReja_lag_24h', 0.0010232226), ('MzPlocMiReja_lag_48h', 0.0008701078 5), ('MzPlocMiReja_rolling_24h', 0.0006203599), ('MzPlocMiReja_rolling_168h', 0.00079914415), ('hour', 0.0010396397), ('day_of_week', 0.0012346249), ('mont h', 0.0013501841), ('season', 0.0)], 'MzRadTochter': [('MzRadTochter', 0.536168 7), ('MzRadTochter_lag_1h', 0.26316074), ('MzRadTochter_lag_2h', 0.010404855), ('MzRadTochter_lag_24h', 0.012139633), ('MzRadTochter_lag_48h', 0.0073520364), ('MzRadTochter_rolling_24h', 0.043539472), ('MzRadTochter_rolling_168h', 0.0148 38301), ('hour', 0.007590716), ('day_of_week', 0.0011199309), ('month', 0.10314 3886), ('season', 0.0005417218)], 'MzSiedKonars': [('MzSiedKonars', 0.9658466 6), ('MzSiedKonars_lag_1h', 0.0009192302), ('MzSiedKonars_lag_2h', 0.001074236 9), ('MzSiedKonars_lag_24h', 0.0048760897), ('MzSiedKonars_lag_48h', 0.00289896 04), ('MzSiedKonars_rolling_24h', 0.0032266953), ('MzSiedKonars_rolling_168h', 0.0034668348), ('hour', 0.0013541873), ('day_of_week', 0.006806806), ('month', 0.002370575), ('season', 0.0071596857)], 'MzWarAlNiepo': [('MzWarAlNiepo', 0.97 489095), ('MzWarAlNiepo_lag_1h', 0.0026223233), ('MzWarAlNiepo_lag_2h', 0.00038 717364), ('MzWarAlNiepo_lag_24h', 0.0007685105), ('MzWarAlNiepo_lag_48h', 0.004 239032), ('MzWarAlNiepo_rolling_24h', 0.010954535), ('MzWarAlNiepo_rolling_168 h', 0.0012936984), ('hour', 0.0027947389), ('day_of_week', 0.00039694557), ('mo nth', 0.0005967279), ('season', 0.0010552853)], 'MzWarBajkowa': [('MzWarBajkow a', 0.9798379), ('MzWarBajkowa_lag_1h', 0.0023510316), ('MzWarBajkowa_lag_2h', 0.0010615644), ('MzWarBajkowa_lag_24h', 0.0009044719), ('MzWarBajkowa_lag_48h', 0.0015354028), ('MzWarBajkowa_rolling_24h', 0.0008490613), ('MzWarBajkowa_rolli ng_168h', 0.0038006795), ('hour', 0.0004317272), ('day_of_week', 0.002403472), ('month', 0.006725463), ('season', 9.9285426e-05)], 'MzWarChrosci': [('MzWarChr osci', 0.97847915), ('MzWarChrosci_lag_1h', 0.001168039), ('MzWarChrosci_lag_2 h', 0.0010323353), ('MzWarChrosci_lag_24h', 0.001486481), ('MzWarChrosci_lag_48 h', 0.002449001), ('MzWarChrosci_rolling_24h', 0.0011294122), ('MzWarChrosci_ro lling_168h', 0.003493828), ('hour', 0.0013253607), ('day_of_week', 0.009180482 5), ('month', 0.00025588), ('season', 0.0)], 'MzWarKondrat': [('MzWarKondrat', 0.9594122), ('MzWarKondrat_lag_1h', 0.0073395395), ('MzWarKondrat_lag_2h', 0.00 22920868), ('MzWarKondrat_lag_24h', 0.0050329007), ('MzWarKondrat_lag_48h', 0.0 046062605), ('MzWarKondrat_rolling_24h', 0.0010448091), ('MzWarKondrat_rolling_ 168h', 0.011882601), ('hour', 0.0022766714), ('day_of_week', 0.0012993304), ('m onth', 0.004813533), ('season', 0.0)], 'MzWarTolstoj': [('MzWarTolstoj', 0.9931 3384), ('MzWarTolstoj_lag_1h', 0.0031118514), ('MzWarTolstoj_lag_2h', 0.0004564 051), ('MzWarTolstoj_lag_24h', 0.00036598952), ('MzWarTolstoj_lag_48h', 0.00085 64396), ('MzWarTolstoj_rolling_24h', 0.0003703869), ('MzWarTolstoj_rolling_168 h', 0.00041938538), ('hour', 0.00066110917), ('day_of_week', 0.00047581788), ('month', 9.042064e-05), ('season', 5.835301e-05)], 'MzWarWokalna': [('MzWarWok alna', 0.98117334), ('MzWarWokalna_lag_1h', 0.004379098), ('MzWarWokalna_lag_2 h', 0.00031156954), ('MzWarWokalna_lag_24h', 0.0018412491), ('MzWarWokalna_lag_ 48h', 0.0034025577), ('MzWarWokalna_rolling_24h', 0.0055283424), ('MzWarWokalna _rolling_168h', 0.0011997191), ('hour', 0.0017535872), ('day_of_week', 0.000208 98948), ('month', 0.00020158493), ('season', 0.0)], 'MzZyraRoosev': [('MzZyraRo osev', 0.9577027), ('MzZyraRoosev_lag_1h', 0.011624215), ('MzZyraRoosev_lag_2 h', 0.00076230697), ('MzZyraRoosev_lag_24h', 0.010048075), ('MzZyraRoosev_lag_4 8h', 0.0040296046), ('MzZyraRoosev_rolling_24h', 0.004419593), ('MzZyraRoosev_r olling_168h', 0.005041184), ('hour', 0.004131911), ('day_of_week', 0.001925590 3), ('month', 0.00021063328), ('season', 0.000104078295)], 'OpKKozBSmial': [('O pKKozBSmial', 0.96941334), ('OpKKozBSmial_lag_1h', 0.01636084), ('OpKKozBSmial_ lag_2h', 0.00095872564), ('OpKKozBSmial_lag_24h', 0.001570373), ('OpKKozBSmial_ lag_48h', 0.0017096838), ('OpKKozBSmial_rolling_24h', 0.0022907236), ('OpKKozBS mial_rolling_168h', 0.0019767787), ('hour', 0.0027405971), ('day_of_week', 0.00 17179681), ('month', 0.00079631095), ('season', 0.00046466928)], 'OpPrudPodgo r': [('OpPrudPodgor', 0.92253315), ('OpPrudPodgor_lag_1h', 0.014976072), ('OpPr udPodgor_lag_2h', 0.008940822), ('OpPrudPodgor_lag_24h', 0.008590763), ('OpPrud Podgor_lag_48h', 0.0064454637), ('OpPrudPodgor_rolling_24h', 0.006447099), ('Op PrudPodgor_rolling_168h', 0.006833314), ('hour', 0.01423416), ('day_of_week', 0.003983433), ('month', 0.0068435436), ('season', 0.00017222586)], 'PdBialWaszy

n': [('PdBialWaszyn', 0.7922845), ('PdBialWaszyn_lag_1h', 0.032335263), ('PdBialWaszyn_lag_2h', 0.01675561), ('PdBialWaszyn_lag_24h', 0.04442643), ('PdBialWaszyn_lag_48h', 0.011404095), ('PdBialWaszyn_rolling_24h', 0.0076105827), ('PdBialWaszyn_rolling_168h', 0.00096650707), ('hour', 0.09080185), ('day_of_week', 0.0019160447), ('month', 0.00035102697), ('season', 0.0011480395)], 'PdBorsukowiz': [('PdBorsukowiz', 0.98017675), ('PdBorsukowiz_lag_1h', 0.004493791), ('PdBorsukowiz_lag_2h', 0.00058871944), ('PdBorsukowiz_lag_24h', 0.0016117584), ('PdBorsukowiz_lag_48h', 0.0026228158), ('PdBorsukowiz_rolling_24h', 0.0019064205), ('PdBorsukowiz_rolling_168h', 0.003149285), ('hour', 0.0024515956), ('day_of_week', 0.0025935608), ('month', 0.00040521403), ('season', 0.0)], 'PkDebiGrottg': [('PkDebiGrottg', 0.9235598), ('PkDebiGrottg_lag_1h', 0.019131437), ('PkDebiGrottg_lag_2h', 0.0012710884), ('PkDebiGrottg_lag_24h', 0.0039353822), ('PkDebiGrottg_lag_48h', 0.002345614), ('PkDebiGrottg_rolling_24h', 0.010520426), ('PkDebiGrottg_rolling_168h', 0.010624576), ('hour', 0.0033520283), ('day_of_week', 0.00433557), ('month', 0.020924158), ('season', 0.0)], 'PkHorZdrParkMOB': [('PkHorZdrParkMOB', 0.8920228), ('PkHorZdrParkMOB_lag_1h', 0.002192375), ('PkHorZdrParkMOB_lag_2h', 0.0065689054), ('PkHorZdrParkMOB_lag_24h', 0.01146682), ('PkHorZdrParkMOB_lag_48h', 0.004693244), ('PkHorZdrParkMOB_rolling_24h', 0.00807148), ('PkHorZdrParkMOB_rolling_168h', 0.00912024), ('hour', 0.009469519), ('day_of_week', 0.006945384), ('month', 0.04944927), ('season', 0.0)], 'PkJarosPruch': [('PkJarosPruch', 0.93112046), ('PkJarosPruch_lag_1h', 0.010959752), ('PkJarosPruch_lag_2h', 0.0014861166), ('PkJarosPruch_lag_24h', 0.005623129), ('PkJarosPruch_lag_48h', 0.009307028), ('PkJarosPruch_rolling_24h', 0.0070536775), ('PkJarosPruch_rolling_168h', 0.016498603), ('hour', 0.008945946), ('day_of_week', 0.006507308), ('month', 0.00089643), ('season', 0.0016015259)], 'PkMielBierna': [('PkMielBierna', 0.8713314), ('PkMielBierna_lag_1h', 0.013365232), ('PkMielBierna_lag_2h', 0.007831132), ('PkMielBierna_lag_24h', 0.027182434), ('PkMielBierna_lag_48h', 0.016247377), ('PkMielBierna_rolling_24h', 0.0025403732), ('PkMielBierna_rolling_168h', 0.029304659), ('hour', 0.024746845), ('day_of_week', 0.005861269), ('month', 0.0015892375), ('season', 0.0)], 'PkPrzemGrunw': [('PkPrzemGrunw', 0.98119146), ('PkPrzemGrunw_lag_1h', 0.0008660487), ('PkPrzemGrunw_lag_2h', 0.0013510662), ('PkPrzemGrunw_lag_24h', 0.0027273602), ('PkPrzemGrunw_lag_48h', 0.00092109037), ('PkPrzemGrunw_rolling_24h', 0.0003130346), ('PkPrzemGrunw_rolling_168h', 0.0044422997), ('hour', 0.00091355754), ('day_of_week', 0.0021911655), ('month', 0.004710121), ('season', 0.0003728304)], 'PkRymZdrPark': [('PkRymZdrPark', 0.98254365), ('PkRymZdrPark_lag_1h', 0.0015477207), ('PkRymZdrPark_lag_2h', 0.0008015121), ('PkRymZdrPark_lag_24h', 0.0017487804), ('PkRymZdrPark_lag_48h', 0.005791087), ('PkRymZdrPark_rolling_24h', 0.0024527179), ('PkRymZdrPark_rolling_168h', 0.0021488972), ('hour', 0.001330673), ('day_of_week', 0.0012758446), ('month', 0.00019929542), ('season', 0.00015982732)], 'PkRzeszPilsu': [('PkRzeszPilsu', 0.714134), ('PkRzeszPilsu_lag_1h', 0.25924522), ('PkRzeszPilsu_lag_2h', 0.0021322249), ('PkRzeszPilsu_lag_24h', 0.0035428223), ('PkRzeszPilsu_lag_48h', 0.0039259503), ('PkRzeszPilsu_rolling_24h', 0.002430968), ('PkRzeszPilsu_rolling_168h', 0.0058103716), ('hour', 0.0036526143), ('day_of_week', 0.003363857), ('month', 0.0011981329), ('season', 0.00056376186)], 'SkKielTargow': [('SkKielTargow', 0.7287556), ('SkKielTargow_lag_1h', 0.22666314), ('SkKielTargow_lag_2h', 0.0029404138), ('SkKielTargow_lag_24h', 0.011221071), ('SkKielTargow_lag_48h', 0.0072325272), ('SkKielTargow_rolling_24h', 0.0057831006), ('SkKielTargow_rolling_168h', 0.008440704), ('hour', 0.003572477), ('day_of_week', 0.002480516), ('month', 0.0019452185), ('season', 0.0009652783)], 'SkSkarZielnaMOB': [('SkSkarZielnaMOB', 0.90299994), ('SkSkarZielnaMOB_lag_1h', 0.012678475), ('SkSkarZielnaMOB_lag_2h', 0.0031145096), ('SkSkarZielnaMOB_lag_24h', 0.009697774), ('SkSkarZielnaMOB_lag_48h', 0.011096091), ('SkSkarZielnaMOB_rolling_24h', 0.008892066), ('SkSkarZielnaMOB_rolling_168h', 0.03395108), ('hour', 0.0060704704), ('day_of_week', 0.010182767), ('month', 0.0013167715), ('season', 0.0)], 'SkStaraZlota': [('SkStaraZlota', 0.88636625), ('SkStaraZlota_lag_1h', 0.014033812), ('SkStaraZlota_lag_2h', 0.008214181), ('SkStaraZlota_lag_24h', 0.013902602), ('SkStaraZlota_lag_48h', 0.017497774), ('SkStaraZlota_rolling_24h', 0.004538685), ('SkStaraZlota_rolling_168h', 0.027916167), ('hour', 0.0048921104), ('day_of_week', 0.0051250504), ('month', 0.017513333), ('season', 0.0)], 'SlBie

lPartyz': [('SlBielPartyz', 0.9630549), ('SlBielPartyz_lag_1h', 0.0030870931), ('SlBielPartyz_lag_2h', 0.013107657), ('SlBielPartyz_lag_24h', 0.004120705), ('SlBielPartyz_lag_48h', 0.003500067), ('SlBielPartyz_rolling_24h', 0.00261750 5), ('SlBielPartyz_rolling_168h', 0.002322703), ('hour', 0.006876611), ('day_of _week', 0.00081740814), ('month', 0.00049528666), ('season', 0.0)], 'SlKatoKoss ut': [('SlKatoKossut', 0.97892183), ('SlKatoKossut_lag_1h', 0.0028208864), ('Sl KatoKossut_lag_2h', 0.00087171217), ('SlKatoKossut_lag_24h', 0.0041376455), ('S lKatoKossut_lag_48h', 0.0019290664), ('SlKatoKossut_rolling_24h', 0.001241589 2), ('SlKatoKossut_rolling_168h', 0.0026010592), ('hour', 0.0028648276), ('day_ of_week', 0.003816017), ('month', 0.0007953093), ('season', 0.0)], 'SlZlotPotLe s': [('SlZlotPotLes', 0.9684481), ('SlZlotPotLes_lag_1h', 0.0070296074), ('SlZl otPotLes_lag_2h', 0.000632317), ('SlZlotPotLes_lag_24h', 0.0006766683), ('SlZlo tPotLes_lag_48h', 0.0008199918), ('SlZlotPotLes_rolling_24h', 0.0006635905), ('SlZlotPotLes_rolling_168h', 0.0067767105), ('hour', 0.002791263), ('day_of_we ek', 0.005012965), ('month', 0.0071190917), ('season', 2.9706174e-05)], 'WmElbB azynsk': [('WmElbBazynsk', 0.9889994), ('WmElbBazynsk_lag_1h', 0.0016317759), ('WmElbBazynsk_lag_2h', 0.00041838337), ('WmElbBazynsk_lag_24h', 0.0006754789 5), ('WmElbBazynsk_lag_48h', 0.0008929374), ('WmElbBazynsk_rolling_24h', 0.0004 6281982), ('WmElbBazynsk_rolling_168h', 0.00091534556), ('hour', 0.0009148097 6), ('day_of_week', 0.0017089174), ('month', 0.0033801643), ('season', 0.0)], 'WmGoldUzdrowMOB': [('WmGoldUzdrowMOB', 0.9519148), ('WmGoldUzdrowMOB_lag_1h', 0.00243344), ('WmGoldUzdrowMOB_lag_2h', 0.0017706879), ('WmGoldUzdrowMOB_lag_24 h', 0.0069131977), ('WmGoldUzdrowMOB_lag_48h', 0.012700037), ('WmGoldUzdrowMOB_ rolling_24h', 0.007222869), ('WmGoldUzdrowMOB_rolling_168h', 0.0034078564), ('h our', 0.009289349), ('day_of_week', 0.0042652185), ('month', 8.257642e-05), ('s eason', 0.0)], 'WmOlsPuszkin': [('WmOlsPuszkin', 0.6506679), ('WmOlsPuszkin_lag _1h', 0.33187494), ('WmOlsPuszkin_lag_2h', 0.008080101), ('WmOlsPuszkin_lag_24 h', 0.001311905), ('WmOlsPuszkin_lag_48h', 0.0003765571), ('WmOlsPuszkin_rollin g_24h', 0.005284839), ('WmOlsPuszkin_rolling_168h', 0.00051097636), ('hour', 0. 0009949483), ('day_of_week', 0.0), ('month', 0.00089776155), ('season', 0.0)], 'WpKaliSawick': [('WpKaliSawick', 0.9337615), ('WpKaliSawick_lag_1h', 0.0062458 827), ('WpKaliSawick_lag_2h', 0.0059808465), ('WpKaliSawick_lag_24h', 0.0028732 715), ('WpKaliSawick_lag_48h', 0.010809049), ('WpKaliSawick_rolling_24h', 0.011 949369), ('WpKaliSawick_rolling_168h', 0.0034978047), ('hour', 0.014807085), ('day_of_week', 0.008634296), ('month', 0.00042063414), ('season', 0.001020350 5)]}

In [ ]:
```
total_importance_by_feature_type = defaultdict(float)
count_by_feature_type = defaultdict(int)
desired_feature_types = [
    "lag_1h",
    "lag_2h",
    "lag_24h",
    "lag_48h",
    "rolling_24h",
    "rolling_168h",
    "hour",
    "day_of_week",
    "month",
    "season",
]

for features_with_importance in feature_importances_with_names.values():
    for feature, importance in features_with_importance:
        for desired_feature_type in desired_feature_types:
            if desired_feature_type in feature:
                total_importance_by_feature_type[desired_feature_type] += import
                count_by_feature_type[desired_feature_type] += 1
                break
        else:
```

```
            total_importance_by_feature_type["PM2.5"] += importance
            count_by_feature_type["PM2.5"] += 1

average_importance_by_feature_type = {}

for feature_type in desired_feature_types:
    if count_by_feature_type[feature_type] != 0:
        average_importance_by_feature_type[feature_type] = (
            total_importance_by_feature_type[feature_type]
            / count_by_feature_type[feature_type]
        )
    else:
        average_importance_by_feature_type[feature_type] = 0

for station_name, total_importance in total_importance_by_feature_type.items():
    count = count_by_feature_type[station_name]
    if count != 0:
        average_importance_by_feature_type[station_name] = total_importance / co
    else:
        average_importance_by_feature_type[station_name] = 0

sorted_feature_types = sorted(
    average_importance_by_feature_type.items(), key=lambda x: x[1], reverse=True
)

for feature_type, average_importance in sorted_feature_types:
    print(
        f"Typ cechy: {feature_type}, Średnia wartość istotności: {average_import
    )
```

```
Typ cechy: PM2.5, Średnia wartość istotności: 0.9125722234899347
Typ cechy: lag_1h, Średnia wartość istotności: 0.03452801036537081
Typ cechy: day_of_week, Średnia wartość istotności: 0.008178778545475945
Typ cechy: rolling_168h, Średnia wartość istotności: 0.007340613687611866
Typ cechy: hour, Średnia wartość istotności: 0.007221069098436187
Typ cechy: rolling_24h, Średnia wartość istotności: 0.006443819623039169
Typ cechy: lag_24h, Średnia wartość istotności: 0.006279724748512969
Typ cechy: month, Średnia wartość istotności: 0.0062091026859356835
Typ cechy: lag_48h, Średnia wartość istotności: 0.005964279956201195
Typ cechy: lag_2h, Średnia wartość istotności: 0.003599519710976231
Typ cechy: season, Średnia wartość istotności: 0.0016628517396177565
```

Wartości istotności cech w modelu XGBoost określają, jak bardzo poszczególne cechy przyczyniają się do predykcji docelowej zmiennej (w tym przypadku poziomu pyłu PM2.5). Im wyższa wartość istotności cechy, tym większy wpływ ma ona na predykcję. W praktyce istotność cech może być wykorzystana do selekcji cech, redukcji wymiarowości danych lub lepszego zrozumienia procesów zachodzących w modelu. Na podstawie istotności cech możemy podejmować decyzje dotyczące optymalizacji modelu lub dalszych działań analitycznych.

Interpretacja wyników:

- **PM2.5:** Najważniejsza cecha - poziom PM2.5 dla danej stacji bez żadnych obróbek.
- **lag_1h:** Opóźnienie o jedną godzinę - istotne, ale mniej ważne niż PM2.5.
- **day_of_week:** Dzień tygodnia - mały wpływ na predykcje w porównaniu do innych cech.
- **lag_24h:** Opóźnienie o 24 godziny - ważne, ale mniej istotne niż lag_1h.

- **hour:** Godzina dnia - niewielki wpływ na predykcje.
- **rolling_168h:** Średnia krocząca z ostatnich 168 godzin - istotne, ale mniej ważne niż opóźnienia czasowe.

Ogólnie rzecz biorąc dane PM2.5 mają największy wpływ na predykcje, a opóźnienia czasowe (lag_1h, lag_24h) również są istotne. Dni tygodnia, godziny oraz sezon mają mniejsze znaczenie.

Podsumowanie

Projekt miał na celu zastosowanie modelu XGBoost do przewidywania poziomu pyłu PM2.5 na podstawie danych z wielu stacji pomiarowych. Realizacja:

1. **Przygotowanie danych:**
    - Dane zostały poddane analizie wstępnej, która obejmowała usuwanie brakujących wartości i wybór odpowiednich cech do modelowania.
2. **Trenowanie modelu:**
    - Wykorzystano model XGBoost, który jest popularnym modelem do regresji i klasyfikacji w przypadku dużych zbiorów danych.
    - Model został wytrenowany na danych treningowych, a następnie oceniony na zbiorze testowym.
3. **Ocena modelu:**
    - Do oceny wydajności modelu wykorzystano metrykę Mean Squared Error (MSE).
    - Przeprowadzono również backtesting, czyli testowanie modelu na danych historycznych, aby zweryfikować jego skuteczność w przewidywaniu przyszłych wartości.
4. **Analiza istotności cech:**
    - Przeprowadzono analizę istotności cech, aby zidentyfikować najważniejsze czynniki wpływające na predykcję poziomu pyłu PM2.5.
    - W wyniku analizy stwierdzono, że najważniejszą cechą jest historyczne PM2.5.
5. **Podsumowanie i wnioski:**
    - Projekt pozwolił na skuteczne przewidywanie poziomu pyłu PM2.5 na podstawie danych z wielu stacji pomiarowych.
    - Istotność cech pozwoliła zidentyfikować kluczowe czynniki wpływające na predykcje.
    - Wyniki projektu mogą być wykorzystane do podejmowania decyzji dotyczących monitorowania jakości powietrza i podejmowania działań w celu jej poprawy.