Prognoza sprzedaży

Zastosowany zostanie model SARIMAX. Więcej informacji: https://www.bounteous.com/insights/2020/09/15/forecasting-time-series-model-using-python-part-one/

1. Import bibliotek

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import date, timedelta, datetime, timezone
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
import pmdarima as pm
import joblib
import warnings
```

2. Import danych

In [ ]:
```
sale_total = pd.read_excel("sprzedaz_total.xlsx", sheet_name=0)
sale_total.head()
```

Out[ ]:

	data	Między wielkością a zanikiem	Mała degeneracja	Od foliowych czapeczek...	Koniec końca historii	Iluzja wolnej Białorusi	Polska za linią Curzona	Wdowa smoleńska	Rzeczpospol trzecia i
0	2020-10-21	15	9	9	NaN	NaN	NaN	NaN	N
1	2020-10-22	25	11	9	NaN	NaN	NaN	NaN	N
2	2020-10-23	23	6	6	NaN	NaN	NaN	NaN	N
3	2020-10-24	21	4	4	NaN	NaN	NaN	NaN	N
4	2020-10-25	16	7	4	NaN	NaN	NaN	NaN	N

3. Czyszczenie danych

3.1. Przygotowanie zbioru danych do prognoz

In [ ]:
```
# Make the date column a datetime object
```

```
pd.to_datetime(sale_total["data"], format="%d-%m-%y")

# Set date to index (it should be done so!)
sale_total = sale_total.set_index("data")
```

4. Eksploracja całego zbioru

4.1. Sprawdzenie wzorców, jakie powtarzają się w szeregach czasowych

-**Poziom** - średnia wartość w szeregu

-**Trend** - wznoszący, opadający czy stały?

-**Sezonowość**

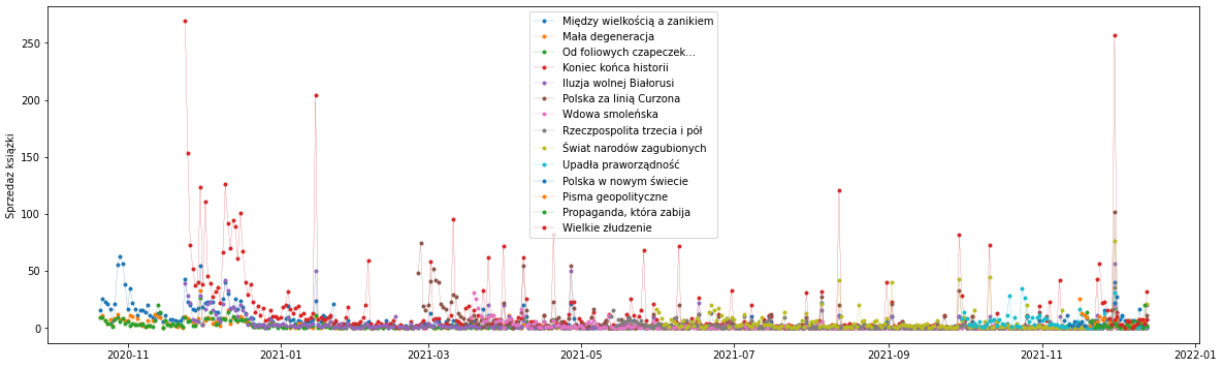
-**Cykliczność**

-**Losowe lub nieregularne zróżnicowanie**

4.2.1. Wykres sprzedaży

Spróbujmy zrobić wykres sprzedaży wszystkich książek dzień po dniu.

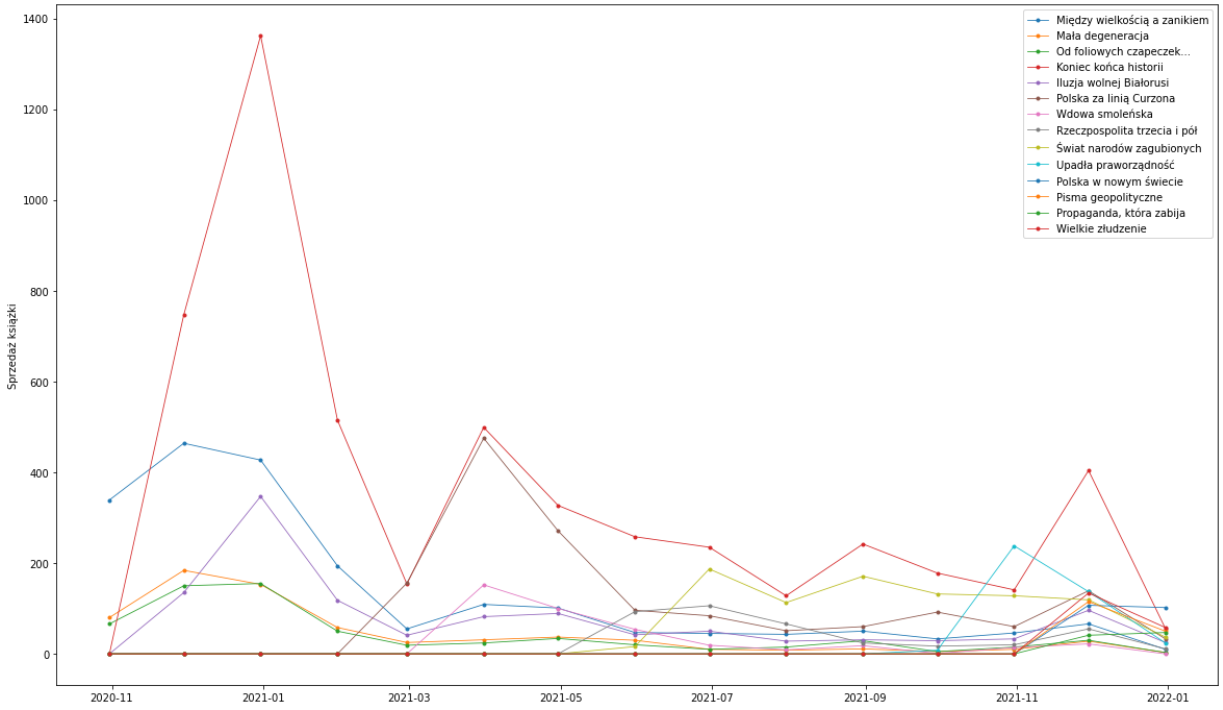
In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(20,6))

# Loop over all books and plot
for book in sale_total:
    y = sale_total[book]
    ax.plot(y, marker=".", linestyle="-", linewidth=0.2, label=book)

ax.set_ylabel("Sprzedaż książki")
ax.legend();
```



Nic nie widać na tym wykresie. A może zmienić okres badany na miesiąc i zmienić wielkość wykresu?

In [ ]:
```
# Resample the dataframe to monthly periods
sale_monthly = sale_total.resample("M").sum()

# Create figure and axes
fig, ax = plt.subplots(figsize=(20,12))

# Loop over all books and plot
for book in sale_monthly:
    y = sale_monthly[book]
    ax.plot(y, marker=".", linestyle="-", linewidth=0.8, label=book)
```

```
ax.set_ylabel("Sprzedaż książki")
ax.legend();
```



Nadal niewiele widać. Trzeba podzielić zbiór danych na mniejsze kawałki - słowem prognozować dla każdej książki z osobna. Zacznijmy od "Między wielkością a zanikiem".

5. "Między wielkością a zanikiem" - prognozy

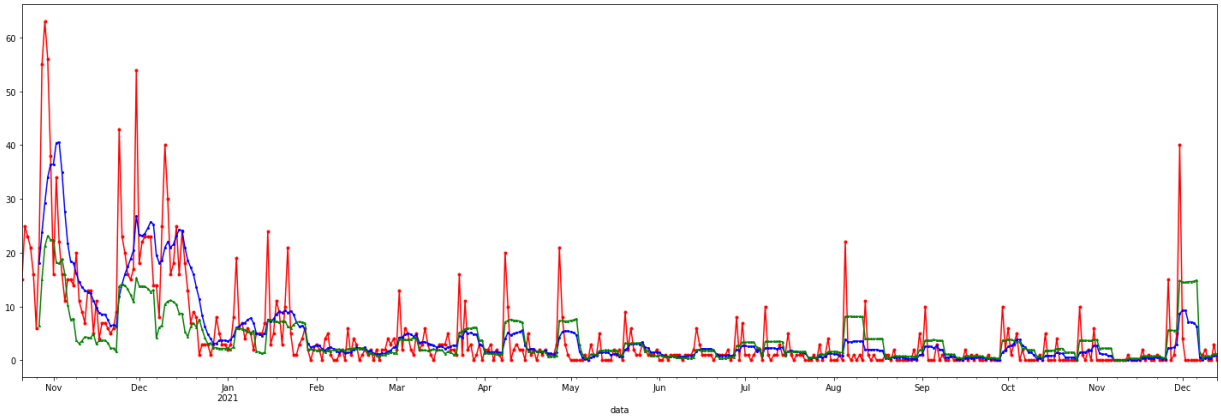
5.1. Eksploracyjna analiza danych

5.1.1. Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Między wielkością a zanikiem"]
```

5.1.2. Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.
Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

5.1.3. Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -3.4104205825150564.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
stacjonarny.
Wartość p: 0.01.
Wartości krytyczne p: {'1%': -3.45, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny. Nie chce się w to za bardzo wierzyć :)
Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-3.4104205825150564,
 0.010608722802444404,
 5,
 413,
 {'1%': -3.4462831955497135,
  '5%': -2.8685636962704395,
  '10%': -2.5705114078759914},
 2444.1422701738015)
```

OK. Jest stacjonarny. Mniej roboty dla mnie.

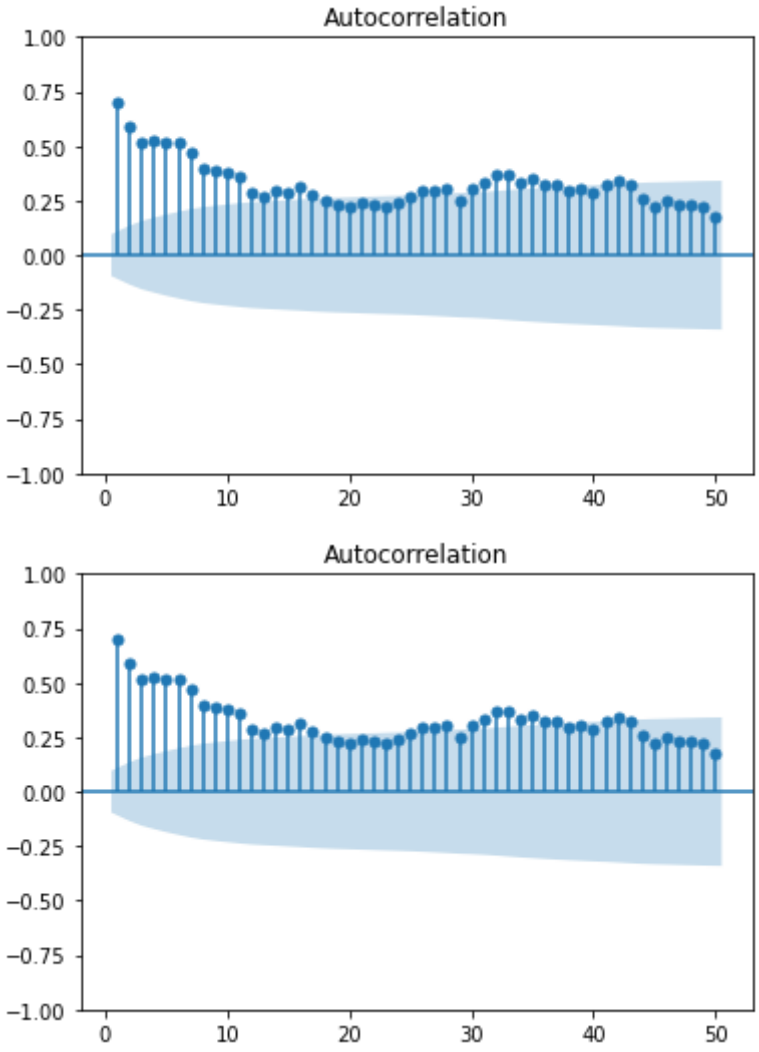
5.1.4. Sprawdzenie sezonowości

Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:



OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

5.2. Tworzenie modelu

5.2.1. Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:
```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
```

```
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
7  2  1  2681.336713  2697.488196
8  2  2  2682.604877  2702.794232
5  1  2  2686.393526  2702.545010
4  1  1  2699.606352  2711.719964
6  2  0  2719.254551  2731.368164
3  1  0  2739.102958  2747.178699
2  0  2  2827.473479  2839.587092
1  0  1  2918.260535  2926.336277
0  0  0  3117.897866  3121.935737
```

5.2.2. Tworzenie i dopasowanie modelu

In [ ]:

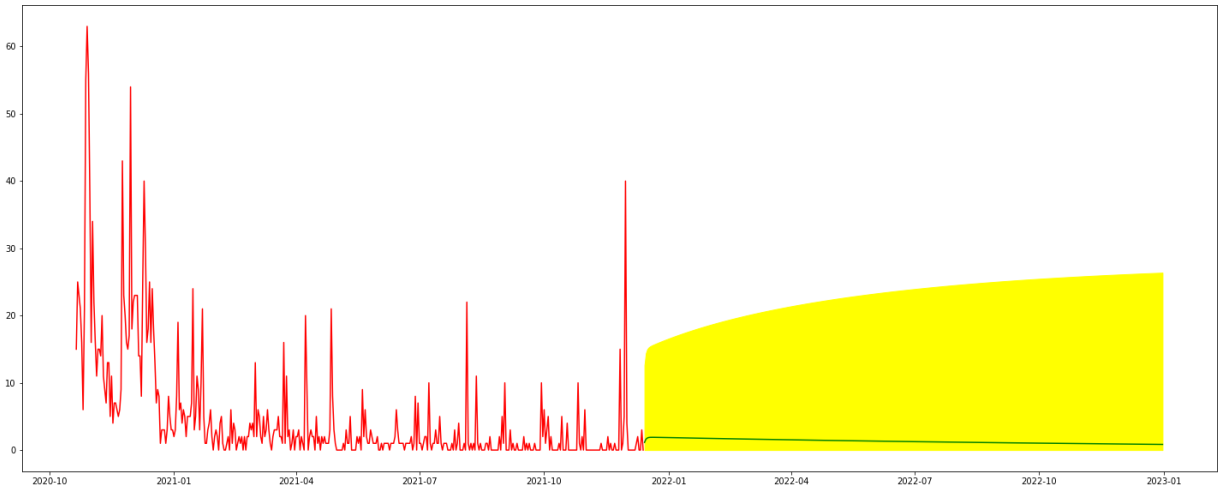
```
# Instantiate a model
model = SARIMAX(df, order=(2,0,1))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

Out[ ]:

```
2021-12-14    1.137917
2021-12-15    1.599691
2021-12-16    1.785627
2021-12-17    1.859035
2021-12-18    1.886540
                ...
2022-12-27    0.849971
2022-12-28    0.848135
2022-12-29    0.846303
2022-12-30    0.844475
2022-12-31    0.842650
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

In [ ]:

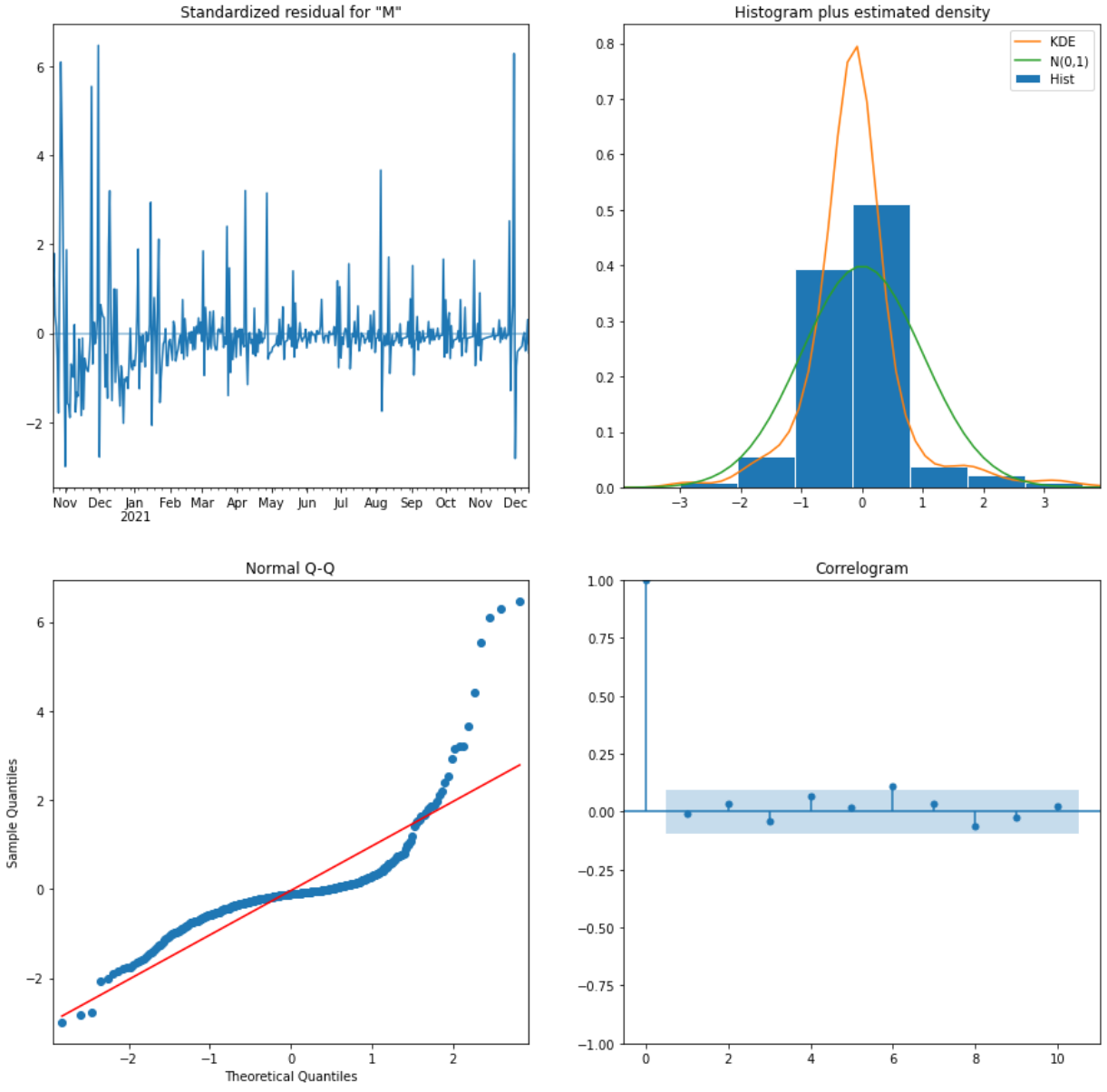
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```



5.3. Diagnostyka

In [ ]:

```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```



5.4. Prognoza

In [ ]:
```
all_books_forecast = pd.DataFrame({"Książka": ["Między wielkością a zanikiem"], "Pes
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857

6. "Mała degeneracja" - prognozy

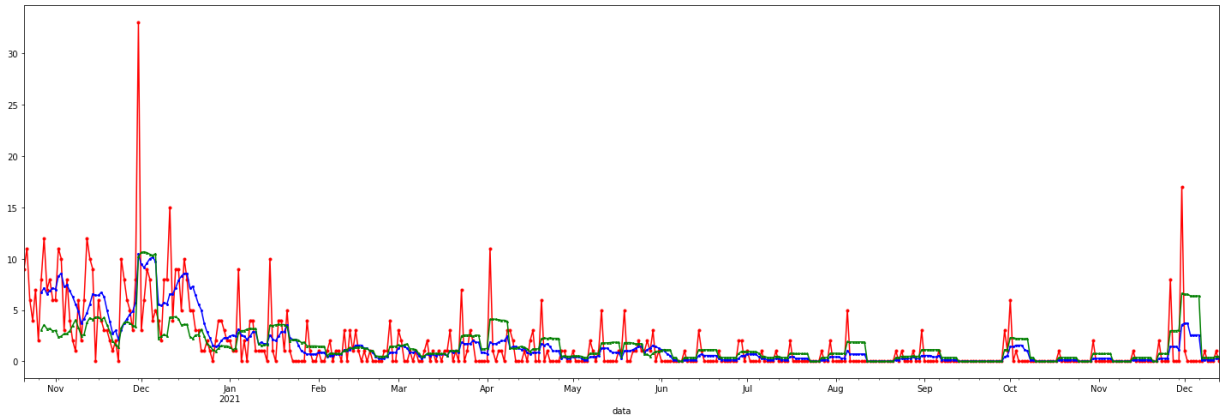
6.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Mała degeneracja"]
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.

Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)
```

```
# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -2.7809590362721024.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
stacjonarny.
Wartość p: 0.06.
Wartości krytyczne p: {'1%': -3.45, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny. Nie chce się w to za bardzo wierzyć :)
Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-2.7809590362721024,
 0.061045530563344566,
 16,
 402,
 {'1%': -3.446722009322339,
  '5%': -2.868756617175256,
  '10%': -2.570614247667137},
 1851.7430260581336)
```

OK. Jest stacjonarny. Mniej roboty dla mnie.

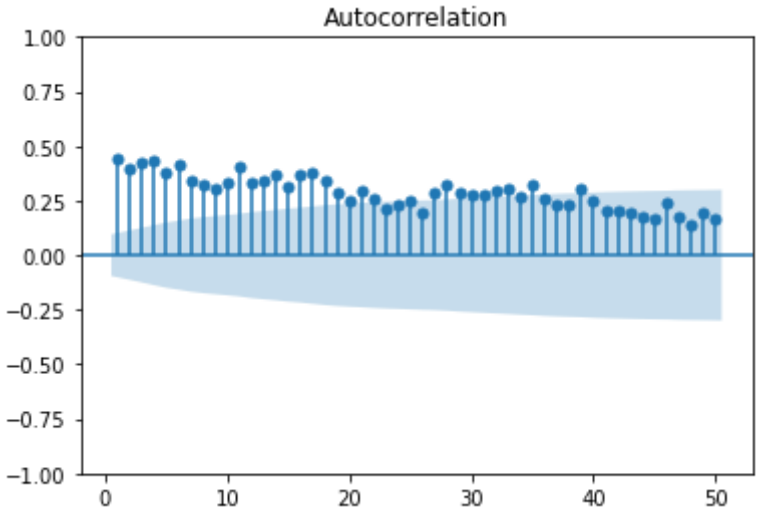
Sprawdzenie sezonowości

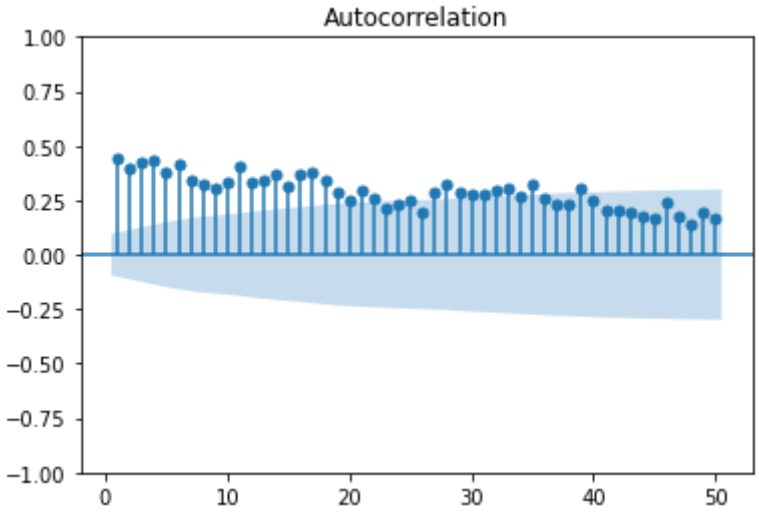
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

6.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:

```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
4  1  1  1964.525751  1976.639364
5  1  2  1965.739526  1981.891010
7  2  1  1965.776448  1981.927932
8  2  2  1968.525728  1988.715083
6  2  0  2049.294952  2061.408565
3  1  0  2090.135778  2098.211520
2  0  2  2122.220228  2134.333841
1  0  1  2154.193127  2162.268869
0  0  0  2246.259417  2250.297288
```

Tworzenie i dopasowanie modelu

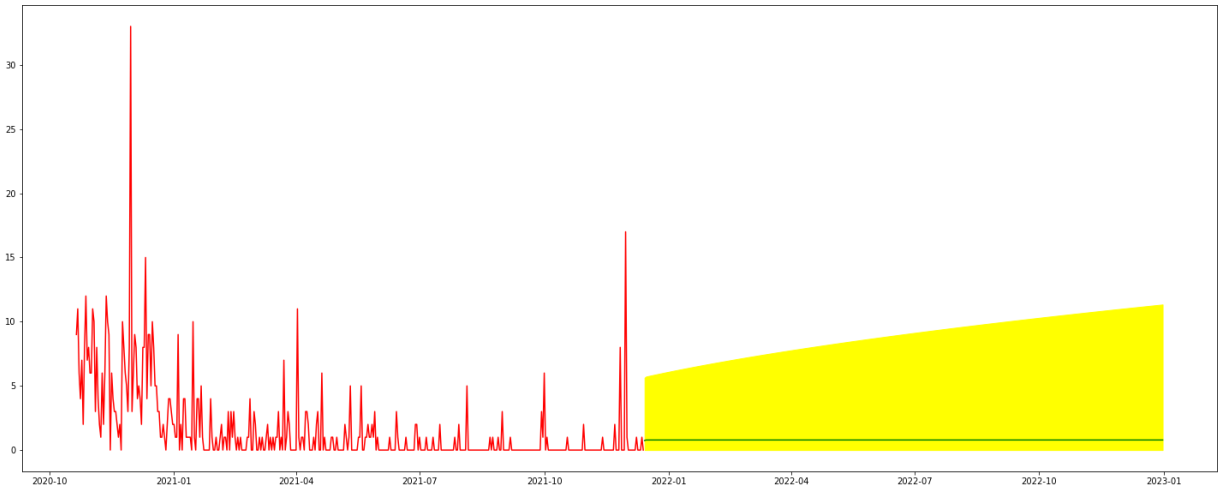
In [ ]:

```
# Instantiate a model
model = SARIMAX(df, order=(1,1,1))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
```

```
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

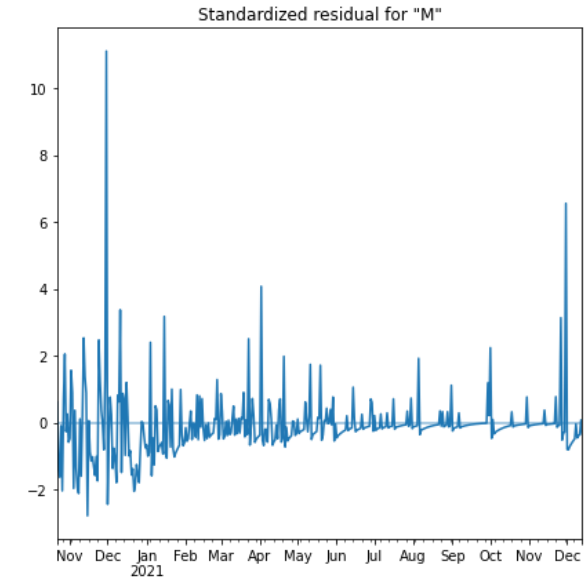
Out[ ]:
```
2021-12-14    0.746380
2021-12-15    0.783097
2021-12-16    0.784904
2021-12-17    0.784992
2021-12-18    0.784997
                ...
2022-12-27    0.784997
2022-12-28    0.784997
2022-12-29    0.784997
2022-12-30    0.784997
2022-12-31    0.784997
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

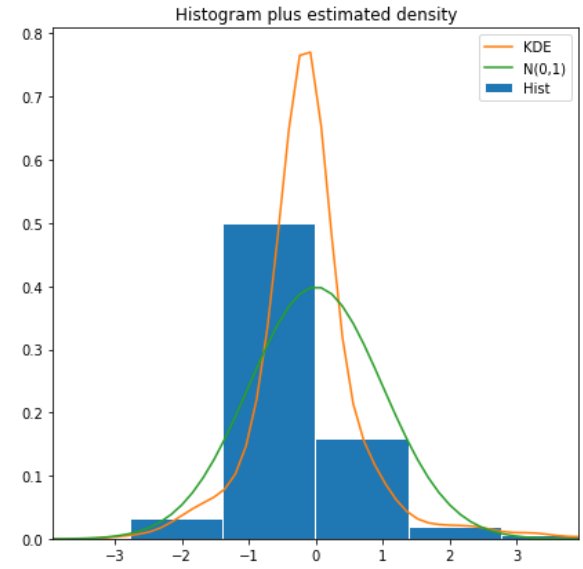
In [ ]:
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```

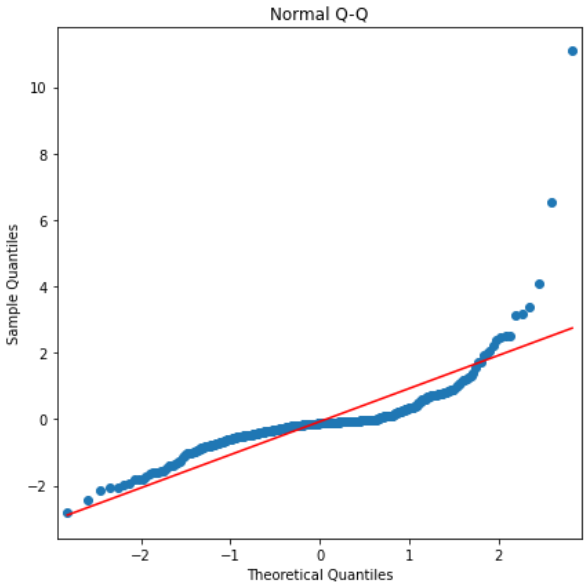


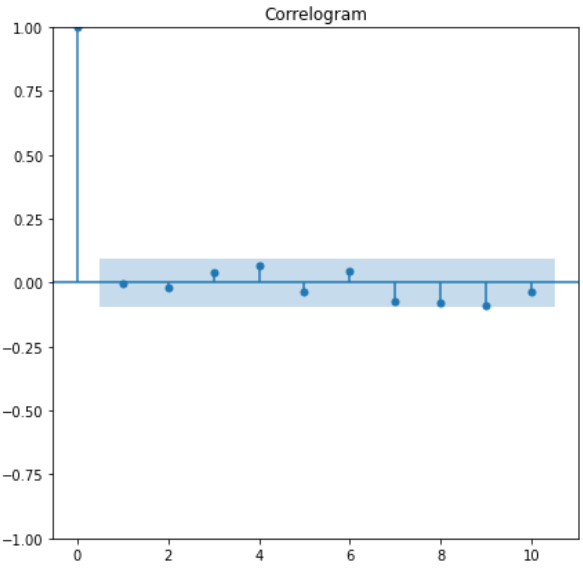
6.3. Diagnostyka

In [ ]:
```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```









6.4. Prognoza

In [ ]:
```
book_2 = {"Książka": "Mała degeneracja", "Pesymistyczny": confidence["lower"].sum(),
all_books_forecast = all_books_forecast.append(book_2, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675

7. "Od foliowych czapeczek" - prognozy

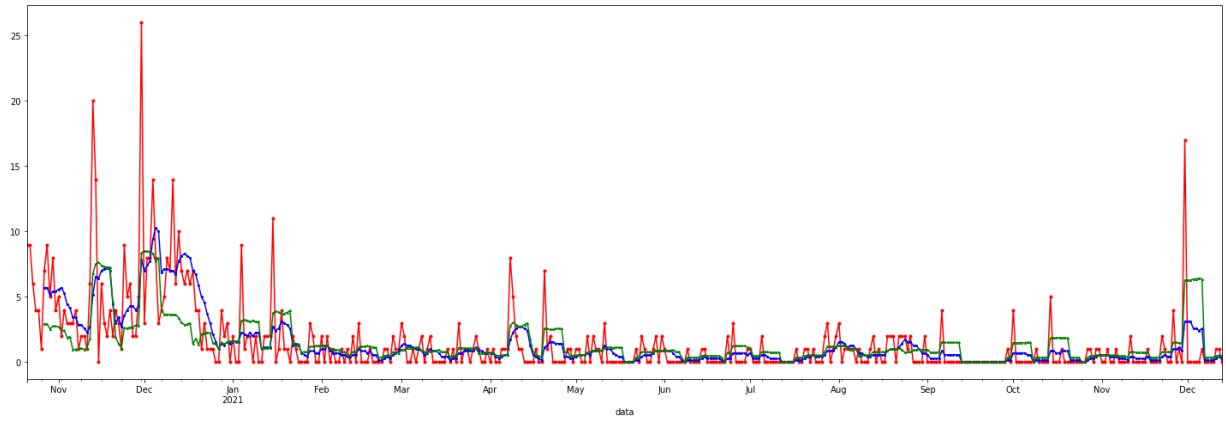
7.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Od foliowych czapeczek..."]
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.
Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -2.024919890023624.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
stacjonarny.
Wartość p: 0.28.
Wartości krytyczne p: {'1%': -3.45, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy nie jest stacjonarny. Trzeba będzie dodać jedną diff.
Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-2.024919890023624,
 0.2757546016639896,
 17,
 401,
 {'1%': -3.4467631030732506,
  '5%': -2.868774682311516,
  '10%': -2.570623877774392},
 1790.871146347616)
```

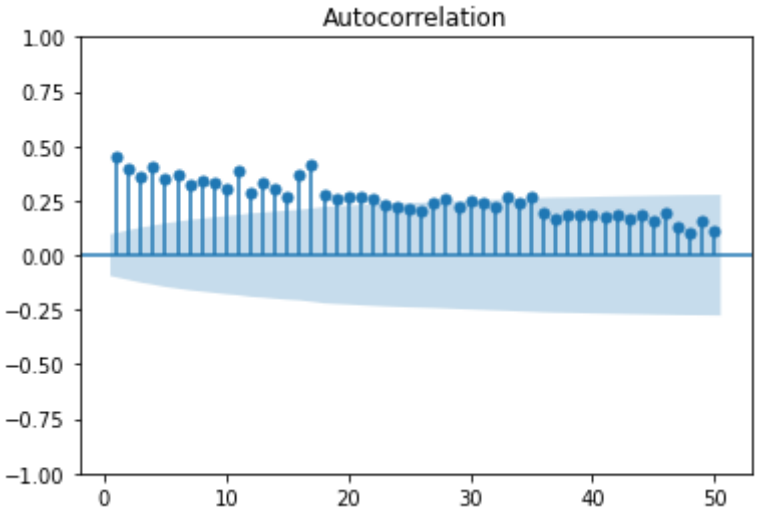
Sprawdzenie sezonowości

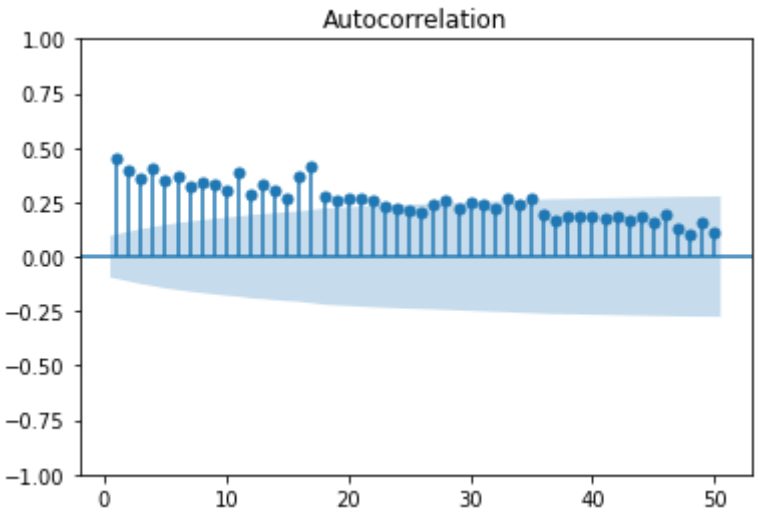
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

7.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:
```
# Ignore warnings
```

```
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
7  2  1  1902.405432  1918.556916
5  1  2  1902.588722  1918.740206
4  1  1  1903.662785  1915.776398
8  2  2  1906.564991  1926.754345
6  2  0  1967.248641  1979.362254
3  1  0  2006.006051  2014.081793
2  0  2  2033.098694  2045.212307
1  0  1  2076.595390  2084.671132
0  0  0  2171.266651  2175.304522
```

Tworzenie i dopasowanie modelu

In [ ]:

```
# Instantiate a model
model = SARIMAX(df, order=(2,1,1))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

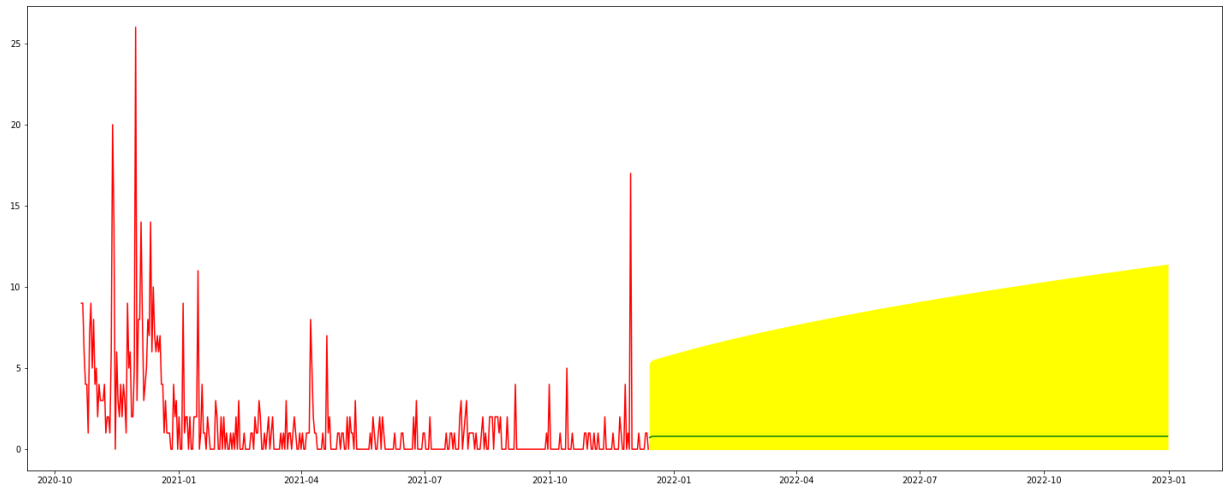
Out[ ]:

```
2021-12-14    0.712334
2021-12-15    0.768097
2021-12-16    0.791287
2021-12-17    0.795206
2021-12-18    0.796196
                ...
2022-12-27    0.796459
2022-12-28    0.796459
2022-12-29    0.796459
2022-12-30    0.796459
2022-12-31    0.796459
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

In [ ]:

```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
```

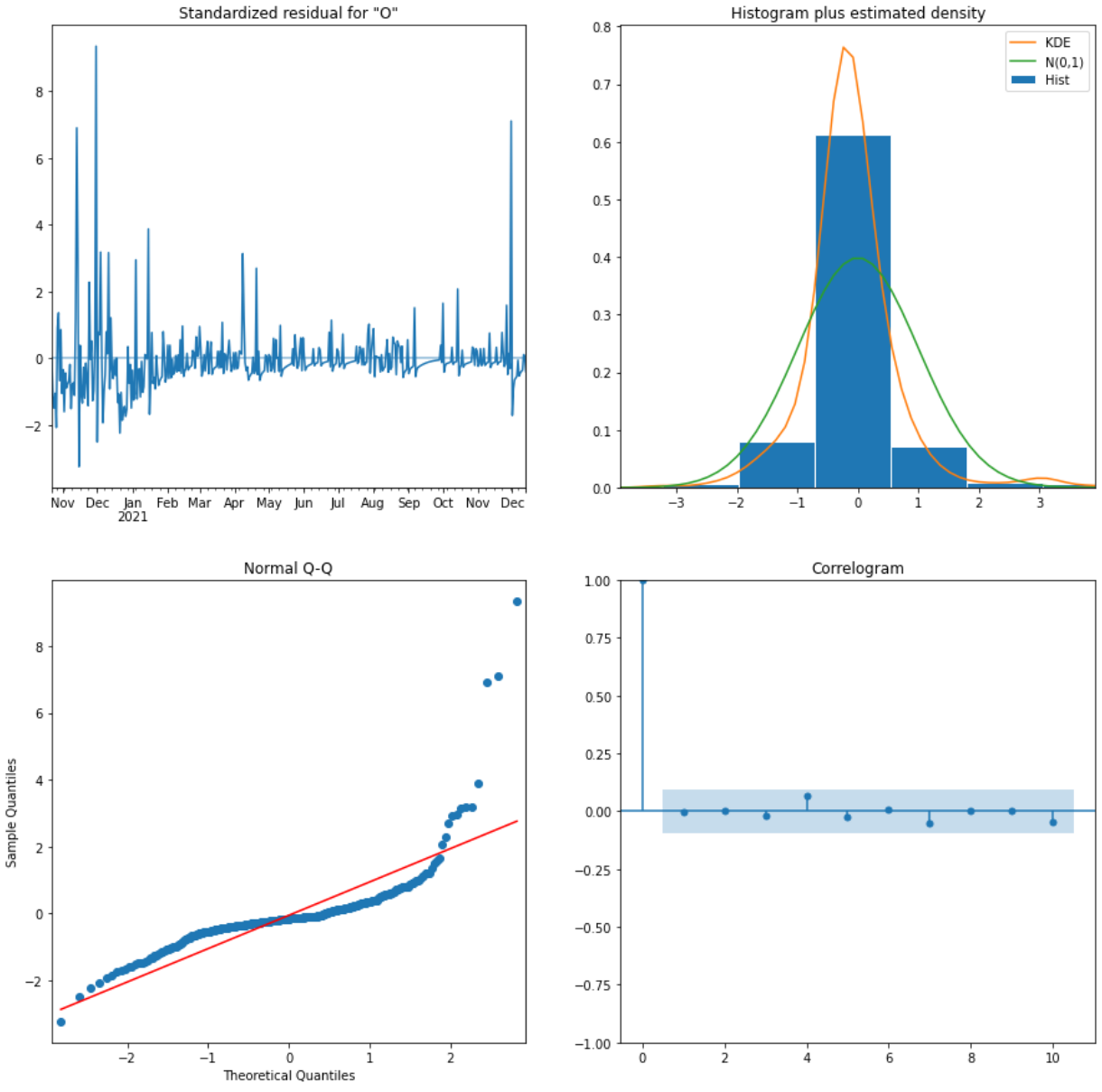
```
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```



7.3. Diagnostyka

In [ ]:

```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```



7.4. Prognoza

In [ ]:
```
book_3 = {"Książka": "Od foliowych czapeczek...", "Pesymistyczny": confidence["lower
all_books_forecast = all_books_forecast.append(book_3, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771

8. "Koniec końca historii" - prognozy

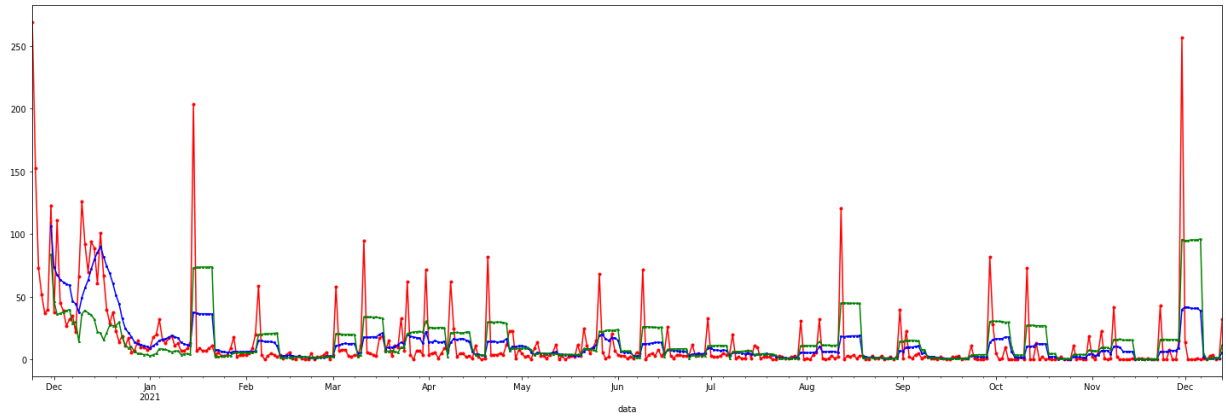
8.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Koniec końca historii"].dropna()
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.
Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -5.602490994554737.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
stacjonarny.
Wartość p: 0.0.
Wartości krytyczne p: {'1%': -3.45, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny.

Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-5.602490994554737,
 1.255550652901804e-06,
 6,
 378,
 {'1%': -3.4477686860685, '5%': -2.869216670067509, '10%': -2.570859500573892},
 3362.7351895472148)
```

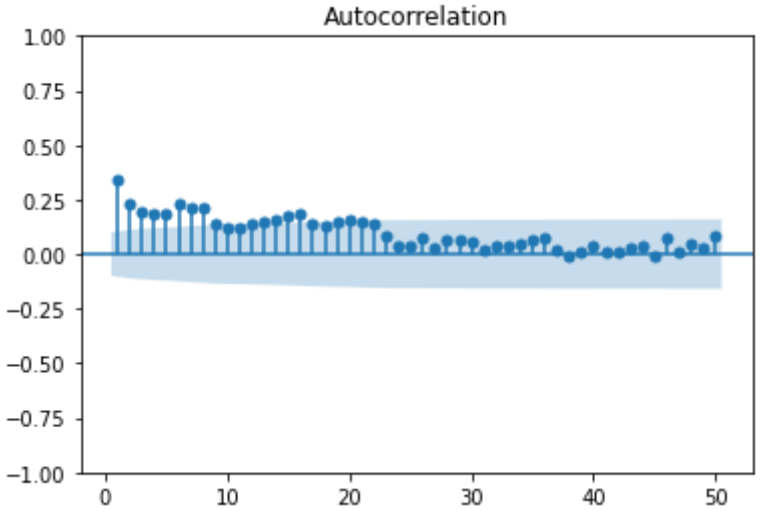
Sprawdzenie sezonowości

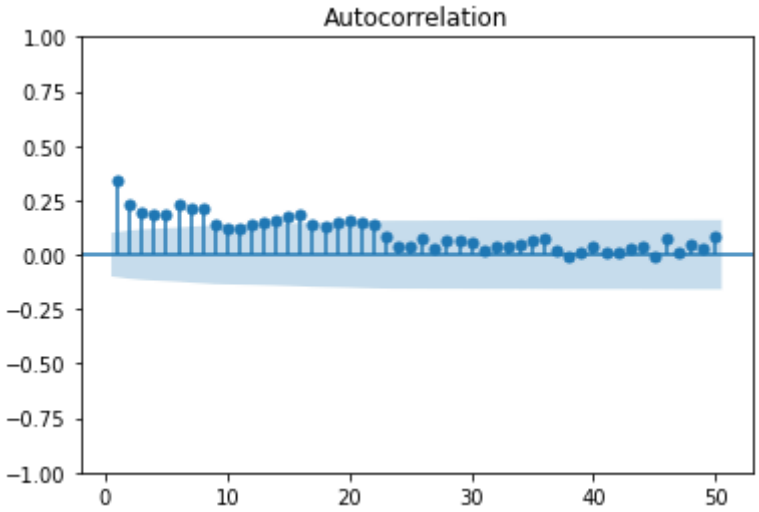
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

8.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:

```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
7  2  1  3627.872600  3643.685574
5  1  2  3628.189695  3644.002669
8  2  2  3629.839058  3649.605274
4  1  1  3631.586761  3643.446491
6  2  0  3671.126096  3682.985826
3  1  0  3692.106484  3700.012971
2  0  2  3706.845803  3718.705533
1  0  1  3730.338180  3738.244667
0  0  0  3791.935630  3795.888873
```

Tworzenie i dopasowanie modelu

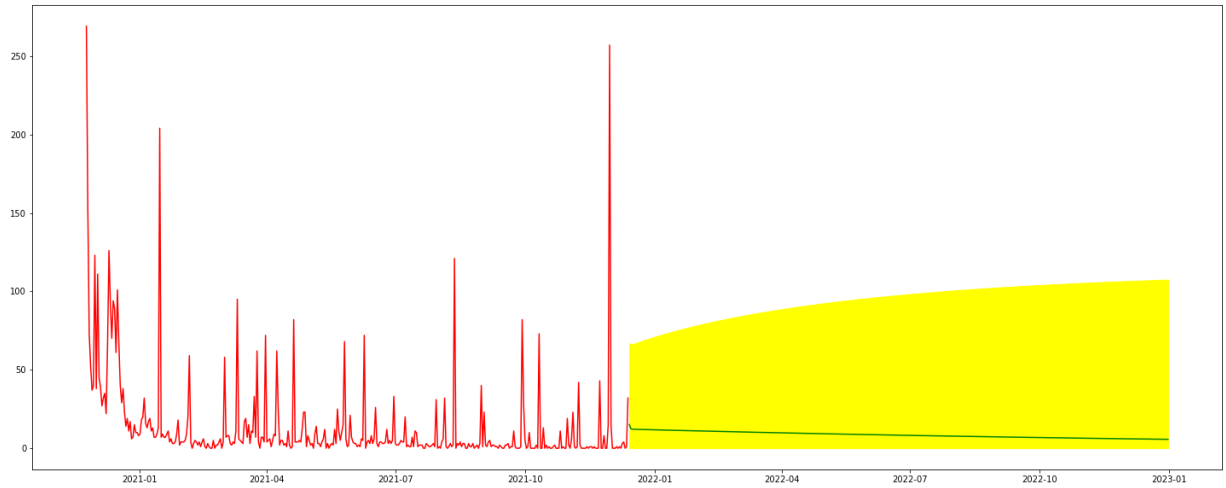
In [ ]:

```
# Instantiate a model
model = SARIMAX(df, order=(2,0,1))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
```

```
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

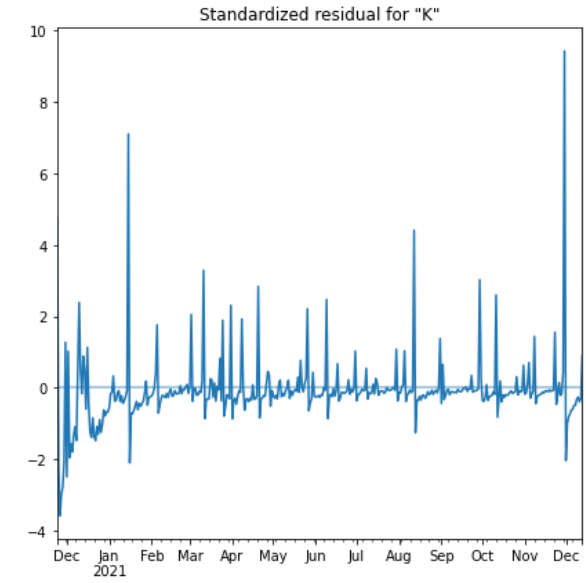
Out[ ]:
```
2021-12-14    15.093617
2021-12-15    12.572370
2021-12-16    12.178852
2021-12-17    12.100085
2021-12-18    12.067913
                ...
2022-12-27     5.722811
2022-12-28     5.711407
2022-12-29     5.700027
2022-12-30     5.688669
2022-12-31     5.677334
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

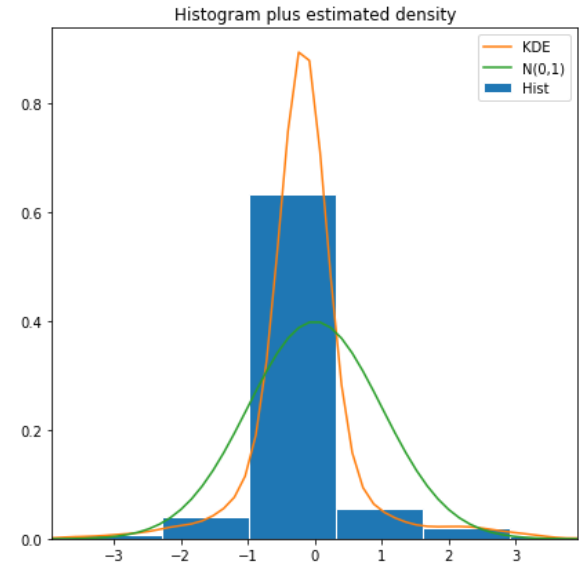
In [ ]:
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```

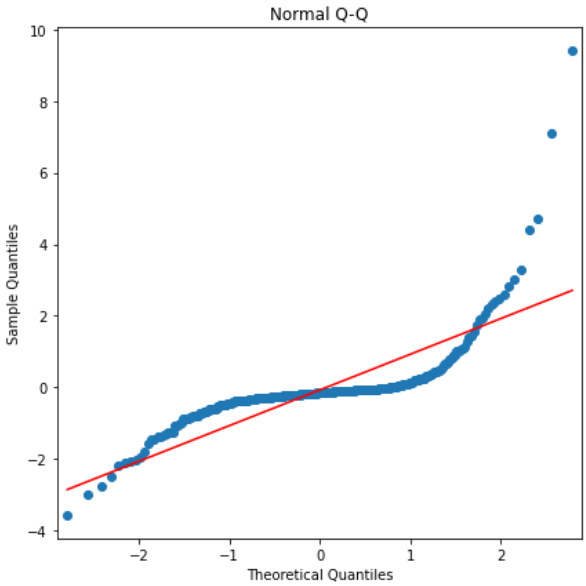


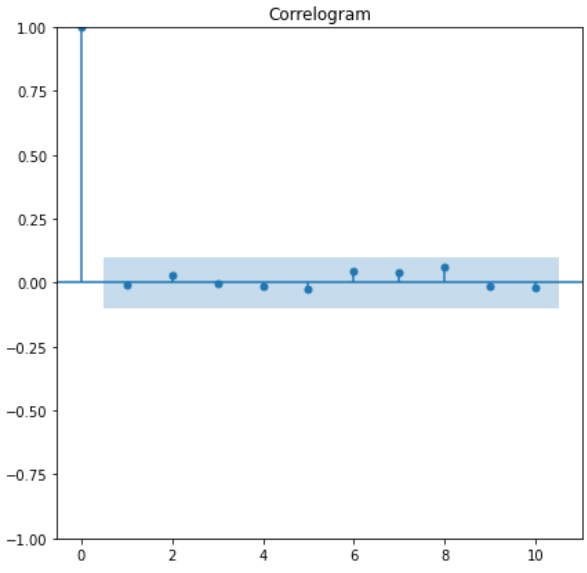
8.3. Diagnostyka

In [ ]:
```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```









8.4. Prognoza

In [ ]:

```
book = {"Książka": "Koniec końca historii", "Pesymistyczny": confidence["lower"].sum
all_books_forecast = all_books_forecast.append(book, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771
3	Koniec końca historii	0	3264.084839	36108.133593

9. "Iluzja wolnej Białorusi" - prognozy

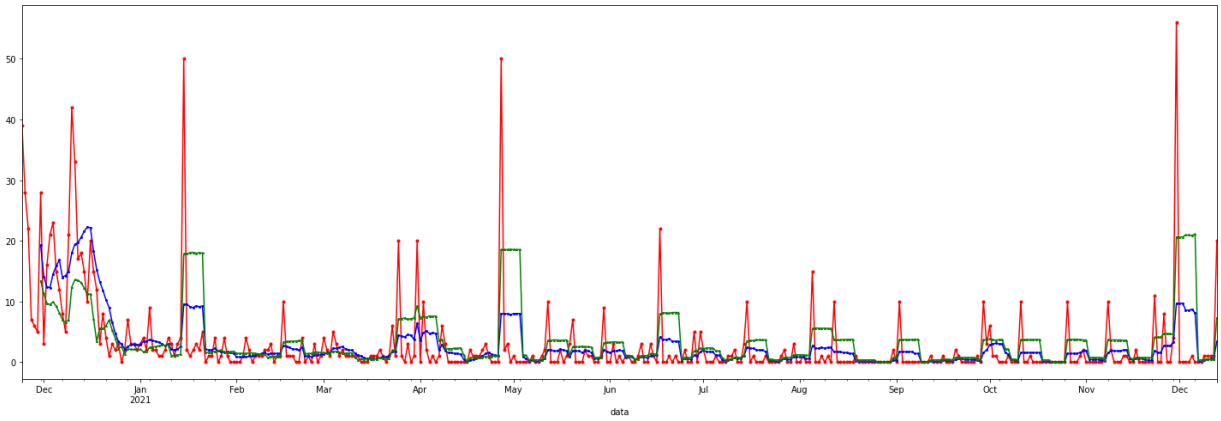
9.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Iluzja wolnej Białorusi"].dropna()
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później.
Zmienia się średnia i odchylenie standardowe.
Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -4.67779383917974.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
stacjonarny.
Wartość p: 0.0.
```

```
Wartości krytyczne p: {'1%': -3.45, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny.
Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-4.67779383917974,
 9.261833888227229e-05,
 5,
 379,
 {'1%': -3.4477224095888497,
  '5%': -2.869196333125208,
  '10%': -2.5708486586002604},
 2320.743085209104)
```

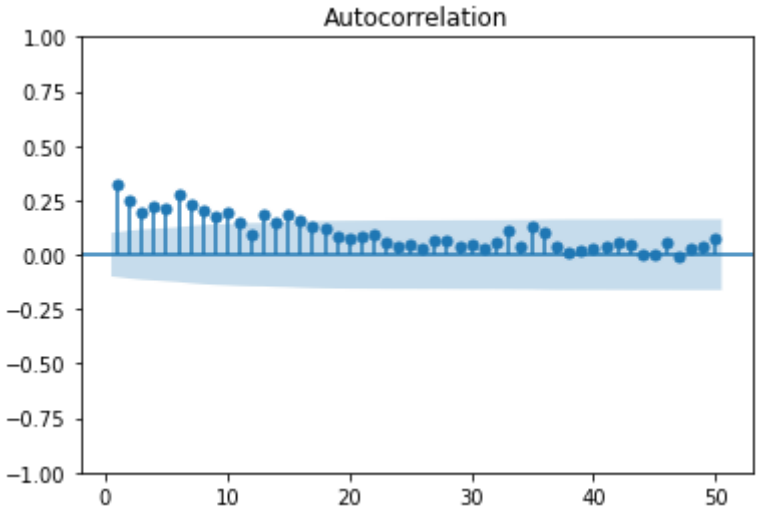
Sprawdzenie sezonowości

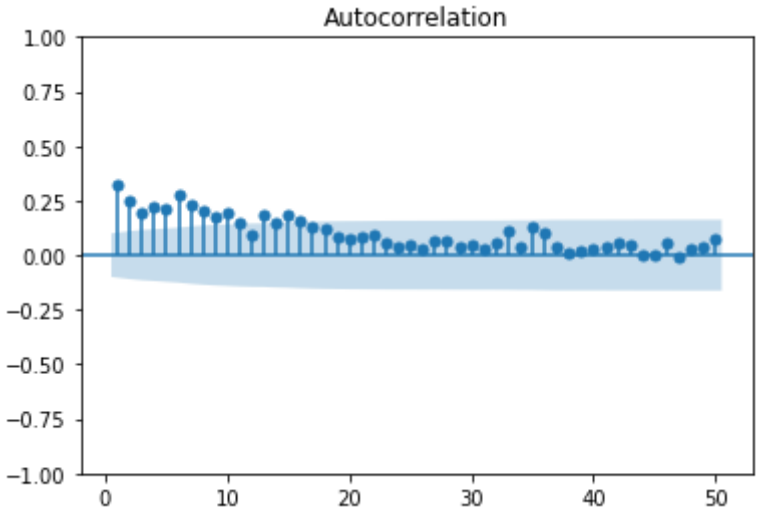
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

9.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:
```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

	p	q	aic	bic
7	2	1	2518.488595	2534.301568
5	1	2	2518.580273	2534.393247
4	1	1	2519.107537	2530.967267
8	2	2	2523.029857	2542.796073
6	2	0	2566.172022	2578.031752
3	1	0	2586.592309	2594.498795
2	0	2	2593.364245	2605.223975
1	0	1	2615.161485	2623.067971
0	0	0	2664.451535	2668.404779

Tworzenie i dopasowanie modelu

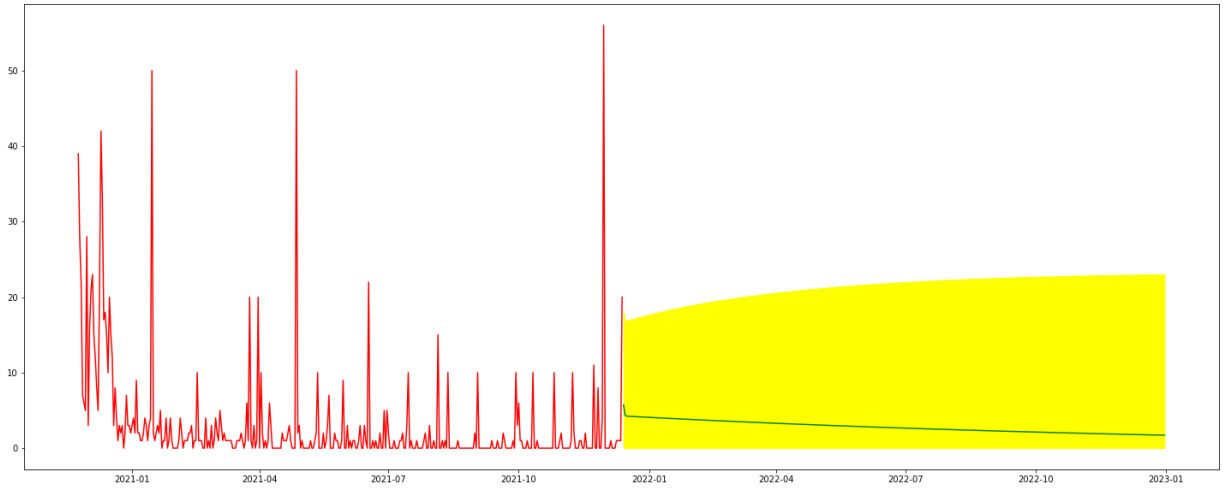
In [ ]:
```
# Instantiate a model
model = SARIMAX(df, order=(2,0,1))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

Out[ ]:
```
2021-12-14    5.741849
2021-12-15    4.384882
2021-12-16    4.247477
2021-12-17    4.225377
2021-12-18    4.214200
                ...
2022-12-27    1.730551
2022-12-28    1.726438
2022-12-29    1.722335
2022-12-30    1.718241
```

```
2022-12-31    1.714157
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

In [ ]:

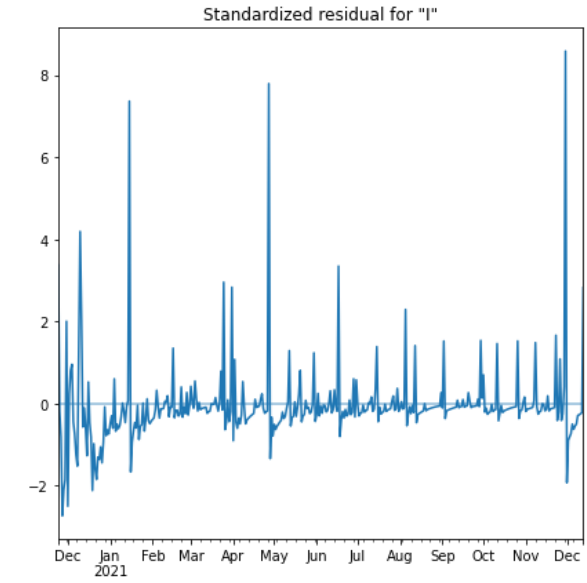
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```

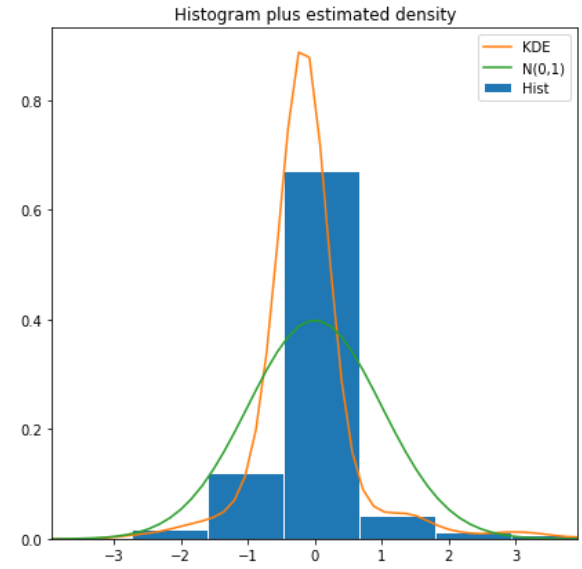


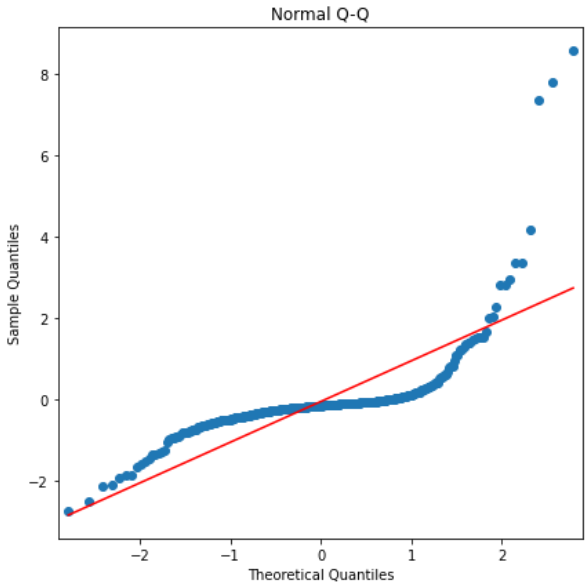
9.3. Diagnostyka

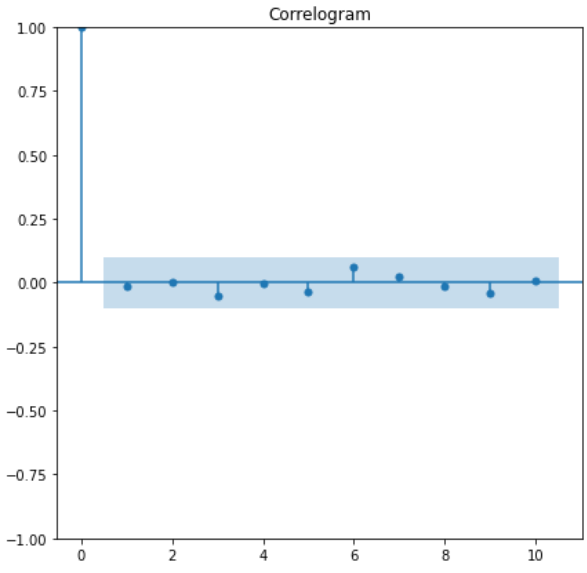
In [ ]:

```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```









9.4. Prognoza

In [ ]:
```
book = {"Książka": "Iluzja wolnej Białorusi", "Pesymistyczny": confidence["lower"].s
all_books_forecast = all_books_forecast.append(book, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771
3	Koniec końca historii	0	3264.084839	36108.133593
4	Iluzja wolnej Białorusi	0	1072.104232	8157.854850

10. "Polska za linią Curzona" - prognozy

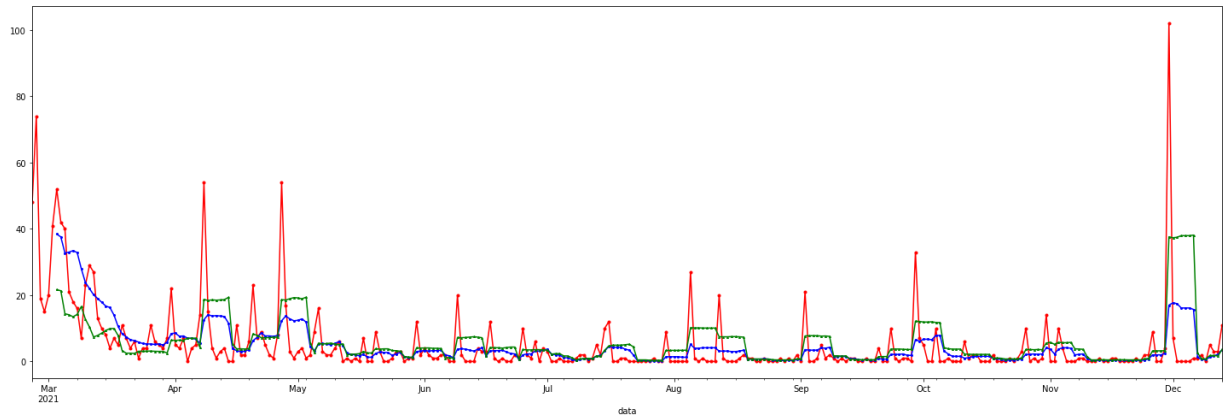
10.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Polska za linią Curzona"].dropna()
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.
Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -11.277728261638877.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
stacjonarny.
Wartość p: 0.0.
```

```
Wartości krytyczne p: {'1%': -3.45, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny.
Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-11.277728261638877,
 1.482224823997384e-20,
 0,
 291,
 {'1%': -3.4530232710482367,
  '5%': -2.871523926671883,
  '10%': -2.5720897694878424},
 1973.7988431388255)
```

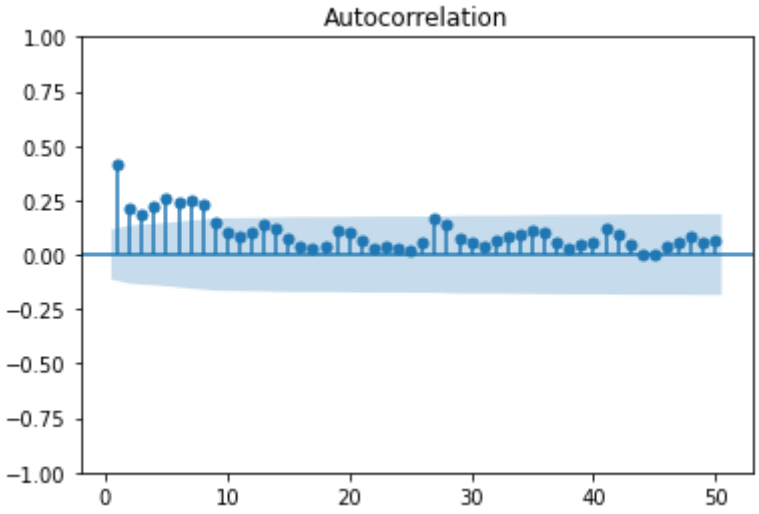
Sprawdzenie sezonowości

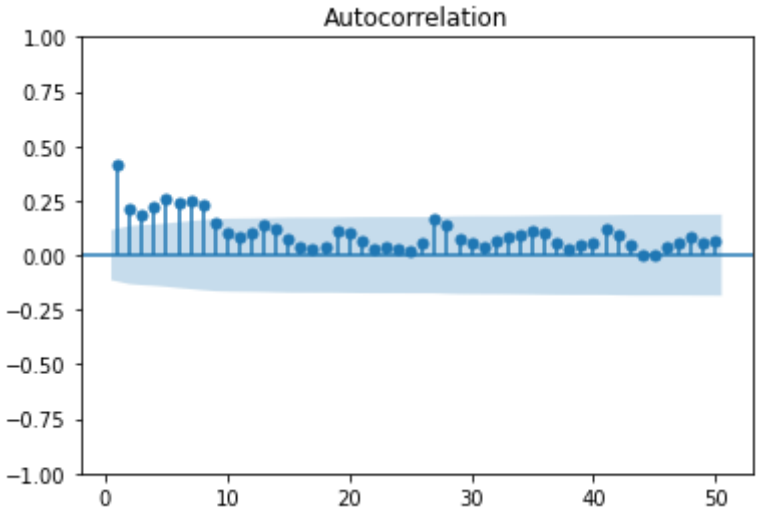
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

10.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:
```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
5  1  2  2166.241476  2180.948491
7  2  1  2167.245343  2181.952359
8  2  2  2168.181387  2186.565156
4  1  1  2172.533640  2183.563901
6  2  0  2203.934835  2214.965096
3  1  0  2208.829573  2216.183081
2  0  2  2221.393659  2232.423921
1  0  1  2236.590767  2243.944274
0  0  0  2300.676512  2304.353266
```

Tworzenie i dopasowanie modelu

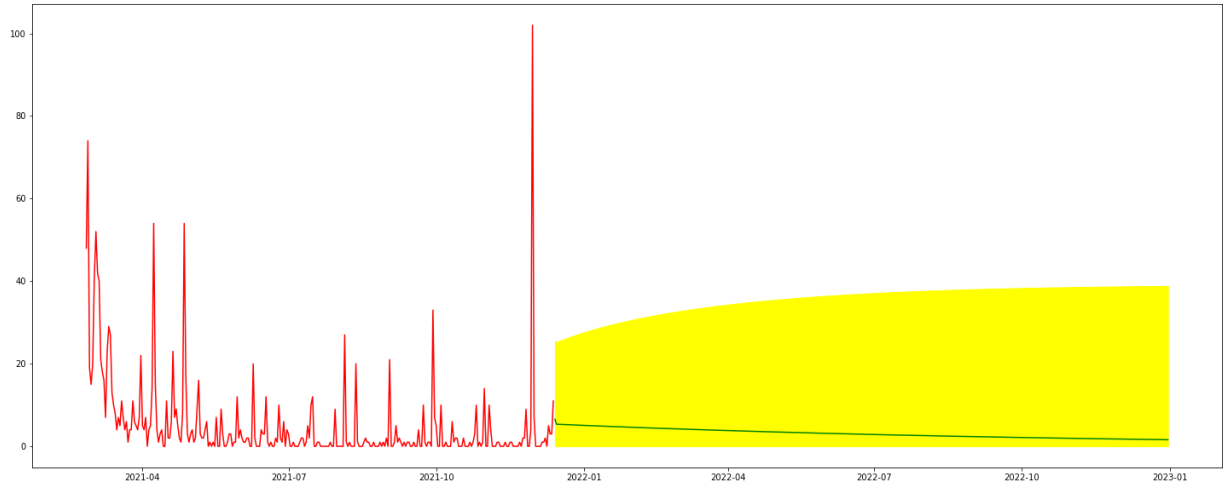
In [ ]:
```
# Instantiate a model
model = SARIMAX(df, order=(1,0,2))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

Out[ ]:
```
2021-12-14    6.540102
2021-12-15    5.350445
2021-12-16    5.333522
2021-12-17    5.316653
2021-12-18    5.299837
                ...
2022-12-27    1.620741
2022-12-28    1.615614
2022-12-29    1.610504
2022-12-30    1.605410
```

```
2022-12-31    1.600333
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

In [ ]:

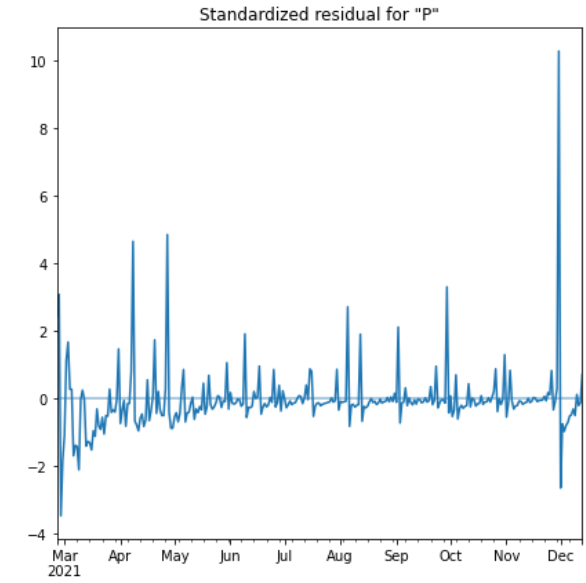
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```

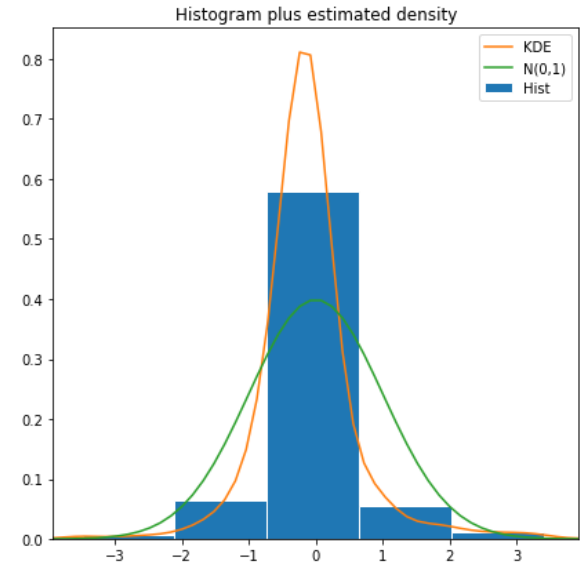


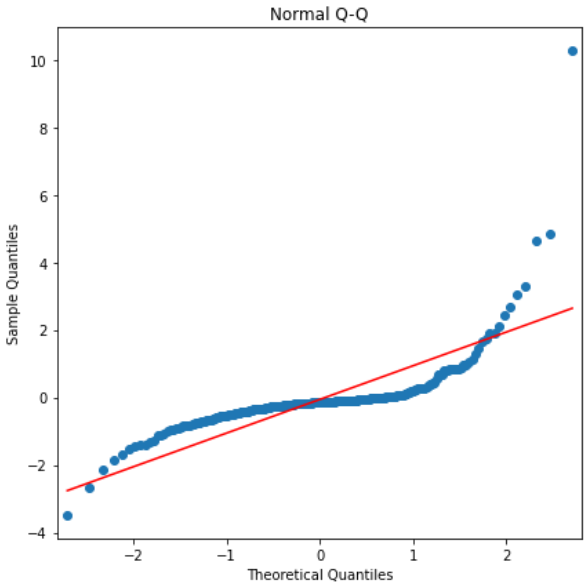
10.3. Diagnostyka

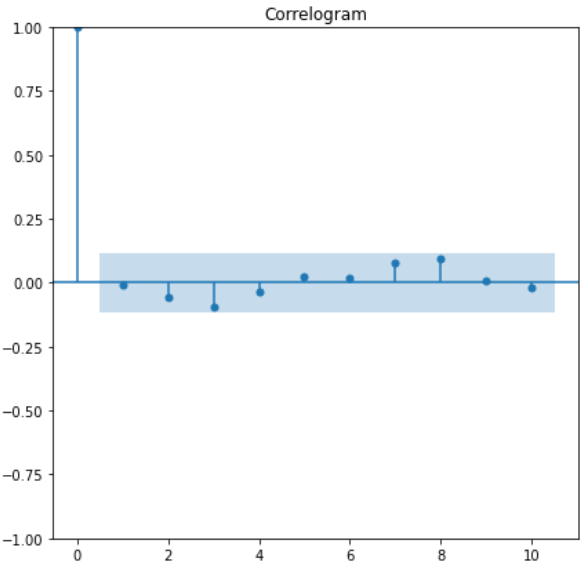
In [ ]:

```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```









10.4. Prognoza

In [ ]:

```
book = {"Książka": "Polska za linią Curzona", "Pesymistyczny": confidence["lower"].s
all_books_forecast = all_books_forecast.append(book, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771
3	Koniec końca historii	0	3264.084839	36108.133593
4	Iluzja wolnej Białorusi	0	1072.104232	8157.854850
5	Polska za linią Curzona	0	1193.803081	13610.228690

11. "Wdowa smoleńska" - prognozy

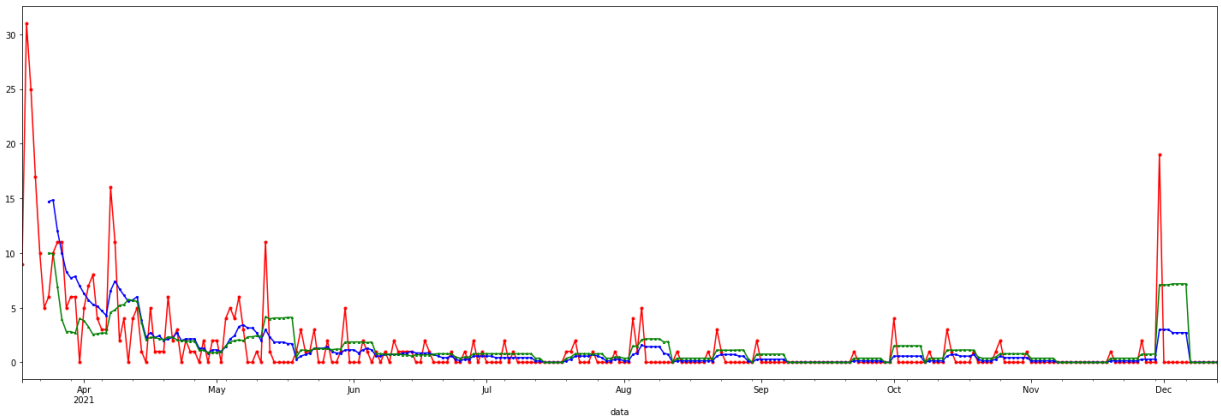
11.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Wdowa smoleńska"].dropna()
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.

Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -4.3511361745298345.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest
```

```
stacjonarny.
Wartość p: 0.0.
Wartości krytyczne p: {'1%': -3.46, '5%': -2.87, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny.

Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-4.3511361745298345,
 0.0003618573032977733,
 16,
 254,
 {'1%': -3.456360306409983,
  '5%': -2.8729872043802356,
  '10%': -2.572870232500465},
 1069.423331707247)
```

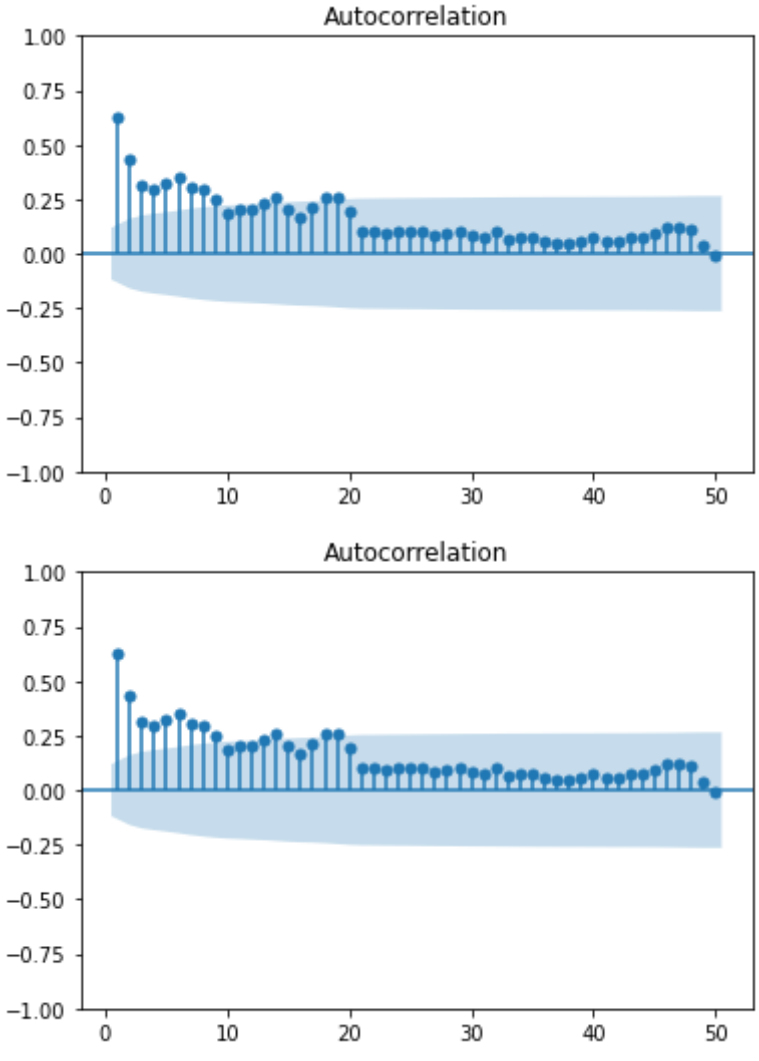
Sprawdzenie sezonowości

Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:



OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

11.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:
```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

	p	q	aic	bic
5	1	2	1297.669282	1312.077758
7	2	1	1298.099265	1312.507740
8	2	2	1299.495932	1317.506526
4	1	1	1304.956353	1315.762710
6	2	0	1327.731510	1338.537866
3	1	0	1333.796936	1341.001174
2	0	2	1361.201627	1372.007984
1	0	1	1397.392983	1404.597221
0	0	0	1503.073164	1506.675283

Tworzenie i dopasowanie modelu

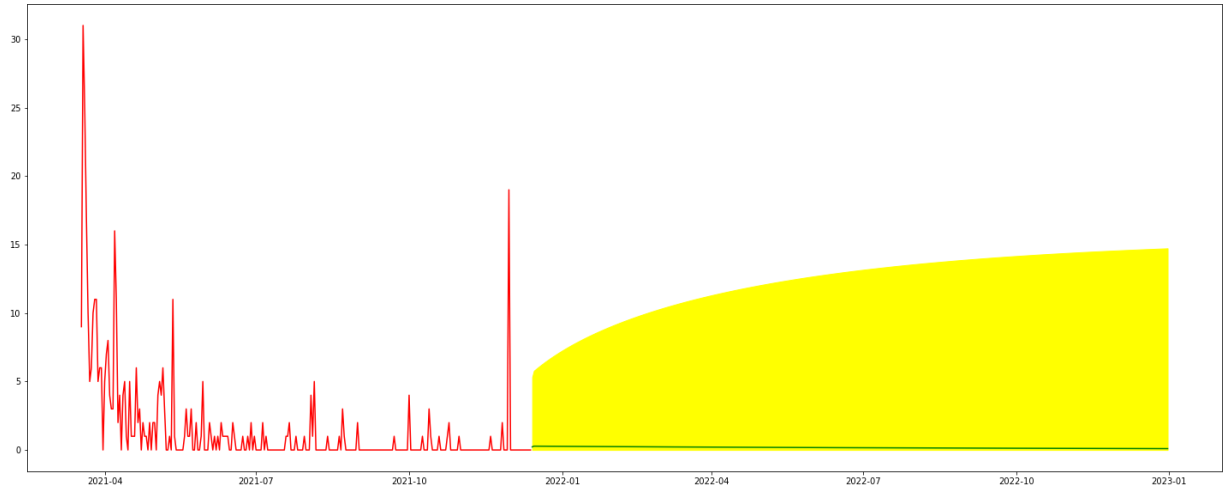
In [ ]:
```
# Instantiate a model
model = SARIMAX(df, order=(1,0,2))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

Out[ ]:
```
2021-12-14    0.225592
2021-12-15    0.278922
2021-12-16    0.278161
2021-12-17    0.277403
2021-12-18    0.276647
                ...
2022-12-27    0.099683
2022-12-28    0.099411
2022-12-29    0.099140
```

```
2022-12-30    0.098870
2022-12-31    0.098601
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

In [ ]:

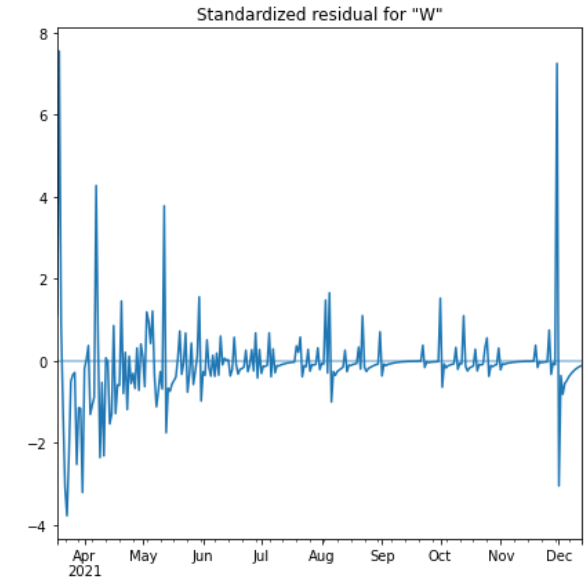
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```

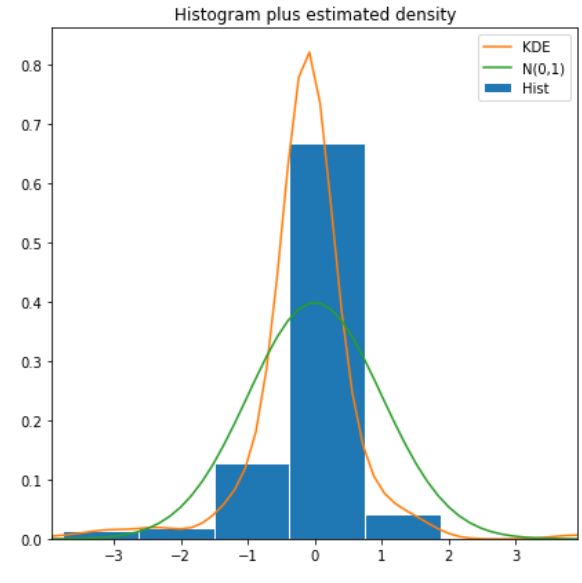


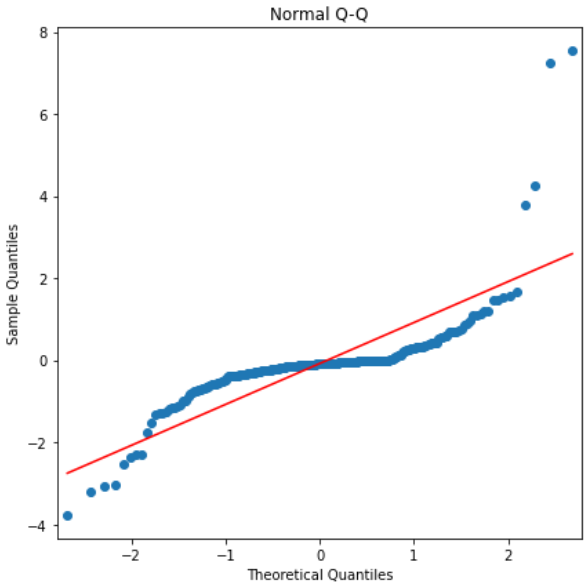
11.3. Diagnostyka

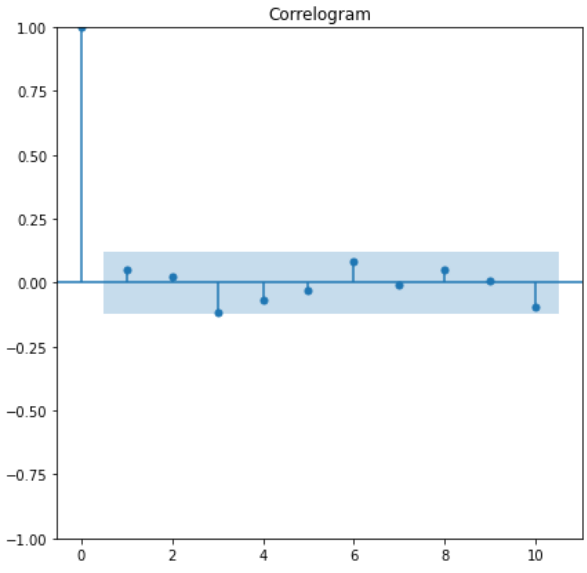
In [ ]:

```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```









11.4. Prognoza

In [ ]:

```
book = {"Książka": "Wdowa smoleńska", "Pesymistyczny": confidence["lower"].sum(), "U
all_books_forecast = all_books_forecast.append(book, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771
3	Koniec końca historii	0	3264.084839	36108.133593
4	Iluzja wolnej Białorusi	0	1072.104232	8157.854850
5	Polska za linią Curzona	0	1193.803081	13610.228690
6	Wdowa smoleńska	0	66.483639	4691.634338

12. "Rzeczpospolita trzecia i pół" - prognozy

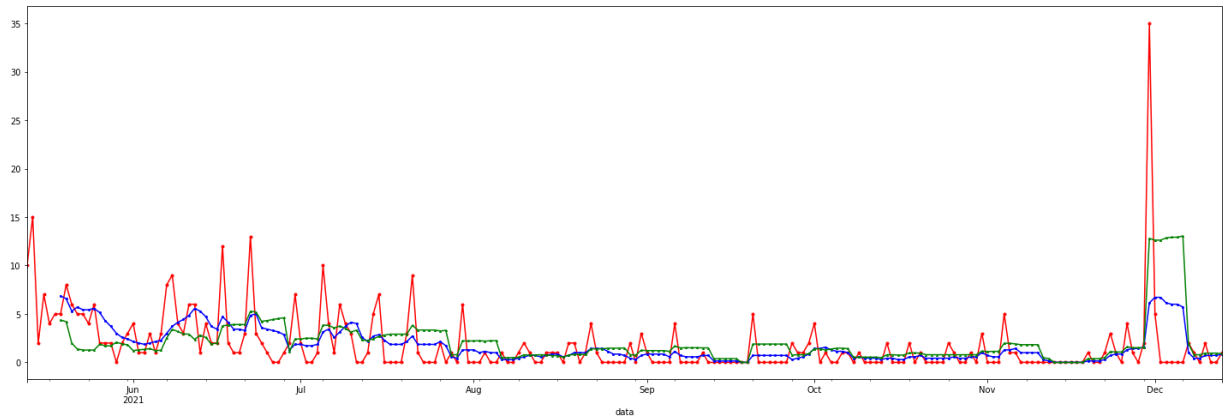
12.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Rzeczpospolita trzecia i pół"].dropna()
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.

Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA
```

```
Wartość testu Dickeya-Fullera: -11.239836240200587.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest stacjonarny.
Wartość p: 0.0.
Wartości krytyczne p: {'1%': -3.46, '5%': -2.88, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny.

Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-11.239836240200587,
 1.818637392851836e-20,
 0,
 214,
 {'1%': -3.4612821203214907,
  '5%': -2.875142613826617,
  '10%': -2.574020122281422},
 1029.788490761346)
```

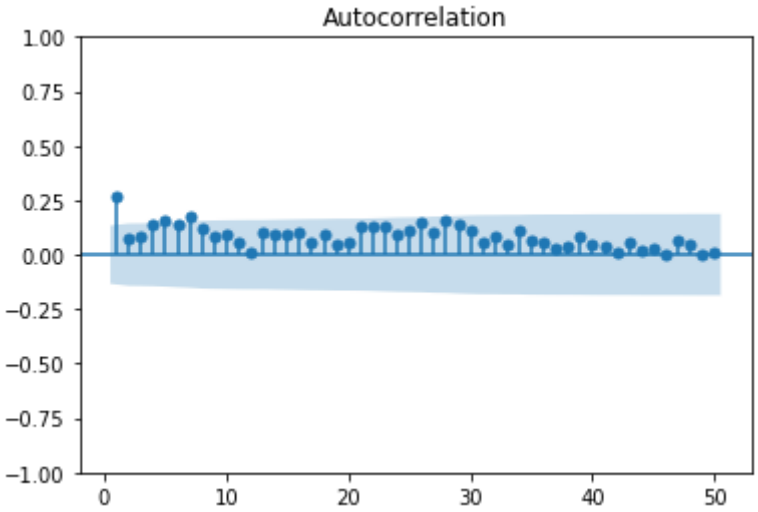
Sprawdzenie sezonowości

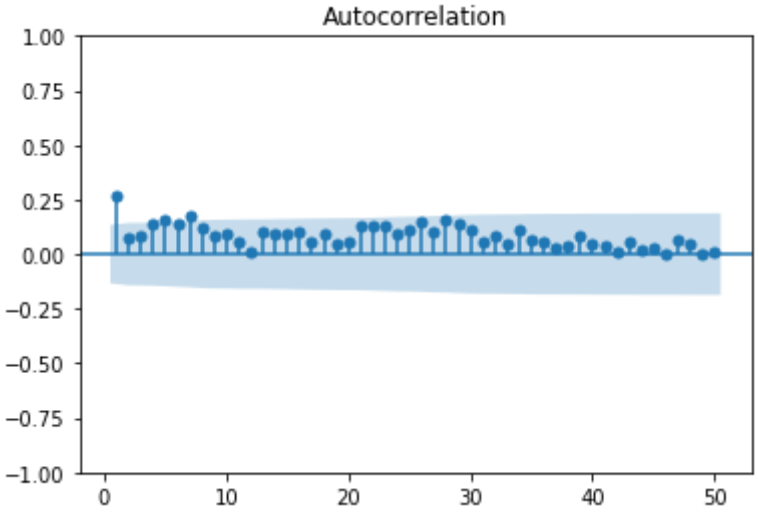
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

12.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:
```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
5  1  2  1120.165232  1133.647785
7  2  1  1120.920428  1134.402980
8  2  2  1121.798592  1138.651782
4  1  1  1123.009320  1133.121234
6  2  0  1151.324679  1161.436593
3  1  0  1152.697963  1159.439239
2  0  2  1159.051833  1169.163748
1  0  1  1162.933425  1169.674701
0  0  0  1194.574270  1197.944908
```

Tworzenie i dopasowanie modelu

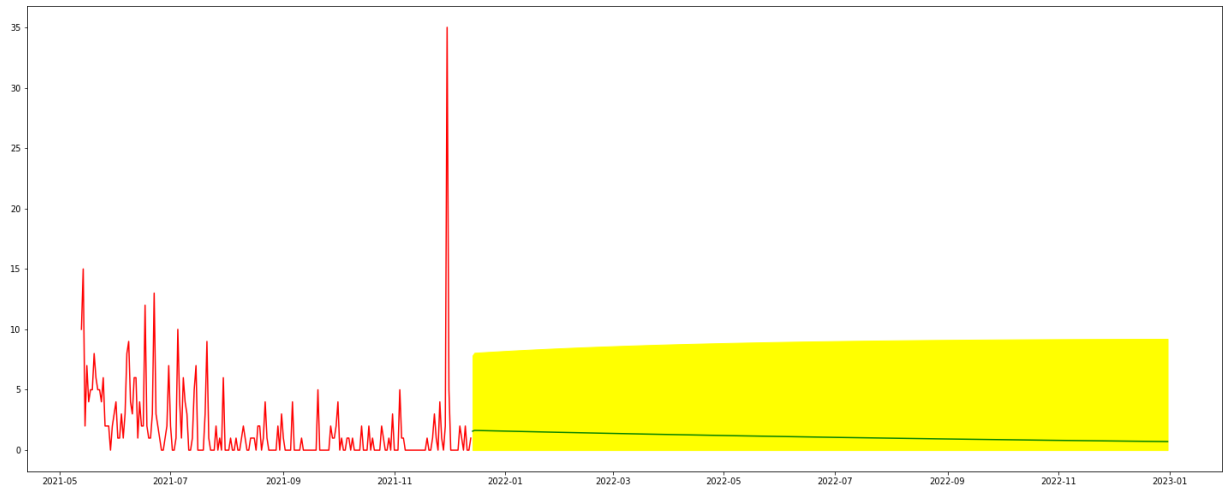
In [ ]:
```
# Instantiate a model
model = SARIMAX(df, order=(1,0,2))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

Out[ ]:
```
2021-12-14    1.568161
2021-12-15    1.634139
2021-12-16    1.630562
2021-12-17    1.626993
2021-12-18    1.623432
                ...
```

```
2022-12-27    0.715354
2022-12-28    0.713788
2022-12-29    0.712226
2022-12-30    0.710667
2022-12-31    0.709112
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

In [ ]:

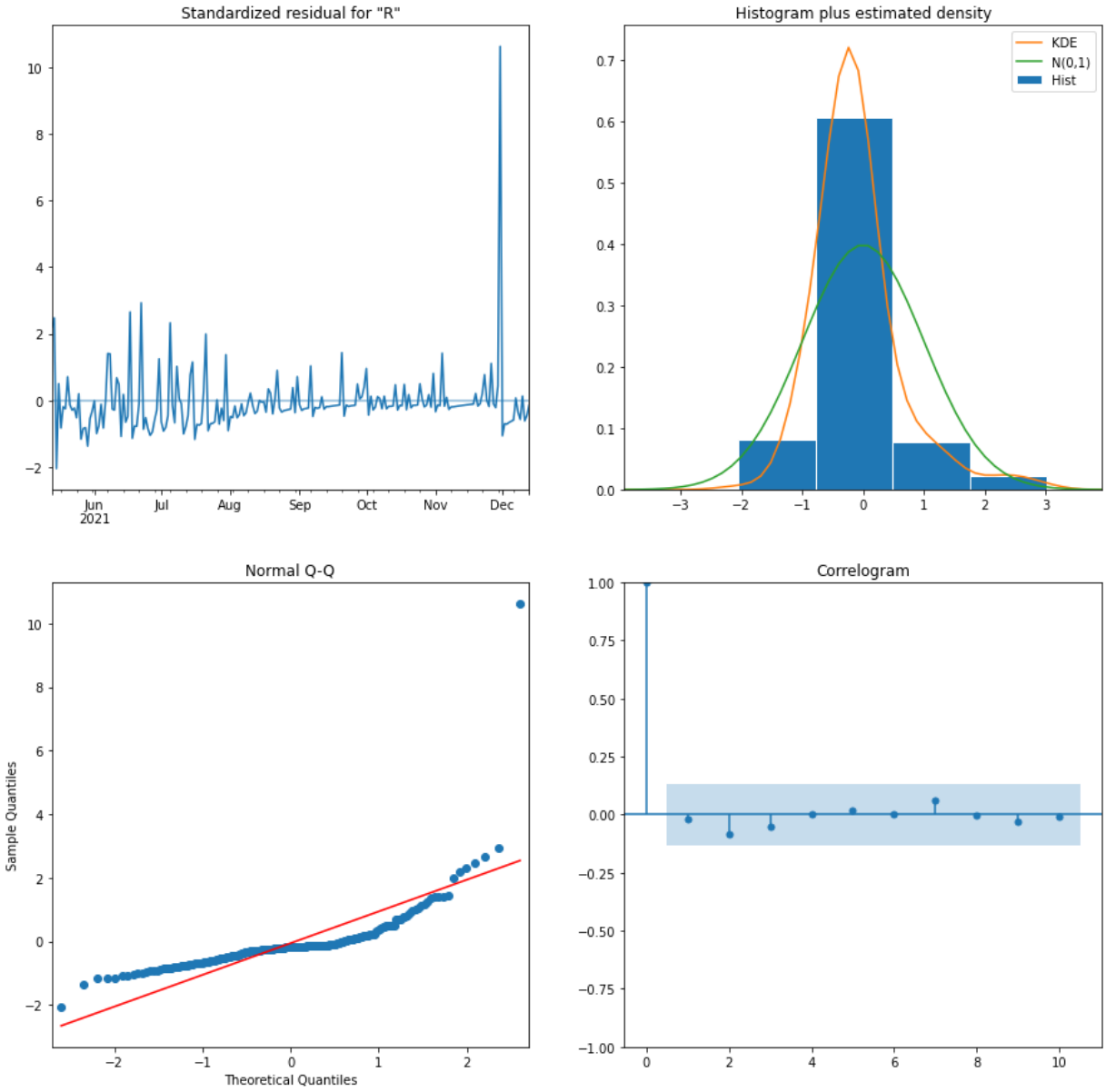
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```



12.3. Diagnostyka

In [ ]:

```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```



12.4. Prognoza

In [ ]:
```
book = {"Książka": "Rzeczpospolita trzecia i pół", "Pesymistyczny": confidence["lowe
all_books_forecast = all_books_forecast.append(book, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771
3	Koniec końca historii	0	3264.084839	36108.133593
4	Iluzja wolnej Białorusi	0	1072.104232	8157.854850
5	Polska za linią Curzona	0	1193.803081	13610.228690
6	Wdowa smoleńska	0	66.483639	4691.634338
7	Rzeczpospolita trzecia i pół	0	424.889971	3404.775235

13. "Świat narodów zagubionych" - prognozy

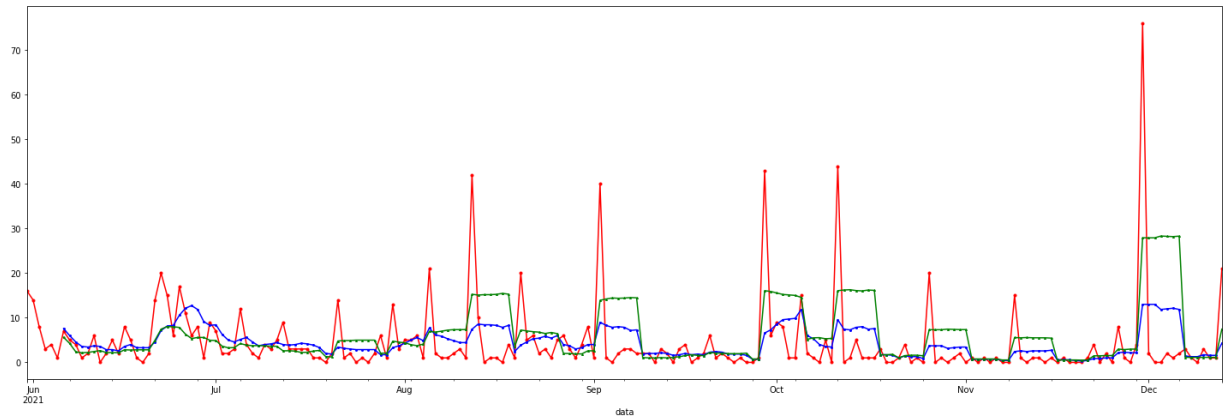
13.1. Eksploracyjna analiza danych

Stworzenie serii czasowej dot. książki

In [ ]:
```
df = sale_total["Świat narodów zagubionych"].dropna()
```

Wykres liniowy sprzedaży książki

In [ ]:
```
# Create figure and axes
fig, ax = plt.subplots(figsize=(25,8))

# Plot the daily sales
_ = df.plot(ax=ax, marker=".", linestyle="-", color="red", label = "Daily")

# Plot the rolling mean and standard deviation
df_rolling_mean = pd.Series(df).rolling(window=7).mean()
df_rolling_std = pd.Series(df).rolling(window=7).std()

# Plot the rolling mean and standard deviation
_ =  ax.plot(df_rolling_mean, marker="o", markersize=2, linestyle="-", color="blue",
_ =  ax.plot(df_rolling_std, marker="^", markersize=2, linestyle="-", color="green",
plt.show()
```



Wykres wskazuje na znaczącą różnicę między danymi do ok. połowy stycznia 2021 i później. Zmienia się średnia i odchylenie standardowe.
Czy jest sezonowy i stacjonarny? Trudno powiedzieć na podstawie samego wykresu. Zróbmy testy.

Test stacjonarności - *augmented Dickey-Fuller*

In [ ]:
```
# Import test
from statsmodels.tsa.stattools import adfuller

# Get results
adf_results = adfuller(df)

# Print values
print("WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA\n")
print(f"Wartość testu Dickeya-Fullera: {adf_results[0]}.\nIm bardziej ujemna wartość
print(f"Wartość p: {round(adf_results[1],2)}.")
# Round the critical values
rounded_criticals = {key: round(value, 2) for key, value in adf_results[4].items()}
print(f"Wartości krytyczne p: {rounded_criticals}.\n")
```

```
WYNIKI WZMOCNIONEGO TESTU DICKEYA-FULLERA

Wartość testu Dickeya-Fullera: -13.712263506564819.
Im bardziej ujemna wartość - tym większe prawdopodobieństwo, że szereg czasowy jest stacjonarny.
Wartość p: 0.0.
Wartości krytyczne p: {'1%': -3.46, '5%': -2.88, '10%': -2.57}.
```

Wygląda na to, że nasz szereg czasowy jest stacjonarny.

Sprawdźmy wszystkie wyniki testu ADF.

In [ ]:
```
adf_results
```

Out[ ]:
```
(-13.712263506564819,
 1.2313591020012048e-25,
 0,
 196,
 {'1%': -3.464161278384219,
  '5%': -2.876401960790147,
  '10%': -2.5746921001665974},
 1313.246278865732)
```

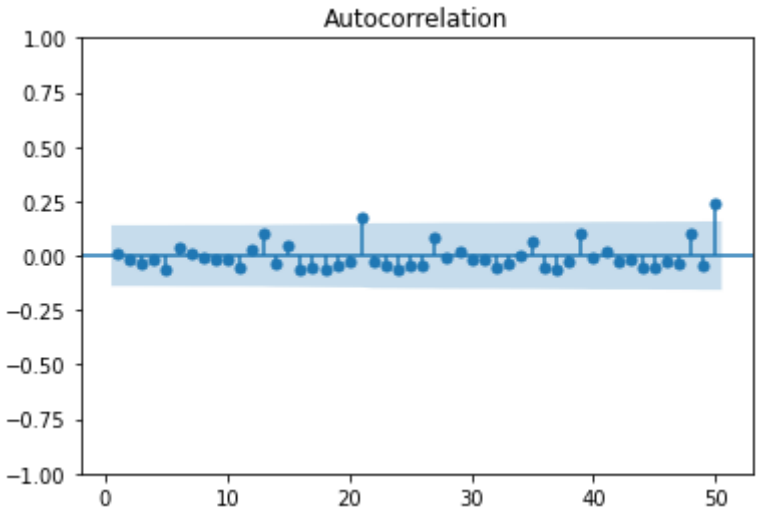
Sprawdzenie sezonowości

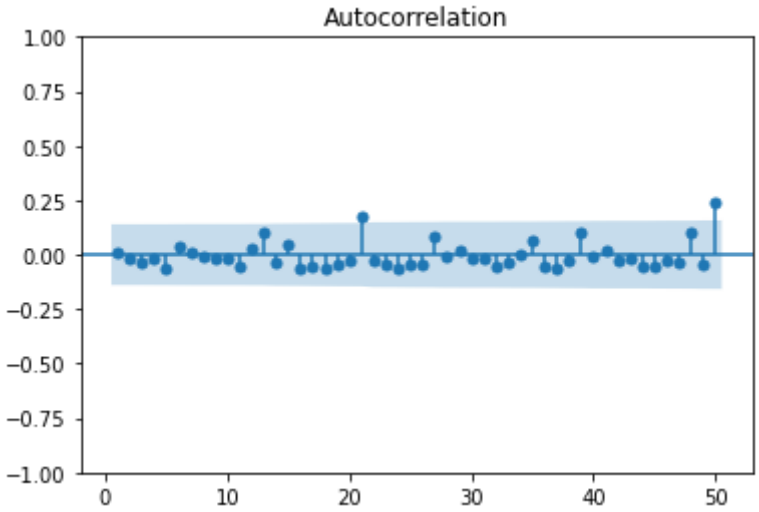
Dekompozycja sezonowa

1) Identyfikacja okresu sezonowego

In [ ]:
```
fig, ax = plt.subplots()
plot_acf(df, lags=50, zero=False, ax=ax)
```

Out[ ]:





OK, wygląda na to, że nie jest to sezonowy szereg czasowy.

13.2. Tworzenie modelu

Znajdowanie odpowiedniego rzędu dla modelu

In [ ]:

```
# Ignore warnings
warnings.filterwarnings("ignore")
# Create an empty list
order_aic_bic = []
# Loop over AR and MA order
for p in range(3):
    for q in range(3):
        # Make model and fit it, and append order and scores to the list
        try:
            model = SARIMAX(df, order=(p,0,q))
            results = model.fit()
        except ValueError:
            print(p, q, None, None)
        order_aic_bic.append((p, q, results.aic, results.bic))
# Convert the list to a DF
order_df = pd.DataFrame(order_aic_bic, columns = ["p", "q", "aic", "bic"])
# Step 5. Sort by AIC and print
print(order_df.sort_values("aic"))
```

```
   p  q          aic          bic
4  1  1  1423.395176  1433.244787
5  1  2  1425.342113  1438.474928
7  2  1  1425.343731  1438.476546
8  2  2  1427.315099  1443.731118
6  2  0  1451.758416  1461.608027
3  1  0  1454.616793  1461.183200
2  0  2  1455.184649  1465.034260
1  0  1  1456.985747  1463.552154
0  0  0  1462.444724  1465.727928
```

Tworzenie i dopasowanie modelu

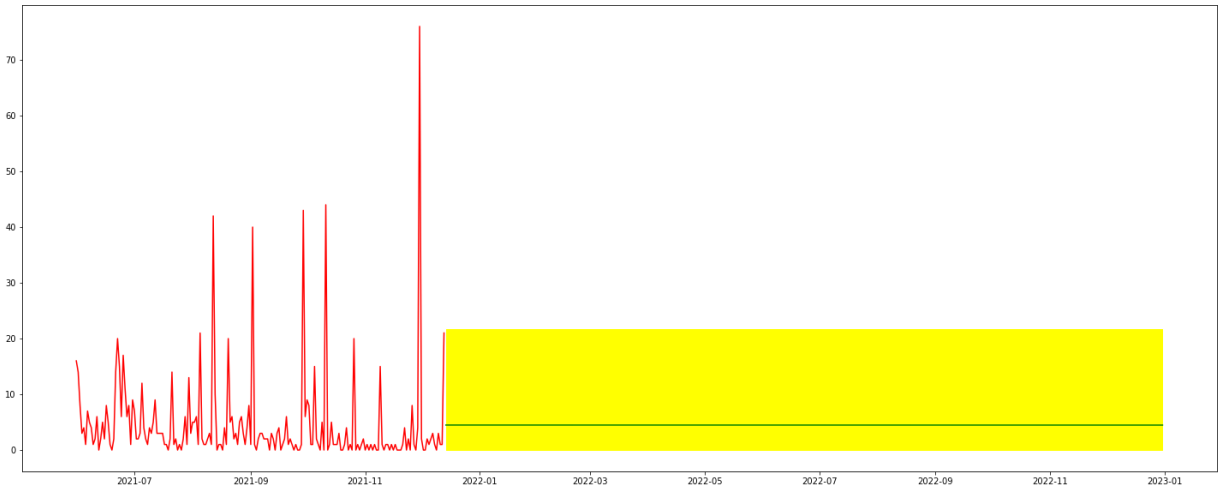
In [ ]:

```
# Instantiate a model
model = SARIMAX(df, order=(1,0,1))

# Fit the model
results=model.fit()

# Forecast for the next 365 days
```

```
forecast = results.get_forecast(steps=383).predicted_mean

confidence = results.get_forecast(steps=383).conf_int()
confidence.columns = ["lower", "upper"]
for i in range(len(confidence)):
    if confidence["lower"][i] < 0:
        confidence["lower"] = 0

forecast
```

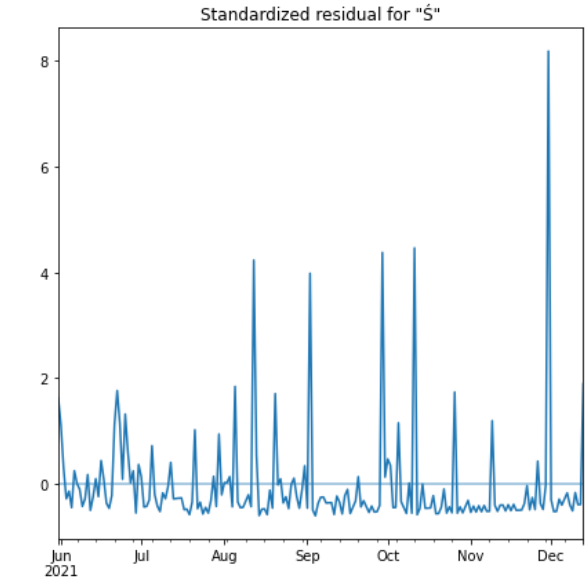
Out[ ]:
```
2021-12-14    4.488009
2021-12-15    4.487984
2021-12-16    4.487958
2021-12-17    4.487932
2021-12-18    4.487907
                ...
2022-12-27    4.478311
2022-12-28    4.478286
2022-12-29    4.478260
2022-12-30    4.478235
2022-12-31    4.478209
Freq: D, Name: predicted_mean, Length: 383, dtype: float64
```

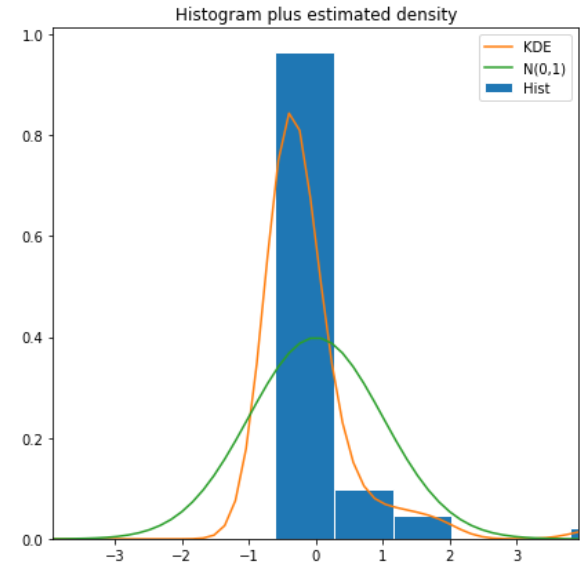
In [ ]:
```
plt.figure(figsize=(25,10))
_ = plt.plot(df, color="red", label="up-to-now")
_ = plt.plot(forecast, color="green", label="prediction")
_ = plt.fill_between(forecast.index, confidence["lower"], confidence["upper"], color
```

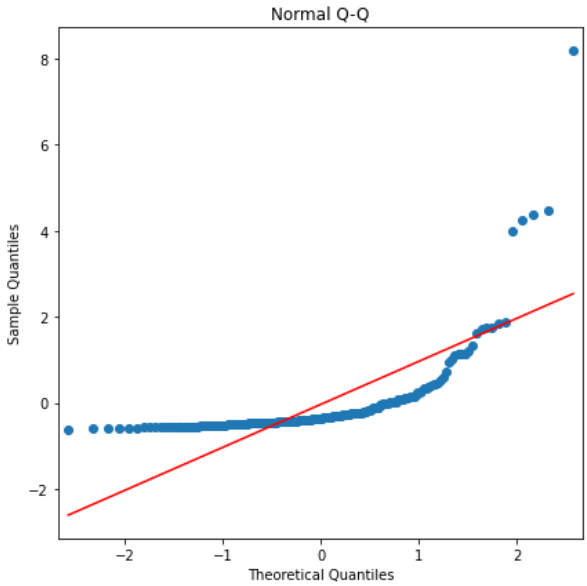


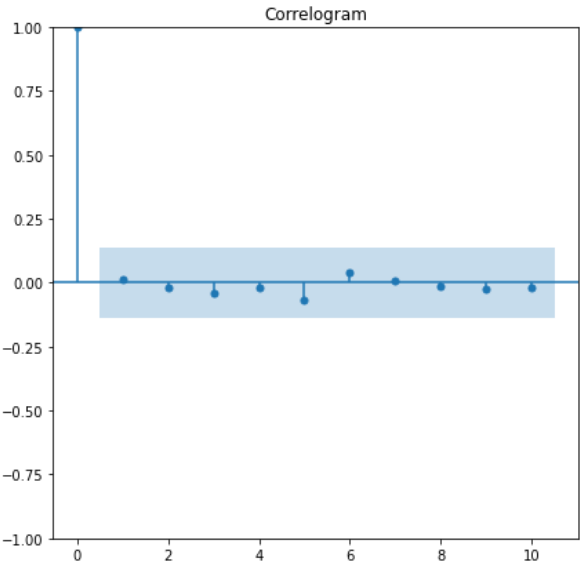
13.3. Diagnostyka

In [ ]:
```
fig=plt.figure(figsize=(15,15))
results.plot_diagnostics(fig=fig)
plt.show()
```









13.4. Prognoza

In [ ]:
```
book = {"Książka": "Świat narodów zagubionych", "Pesymistyczny": confidence["lower"]
all_books_forecast = all_books_forecast.append(book, ignore_index=True)
all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny
0	Między wielkością a zanikiem	0	500.518608	8751.793857
1	Mała degeneracja	0	300.613268	3394.053675
2	Od foliowych czapeczek...	0	304.924423	3364.772771
3	Koniec końca historii	0	3264.084839	36108.133593
4	Iluzja wolnej Białorusi	0	1072.104232	8157.854850
5	Polska za linią Curzona	0	1193.803081	13610.228690
6	Wdowa smoleńska	0	66.483639	4691.634338
7	Rzeczpospolita trzecia i pół	0	424.889971	3404.775235
8	Świat narodów zagubionych	0	1717.030119	8299.547839

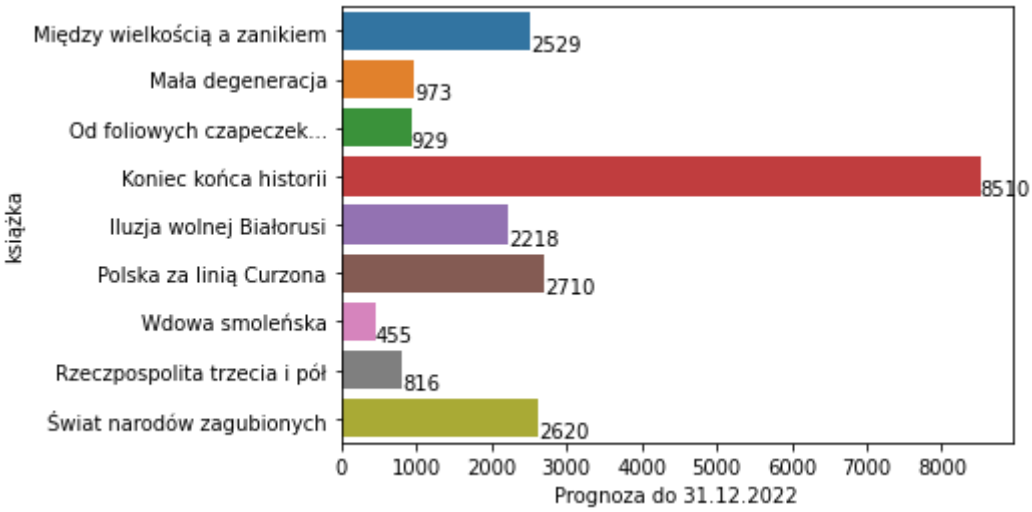
In [ ]:
```
for i in range(len(all_books_forecast)):
    all_books_forecast["Prognoza"][i] = int(all_books_forecast["Umiarkowany"][i]) +


all_books_forecast
```

Out[ ]:

	Książka	Pesymistyczny	Umiarkowany	Optymistyczny	Prognoza
0	Między wielkością a zanikiem	0	500.518608	8751.793857	2529.0
1	Mała degeneracja	0	300.613268	3394.053675	973.0
2	Od foliowych czapeczek...	0	304.924423	3364.772771	929.0
3	Koniec końca historii	0	3264.084839	36108.133593	8510.0
4	Iluzja wolnej Białorusi	0	1072.104232	8157.854850	2218.0
5	Polska za linią Curzona	0	1193.803081	13610.228690	2710.0
6	Wdowa smoleńska	0	66.483639	4691.634338	455.0
7	Rzeczpospolita trzecia i pół	0	424.889971	3404.775235	816.0
8	Świat narodów zagubionych	0	1717.030119	8299.547839	2620.0

Wykres

In [ ]:
```
def show_values_on_bars(axs, h_v="v", space=0.4):
    def _show_on_single_plot(ax):
        if h_v == "v":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() / 2
                _y = p.get_y() + p.get_height()
                value = int(p.get_height())
                ax.text(_x, _y, value, ha="center")
        elif h_v == "h":
            for p in ax.patches:
                _x = p.get_x() + p.get_width() + float(space)
                _y = p.get_y() + p.get_height()
                value = int(p.get_width())
                ax.text(_x, _y, value, ha="left")

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)

_ = sns.barplot(x="Prognoza", y="Książka", data=all_books_forecast, orient="h")
_.set(xlabel="Prognoza do 31.12.2022", ylabel="książka")
show_values_on_bars(_, "h", 0.3)
```


Między wielkością a zanikiem
Mała degeneracja
Od foliowych czapeczek...
Koniec końca historii
Iluzja wolnej Białorusi
Polska za linią Curzona
Wdowa smoleńska
Rzeczpospolita trzecia i pół
Świat narodów zagubionych
2529
973
929
8510
2218
2710
455
816
2620
książka
0
1000
2000
3000
4000
5000
6000
7000
8000
Prognoza do 31.12.2022