Analiza cen akcji wybranych spółek WIG20

Wstęp

Celem tego projektu jest analiza i prognozowanie cen akcji spółek notowanych na indeksie WIG20 na Giełdzie Papierów Wartościowych w Warszawie. WIG20 to indeks 20 największych i najbardziej płynnych spółek na GPW. Analiza tych spółek może dostarczyć cennych informacji o ogólnych trendach na polskim rynku giełdowym.

W tym projekcie skupię się na analizie historycznych danych cenowych tych spółek, które pobieram ze strony *stooq.pl*. Dane te obejmują codzienne ceny otwarcia, zamknięcia, najwyższe, najniższe oraz wolumen obrotu.

Do analizy danych i prezentacji wyników wykorzystam skrypty oraz wykresy napisane i wygenerowane przy użyciu języka programowania *Python*.

Moim celem jest dobranie odpowiednich modeli do danych, a następnie zastosowanie tych informacji do prognozowania przyszłych cen akcji. W tym celu będę korzystać z różnych technik analizy danych i modelowania predykcyjnego.

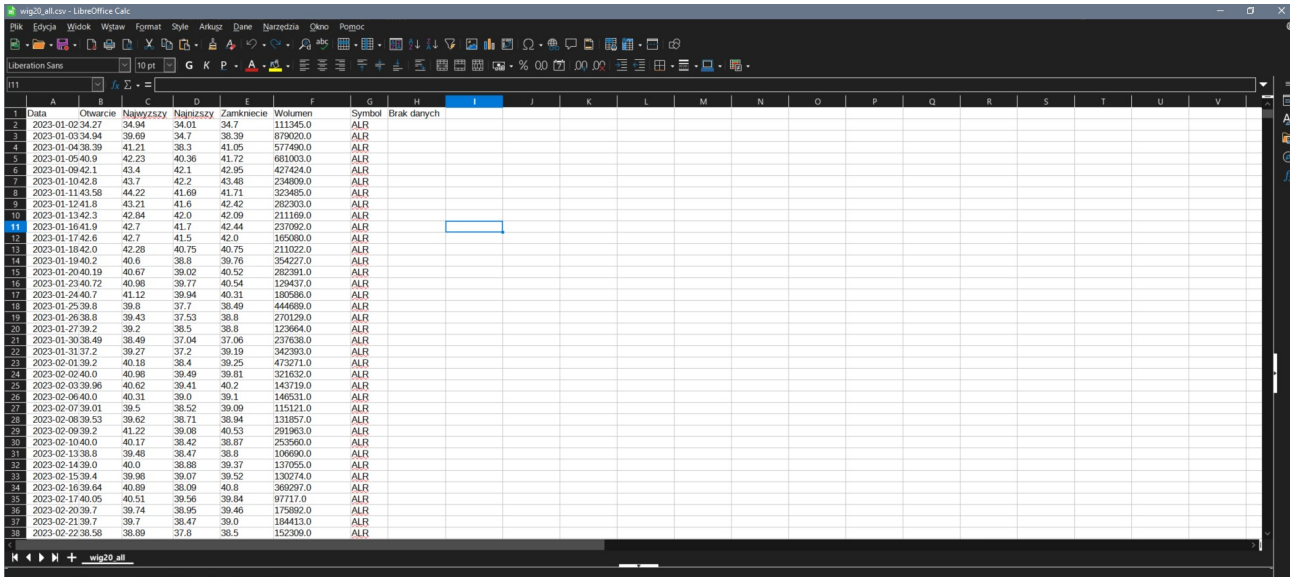
Oczywiście zdaję sobie sprawę z faktu, że na ceny akcji ma wpływ wiele czynników i nie powinno się opierać jedynie na danych historycznych, dlatego też poniższy projekt traktuję czysto hobbistycznie.

1. Pobieranie danych

Dane do analizy pobieram ze strony *stooq.pl*, która udostępnia historyczne ceny akcji spółek giełdowych. Pozyskuję dane przy użyciu skryptu *get_wig20.py*, wykorzystując bibliotekę *pandas*.

```
# get_wig20.py

import pandas as pd

def download_data(symbol, start_date, end_date):
    url = f"https://stooq.pl/q/d/l/?s={symbol}&d1={start_date}&d2={end_date}&i=d"
    data = pd.read_csv(url)
    data['Symbol'] = symbol  # Dodajemy kolumnę z symbolem spółki
    return data

# Zahardkodowana lista symboli spółek
symbols = ['ALR', 'CCC', 'CDR', 'CPS', 'DNP', 'JSW', 'KGH', 'LTS', 'LPP', 'MBK', 'OPL', 'PEO',
'PGE', 'PGN', 'PKN', 'PKO', 'PLY', 'PZU', 'SPL', 'TPE']

# Pobranie danych dla każdej spółki
all_data = []
for symbol in symbols:
    data = download_data(symbol, "20230101", "20240115")
    all_data.append(data)

# Połączenie wszystkich danych w jedną ramkę danych i zapis
all_data = pd.concat(all_data)

filename = "wig20_all.csv"
all_data.to_csv(filename, index=False)
```

Tabela 1: Pobrane dane



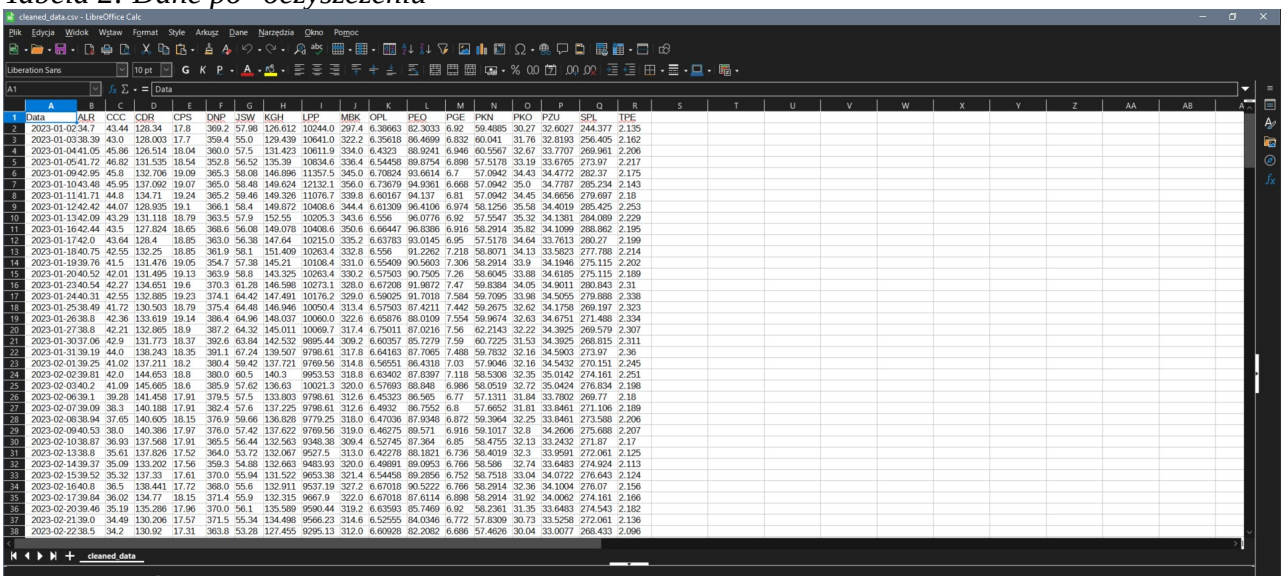
Data	Otwarcie	Najwyższy	Najniższy	Zamknięcie	Wolumen	Symbol	Brak danych
2023-01-02	34.27	34.94	34.01	34.7	111345.0	ALR	
2023-01-03	34.94	39.69	34.7	38.39	879020.0	ALR	
2023-01-04	38.39	41.21	38.3	41.05	577490.0	ALR	
2023-01-05	40.9	42.23	40.36	41.72	681003.0	ALR	
2023-01-09	42.1	43.4	42.1	42.95	427424.0	ALR	
2023-01-10	42.8	43.7	42.2	43.48	234809.0	ALR	
2023-01-11	43.58	44.22	41.69	41.71	323485.0	ALR	
2023-01-12	41.8	43.21	41.6	42.42	282303.0	ALR	
2023-01-13	42.3	42.84	42.0	42.09	211169.0	ALR	
2023-01-16	41.9	42.7	41.7	42.44	237092.0	ALR	
2023-01-17	42.6	42.7	41.5	42.0	165080.0	ALR	
2023-01-18	42.0	42.28	40.75	40.75	211022.0	ALR	
2023-01-19	40.2	40.6	38.8	39.76	354227.0	ALR	
2023-01-20	40.19	40.67	39.02	40.52	282391.0	ALR	
2023-01-23	40.72	40.98	39.77	40.54	129437.0	ALR	
2023-01-24	40.7	41.12	39.94	40.31	180586.0	ALR	
2023-01-25	39.8	39.8	37.7	38.49	444689.0	ALR	
2023-01-26	38.8	39.43	37.53	38.8	270129.0	ALR	
2023-01-27	39.2	39.2	38.5	38.8	123664.0	ALR	
2023-01-30	38.49	38.49	37.04	37.06	237638.0	ALR	
2023-01-31	37.2	39.27	37.2	39.19	342393.0	ALR	
2023-02-01	39.2	40.18	38.4	39.25	473271.0	ALR	
2023-02-02	40.0	40.98	39.49	39.81	321632.0	ALR	
2023-02-03	39.96	40.62	39.41	40.2	143719.0	ALR	
2023-02-06	40.0	40.31	39.0	39.1	146531.0	ALR	
2023-02-07	39.01	39.5	38.52	39.09	115121.0	ALR	
2023-02-08	39.53	39.62	38.71	38.94	131857.0	ALR	
2023-02-09	39.2	41.22	39.08	40.53	291963.0	ALR	
2023-02-10	40.0	40.17	38.42	38.87	253560.0	ALR	
2023-02-13	38.8	39.48	38.47	38.8	106690.0	ALR	
2023-02-14	39.0	40.0	38.88	39.37	137055.0	ALR	
2023-02-15	39.4	39.98	39.07	39.52	130274.0	ALR	
2023-02-16	39.64	40.89	38.09	40.8	369297.0	ALR	
2023-02-17	40.05	40.51	39.56	39.84	97717.0	ALR	
2023-02-20	39.7	39.74	38.95	39.46	175892.0	ALR	
2023-02-21	39.7	39.7	38.47	39.0	184413.0	ALR	
2023-02-22	38.58	38.89	37.8	38.5	152309.0	ALR	

2. Czyszczenie danych

Sprawdzam zgodność typów oraz czy w danych nie pojawiły się luki. Wśród pobranych informacji są również dane, które okazują się zbędne na potrzeby mojej prostej analizy. Przekształcam kolumny, żeby umożliwić wygodną analizę. Oczyszczone dane zapisuję w nowym pliku *cleaned_data.csv.*

```
#clean_data.py

import pandas as pd

data = pd.read_csv("wig20_all.csv")


#Sprawdzenie typów danych i braków danych

print(data.dtypes)

data['Data'] = pd.to_datetime(data['Data'])

print(data.dtypes)


print(data.isnull().sum())

#data = data.drop(columns=['Brak danych','Wolumen','Najnizszy','Otwarcie','Najwyzszy'])


#Przekształcenie symboli spółek w kolumny

data_pivot = data.pivot(index='Data', columns='Symbol', values='Zamkniecie')

data_pivot.to_csv('cleaned_data.csv')
```

Tabela 2: Dane po "oczyszczeniu"



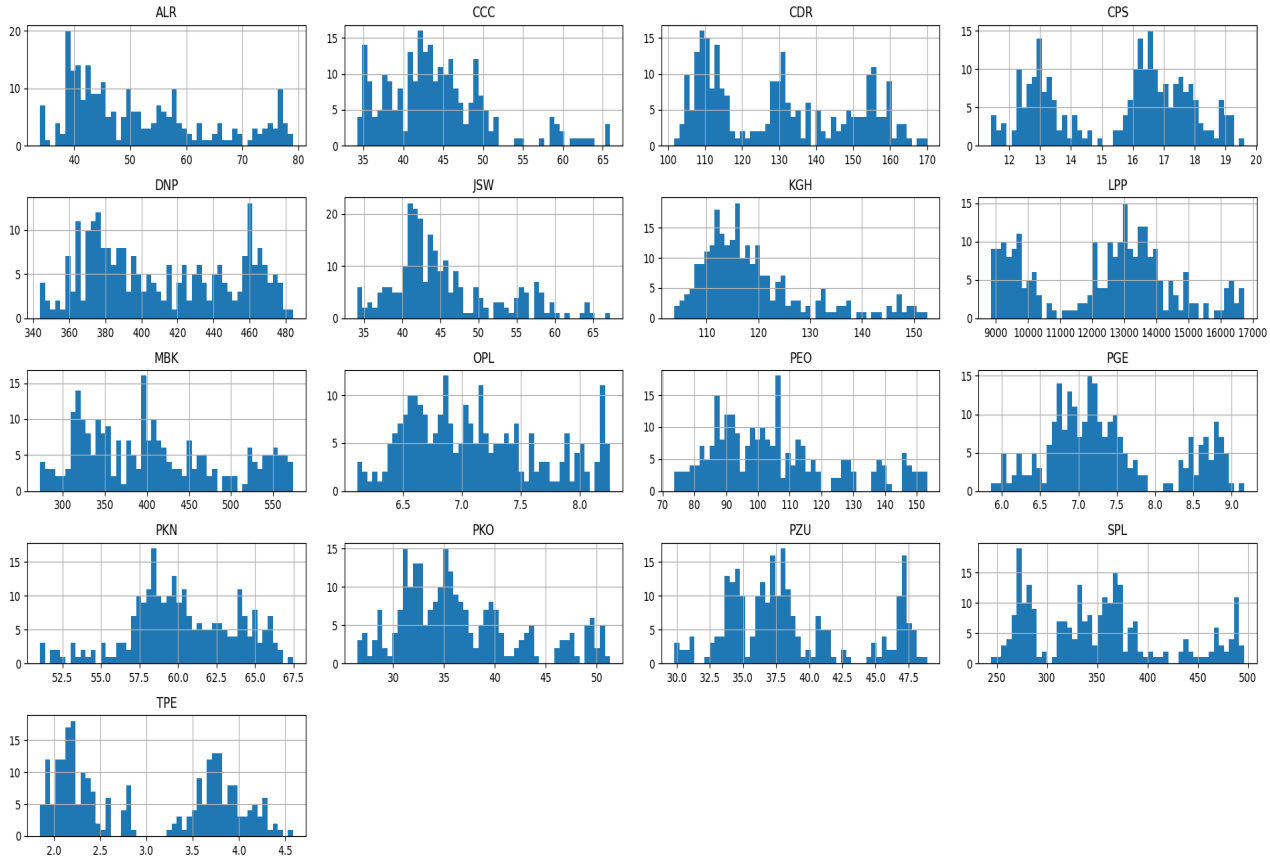
3. Wybór słabo skorelowanych spółek

Wybieram spółki słabo skorelowane, ponieważ takie podejście pozwoli mi zdywersyfikować ewentualny portfel inwestycyjny.

Stosuję do tego algorytm, który iteracyjnie dodaje spółki do zbioru, jeśli są słabo skorelowane ze wszystkimi spółkami, które już dodaliśmy. Nie gwarantuje on co prawda znalezienia maksymalnego zbioru spółek słabo skorelowanych, ale jest za to prosty i szybki.

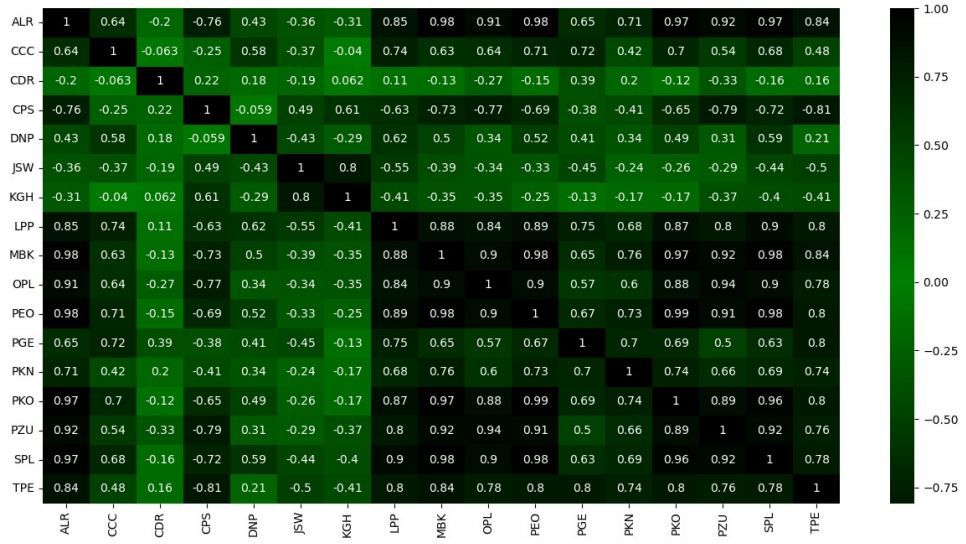
```
#eda.py


import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np


data = pd.read_csv("cleaned_data.csv")


data.hist(bins=50, figsize=(15,10))

plt.show()
```

Tabela 3: Wizualizacja rozkładu danych



```
#Tworzenie macierzy korelacji z typów numerycznych zawartych w ramce danych
numeric_data = data.select_dtypes(include=[np.number])
corr_matrix = numeric_data.corr()
threshold = 0.5

# Tworzenie i wyświetlanie mapy ciepła
sns.heatmap(corr_matrix, annot=True, cmap=cmap, center=0)
plt.show()
```

Tabela 4: Macierz korelacji




```
# Wybieranie spółek słabo skorelowanych
weak_corr = []
for i in range(len(corr_matrix.columns)):
    for j in range(i+1, len(corr_matrix.columns)):
        if abs(corr_matrix.iloc[i,j])<threshold:
            weak_corr.append((corr_matrix.columns[i],corr_matrix.columns[j]))

selected_stock = set()
for stock1, stock2 in weak_corr:
    if all(abs(corr_matrix.loc[stock, stock1]) < threshold and abs(corr_matrix.loc[stock, stock2]) <
threshold for stock in selected_stock):
        selected_stock.add(stock1)
        selected_stock.add(stock2)
# Tworzenie nowej ramki danych, zawierającej tylko wybrane kolumny i zapis nowego pliku CSV
selected_stock_df = data[["Data"]+list(selected_stock)]
selected_stock_df.to_csv("portfel.csv", index=False
```

4. Dopasowanie modelu

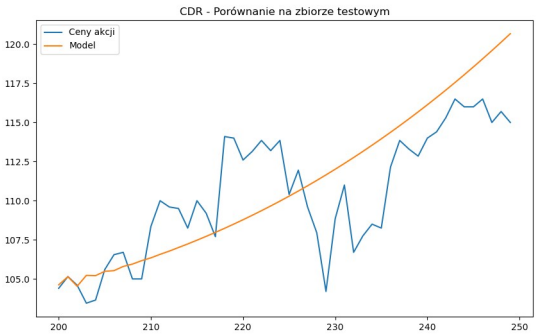
Dopasowanie modelu zaczynam od podziału danych na dwa zbiory: treningowy (80%) i testowy (20%) (z zachowaniem porządku chronologicznego, żeby móc ocenić sprawdzalność prognozy). Następnie skorzystam z modelu *ARIMA* (AutoRegressive Integrated Moving Average) z parametrami dobranymi metodą *grid search* tak, aby zminimalizować błąd średniokwadratowy.

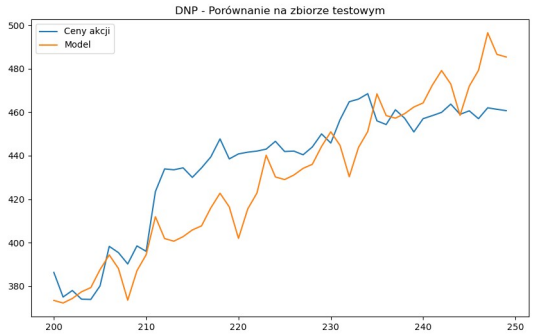
```
#model.py
from sklearn.metrics import mean_squared_error
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

import numpy as np

df = pd.read_csv("portfel.csv")

# Podział danych na zbiory treningowy i testowy
train_size = 0.8
train_num = int(df.shape[0] * train_size)
train_df = df.iloc[:train_num]
test_df = df.iloc[train_num:]

#Grid search

predictions = {}
tests = {}
min_mse={}
best_params={}
columns = train_df.columns[1:]

p_range = range(0,6)
d_range = range(0,6)
q_range = range(0,6)

for column in columns:
    y_train = train_df[column]
    y_test = test_df[column]

    best_params[column] = None
    lowest_mse[column] = np.inf

    for p in p_range:
        for d in d_range:
            for q in q_range:
                try:
                    model = ARIMA(y_train, order=(p,d,q))
                    model_fit = model.fit()

                    y_pred = model_fit.predict(start=len(train_df),end=len(train_df)+len(test_df)-1)

                    mse = mean_squared_error(y_test, y_pred)

                    if mse < lowest_mse[column]:
                        best_params[column] = (p, d, q)
                        lowest_mse[column] = mse
                except:
                    continue
for col in columns:
    print(f'Best params for {col}: {best_params[col]}, MSE: {lowest_mse[col]}')
```

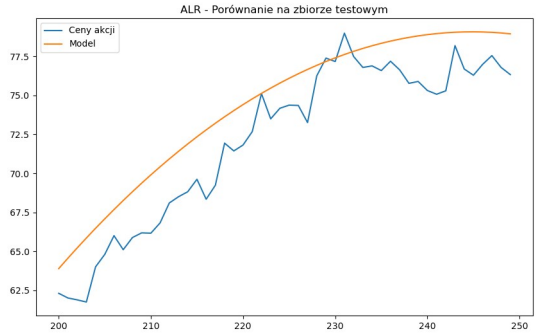
```
# Best params for JSW: (5,1,0),   MSE: 11.843689481087306 --OK
# Best params for ALR: (0, 3, 1), MSE: 6.557770249093277  --OK
# Best params for CDR: (3, 4, 2), MSE: 10.408574870811051 --OK
# Best params for DNP: (4, 3, 4), MSE: 1677.697466214512
```

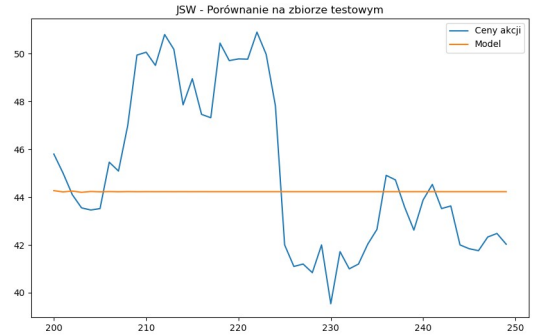
Błąd średniokwadratowy dla spółki DNP jest zbyt wysoki w stosunku do wariancji, więc spróbowałem użyć modelu *SARIMAX*, który pozwolił obniżyć *MSE*, ale nie sądzę, żeby ceny akcji DNP wykazywały się miesięczną sezonowością, więc myślę że nie warto brać ich prognoz pod uwagę podczas dalszej analizy.

```
#Sprawdzanie wariancję, żeby ocenić MSE
variance={}
for column in train_df.columns[1:]:
    variance[column]= train_df[column].var()
    print(f'Wariancja dla kolumny {column}: {variance[column]}, MSE: {min_mse[column]}, BEST PARAMS:
{best_params[column]}')

# # Wariancja dla kolumny ALR: 51.0976731155779, MSE: 6.557770249093277, BEST PARAMS: (0, 3, 1)
# # Wariancja dla kolumny JSW: 56.38201414824117, MSE: 11.843689481087306, BEST PARAMS: (5, 1, 0)
# # Wariancja dla kolumny CDR: 341.07862969846747, MSE: 10.408574870811051, BEST PARAMS: (3, 4, 2)
# # Wariancja dla kolumny DNP: 1510.9146530150758, MSE: 329.1053547512303, BEST PARAMS: (1, 0, 0, 1,
0, 0, 12)
```





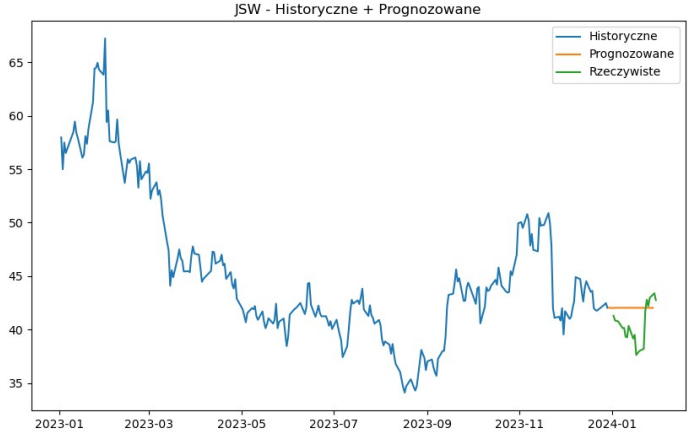


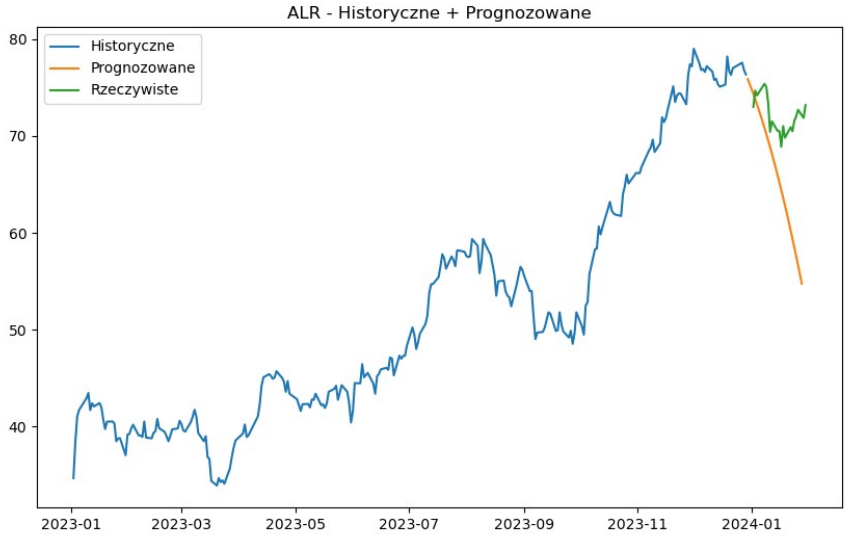


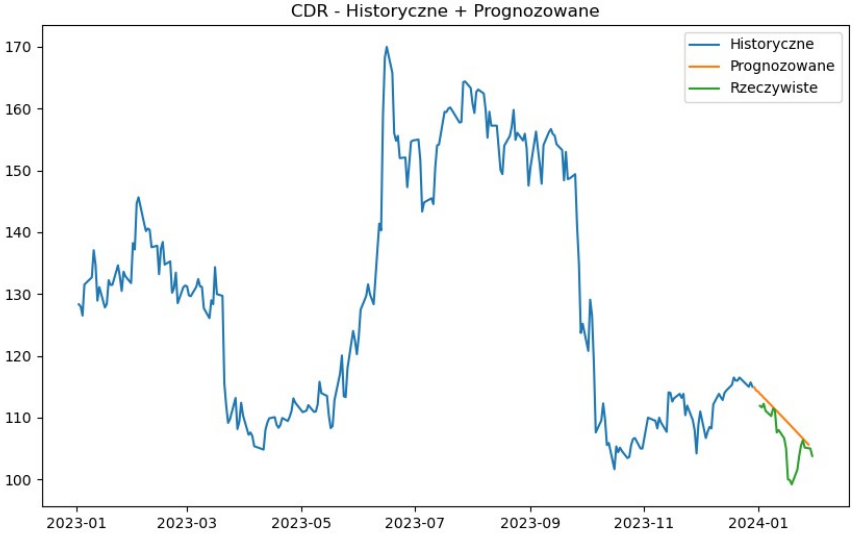
5. Prognoza

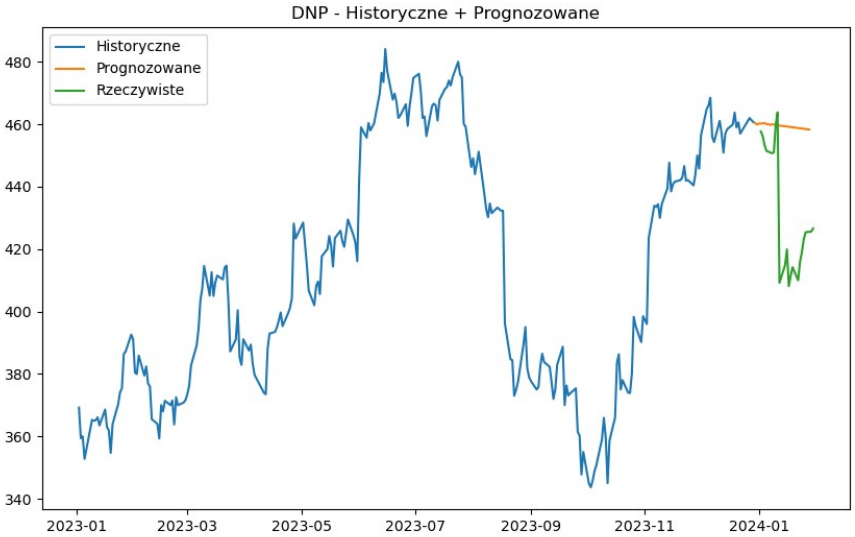
Na podstawie wygenerowanych modelów tworzę prognozy na okres 01.01.2024 – 31.01.2024 i porównuję je z rzeczywistymi cenami akcji.

```
#prognoza.py

import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

df = pd.read_csv("portfel.csv", index_col='Data', parse_dates=True)

# Definiowanie parametrów modelu
best_params = {
    'ALR': (0, 3, 1),
    'JSW': (5, 1, 0),
    'CDR': (3, 4, 2),
    'DNP': (1, 0, 0, 1, 0, 0, 12)
}

# Przygotowanie słownika do przechowywania prognoz
predictions = {}

from pandas.tseries.offsets import DateOffset

# Trenowanie modelu i generowanie prognozy dla każdej kolumny
for column in df.columns:
    if column == 'DNP':
        model = SARIMAX(df[column], order=(1, 0, 0), seasonal_order=(1, 0, 0, 12))
    else:
        model = ARIMA(df[column], order=best_params[column])
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=30)  # prognoza na następne 30 dni
    predictions[column] = forecast

future_data=pd.read_csv('cleaned_future_stocks.csv', index_col='Data', parse_dates=True)

# Wyświetlenie wykresów prognoz
for column in predictions.keys():
    plt.figure(figsize=(10,6))
    plt.plot(df[column], label='Historyczne')
    plt.plot(predictions[column], label='Prognozowane')
    plt.plot(future_data[column], label='Rzeczywiste')
    plt.title(f'{column} - Historyczne + Prognozowane')
    plt.legend()
    plt.show()
```




ALR - Historyczne + Prognozowane
Historyczne
Prognozowane
Rzeczywiste
80
70
60
50
40
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


CDR - Historyczne + Prognozowane
Historyczne
Prognozowane
Rzeczywiste
170
160
150
140
130
120
110
100
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01


DNP - Historyczne + Prognozowane
Historyczne
Prognozowane
Rzeczywiste
480
460
440
420
400
380
360
340
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
2024-01

6. Interpretacja i wnioski

Na podstawie wykresów można zauważyć, że prognozy generowane przez modele ARIMA i SARIMAX nie odzwierciedlają w pełni dynamiki obserwowanej w danych empirycznych. W szczególności, prognozy nie uwzględniają znaczących fluktuacji cen, które są typowe dla rynków finansowych.

Ponadto, prognozy nie odzwierciedlają trendów obserwowanych w danych empirycznych. Na przykład w przypadku JSW, w danych empirycznych widzimy, że cena spada od 41.29 do 37.62, a następnie wzrasta do 43.40. Tymczasem prognozy utrzymują się na stałym poziomie około 42.03, nie odzwierciedlając tych trendów.

Jednym z możliwych wyjaśnień takiego zachowania modeli ARIMA i SARIMAX może być fakt, że są one modelami liniowymi i mogą mieć trudności z modelowaniem nieliniowych, długoterminowych zależności obserwowanych w danych finansowych.

W związku z powyższym, prognozy generowane przez te modele mogą nie być użyteczne do celów inwestycyjnych. Wysokie ryzyko błędu prognozy, jak wskazuje brak zgodności prognoz z obserwowanymi trendami i fluktuacjami, może prowadzić do błędnych decyzji inwestycyjnych i potencjalnych strat.

Podsumowanie

Rynki finansowe są złożone i ma na nie wpływ wiele różnych czynników, takich jak zmiany w gospodarce, polityce, technologii, czy też nastroje inwestorów. Te czynniki są trudne do przewidzenia i modelowania, co sprawia, że prognozy oparte na samych danych historycznych są często niedokładne.

Dlatego, choć te modele mogą być użyteczne jako część strategii inwestycyjnej, zawsze powinny być używane z ostrożnością i w połączeniu z innymi narzędziami i informacjami. Zawsze istnieje ryzyko, że rzeczywiste wyniki będą różnić się od prognoz.