



POLITECHNIKA RZESZOWSKA

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

Eryk Jabłoński, Jakub Jędrzejczyk

nr. Index 173148, 173150

Kierunek
Inżynieria i Analiza Danych

Administracja Systemów Rozproszonych
Praca Projektowa: „Prognoza pogody w Rzeszowie”

Rzeszów, 2024

Spis treści

Zmiana tematu projektu.....	4
Dane meteorologiczne	4
Załadowanie danych.....	4
Wizualizacja oryginalnych danych.....	5
Dzielenie na części train/test o różnych proporcjach	5
Ogólny wstęp techniczny.....	6
Model AR.....	7
Wstęp techniczny dla modelu AR.....	7
Wybieranie parametrów modelu AR.....	7
Wyniki Modelu AR.....	8
Interpretacja wyników	10
Model MA.....	11
Wstęp techniczny dla modelu MA	11
Wybieranie parametrów modelu MA	11
Wyniki Modelu MA.....	12
Interpretacja wyników	14
Model ARIMA	15
Wstęp techniczny do modelu ARIMA.....	15
Proces dobieranie parametrów dla modelu ARIMA	15
Wyniki Modelu ARIMA	16
Interpretacja wyników	18
Model SARIMA	19
Próba użycie modelu	19
Model Facebook Prophet.....	20
Kod potrzebny do działania modelu.....	20
Wyniki Modelu Facebook Prophet.....	21
Interpretacja wyników	23
Używanie Modeli AR, MA i ARIMA na trendzie naszych danych.....	23
Wstęp techniczny	23
Modyfikowanie danych regresja liniowa.....	24
Modyfikowanie danych seasonal decompose	24
Informacje o różnicy otrzymanych trendach.....	25
Kody modeli.....	25
Regresja linearana	26
Wyniki model AR	26

Wyniki model MA.....	28
Wyniki model ARIMA.....	30
Interpretacja wyników	31
Seasonal decompose	32
Wyniki model AR	32
Wyniki model MA.....	34
Wyniki model ARIMA.....	36
Interpretacja wyników	37

Zmiana tematu projektu

Zdecydowaliśmy, że większą wartość edukacyjną oraz bardziej praktyczne umiejętności zyskam, koncentrując się na prostszych, ale również efektywnych technikach analizy danych i modelowania statystycznego. Przejście na projekt, który obejmuje analizę i prognozowanie serii czasowych przy użyciu różnorodnych podejść, takich jak AR, MA, ARIMA, SARIMA, a także metody oparte na uczeniu maszynowym, pozwoli na głębsze zrozumienie tych technik i ich praktyczne zastosowanie w rzeczywistych scenariuszach danych.

Dane meteorologiczne

Nasze dane zostały pobrane ze strony: <https://power.larc.nasa.gov/data-access-viewer/>

Zawierają one temperaturę zarejestrowaną każdego dnia na wysokości 2 metrów nad ziemią w Rzeszowie z lat 1982-2024.

Załadowanie danych

Łączymy trzy kolumny by stworzyć kolumnę z datą, którą potem ustawiamy jako indeks. Możemy zauważać, że indeksowanie powiodło się gdyż częstotliwość jest ustalona na dzienną.

```
In [3]: # Tworzenie kolumny 'DATE' ręcznie przez sformatowanie ciągu znaków i konwersję na date. Ta metoda nie wyrzuca błędu
# Tworzymy nową kolumnę z trzech już istniejących kolumn, tak mamy kolumnę która posiada date
data_xxx['DATE'] = (pd.to_datetime(data_xxx['YEAR'].astype(str) + '-' + data_xxx['MO'].astype(str) + '-' + data_xxx['DY'].astype(str))).dt.date

In [4]: # Ustawienie 'DATE' jako indeksu DataFrame
data_xxx.set_index('DATE', inplace=True)

In [5]: # Usuwamy nie potrzebne już kolumny
data_xxx=data_xxx.drop(columns=['YEAR', 'MO', 'DY'])

In [6]: # Usunięcie duplikatów, zachowanie pierwszego wystąpienia każdej daty
data_xxx = data_xxx[~data_xxx.index.duplicated(keep='first')]

In [7]: # Ustawienie codziennej częstotliwości, aby zagwarantować kompletność danych
data_xxx = data_xxx.asfreq('D')
print(data_xxx.index.freq)
```

<Day>

Usuwamy i interpolujemy wartości anomalne. Jak widać procedura nie naruszyła naszych danych.

```
In [8]: # Identyfikacja i zastąpienie anomalii wartością NaN
data_xxx.loc[data_xxx['T2M'] < -100, 'T2M'] = None

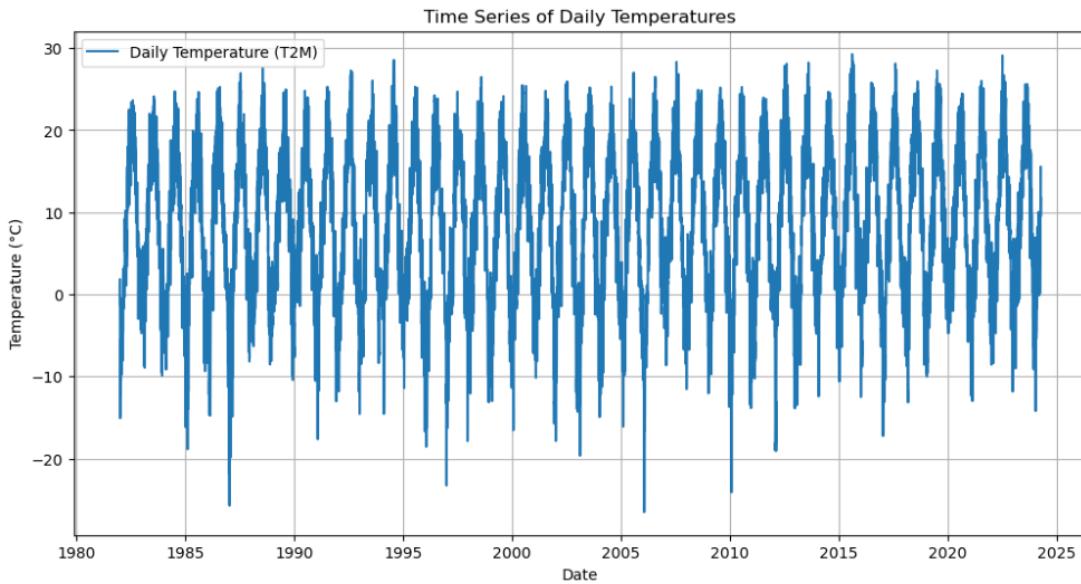
In [9]: # Wypełnienie brakujących wartości metodą 'backfill'
data_xxx['T2M'].fillna(method='bfill', inplace=True)

In [10]: # Wyświetlanie danych
print(data_xxx.head())
```

	T2M
DATE	
1982-01-01	1.23
1982-01-02	-1.28
1982-01-03	-1.10
1982-01-04	0.94
1982-01-05	1.87

Wizualizacja oryginalnych danych

```
In [11]: # Rysowanie wykresu serii czasowej temperatury
plt.figure(figsize=(12, 6))
plt.plot(data_xxx['T2M'], label='Daily Temperature (T2M)')
plt.title('Time Series of Daily Temperatures')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)
plt.show()
```



Dzielenie na części train/test o różnych proporcjach

Tworzymy funkcje do dzielenia danych i wybieramy nasze proporcje.

```
In [12]: # Funkcja dzieląca dane na zestawy treningowe i testowe
def split_data(data, train_ratio):
    train_size = int(len(data) * train_ratio)
    train_set = data.iloc[:train_size]
    test_set = data.iloc[train_size:]
    return train_set, test_set
```

```
In [13]: # Podziąły na zestawy
splits = [0.35, 0.50, 0.65, 0.80]
datasets = {}
```

```
In [14]: # Tworzenie zestawów train/test
for split in splits:
    train, test = split_data(data_xxx, split)
    datasets[f'train_{int(split*100)}'] = train
    datasets[f'test_{int(split*100)}'] = test
```

Następnie upewniamy się że nasze dane są poprawnie oddzielone logicznie i jeszcze raz wypełniamy wartości Nan jeśli takie powstały.

```
In [16]: # Aby uniknąć SettingWithCopyWarning, upewnij się, że każdy z datasetów jest niezależnym DataFrame'em.
for key in datasets.keys():
    datasets[key] = datasets[key].copy()

# Następnie możesz bezpieczne zastosować .loc do wypełnienia wartości NaN
for key in ['train_35', 'test_35', 'train_50', 'test_50', 'train_65', 'test_65', 'train_80', 'test_80']:
    datasets[key].loc[:, 'T2M'] = datasets[key]['T2M'].fillna(method='ffill')
```

Ogólny wstęp techniczny

Do predykcji temperatury w Rzeszowie będziemy używać różnym modeli szeregów czasowych. Działają one poprzez przewidywanie przyszłych wartości na podstawie wartości wcześniejszych. Każdy model będzie dokładniej opisywany w jego części sprawozdania, jednak dobrze jest wiedzieć przed przeczytaniem dalszej części jak właściwości modeli będą zapisywane.

Większość modeli użytych w tym projekcie jest w stanie być opisana przy pomocy podanego wektora: $(p,d,q)(P,D,Q)(s)$

Gdzie:

Części ARIMA:

p - liczba opóźnień użytych w modelu autoregresyjnym (AR)

d - stopień różnicowania stosowany do uzyskania stacjonarności serii (I)

q - liczba opóźnień błędu prognozy w modelu średniej ruchomej (MA)

Części SARIMA:

P - liczba sezonowych opóźnień w modelu AR

D - stopień sezonowego różnicowania

Q - liczba sezonowych opóźnień w modelu MA

s - długość sezonu (np. 12 dla danych miesięcznych z rocznym cyklem)

Wartości tego wektora mogą wynosić od 0 do nieskończoności, 0 oznacza, że dana część modelu nie występuje.

Przykłady:

$(3,0,0)(0,0,0)(0)$ - Model AR z liczbą opóźnień użytych w modelu równą trzy

$(0,0,2)(0,0,0)(0)$ - Model MA z liczbą opóźnień użytych w modelu równą trzy

$(3,0,2)(0,0,0)(0)$ - Model ARMA z liczbą opóźnień AR równą trzy i liczbą opóźnień MA równą dwa

$(3,1,3)(1,0,1)(12)$ - Model SARIMA z liczbą opóźnień AR równą trzy, stopniem różnicowanie równym jeden, liczbą opóźnień MA równą dwa dla modelu standardowego i liczbą opóźnień AR równą jeden, liczbą opóźnień MA równą jeden i długością sezonu równą dwanaście dla modelu sezonowego

Model AR

Wstęp techniczny dla modelu AR

Model autoregresyjny (AR) w szeregu czasowym to jeden z głównych narzędzi stosowanych w analizie i prognozowaniu danych czasowych. Model ten opisuje wartość zmiennej w danym czasie jako funkcję jej własnych poprzednich wartości.

Równanie modelu autoregresyjnego $\text{AR}(p)$, gdzie p to rząd modelu (czyli liczba uwzględnianych opóźnień), można zapisać jako:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \dots + \phi_p X_{t-p} + \varepsilon_t$$

gdzie:

- X_t to wartość szeregu w czasie t ,
- c to stała (intercept),
- $\phi_1, \phi_2, \dots, \phi_p$ to parametry modelu odpowiadające odpowiednio za wpływ wartości z $1, 2, \dots, p$ okresów wcześniej,
- ε_t to błąd losowy w czasie t , który jest zazwyczaj założony jako biały szum (tj. szereg stochastyczny o zerowej średniej i stałej wariancji, niezależny dla różnych t).

Posiada on tylko jeden parametr: $(p, 0, 0)(0, 0, 0)(0)$

Czyli liczbę przeszłych wartości na podstawie jakich mamy wykonać predykcje, liczbę to nazywamy lagami.

Wybieranie parametrów modelu AR

Nasza funkcja to pętla for, która tworzy model z odpowiednią ilością lagów. Następnie przeprowadza test AIC. Model z najniższym, czyli najlepszym, wynikiem zostaje zapamiętany.

```
In [17]: # Funkcja do znajdowania najlepszego modelu AR dla danego zestawu danych
def best_ar_model(data):
    best_aic = np.inf
    best_order = None
    best_model = None

    # Testujemy różne rzędy modelu AR
    for i in range(1, 5):  # Możesz zmienić zakres rzędów zgodnie z potrzebami
        try:
            model = AutoReg(data, lags=i)
            results = model.fit()
            if results.aic < best_aic:
                best_aic = results.aic
                best_order = i
                best_model = results
        except Exception as e:
            print(f"Blad podczas dopasowywania modelu AR({i}): {e}")
            continue

    return best_order, best_aic, best_model
```

Poniższa funkcja tworzy nam model o wybranym najlepszym parametrze i wizualizuje ją.

```
In [18]: # Funkcja do tworzenia modeli AR, prognozowania i wizualizacji
def plot_predictions(train_data, test_data, best_order):
    model = AutoReg(train_data, lags=best_order)
    model_fitted = model.fit()
    predictions = model_fitted.predict(start=len(train_data), end=len(train_data) + len(test_data) - 1, dynamic=False)

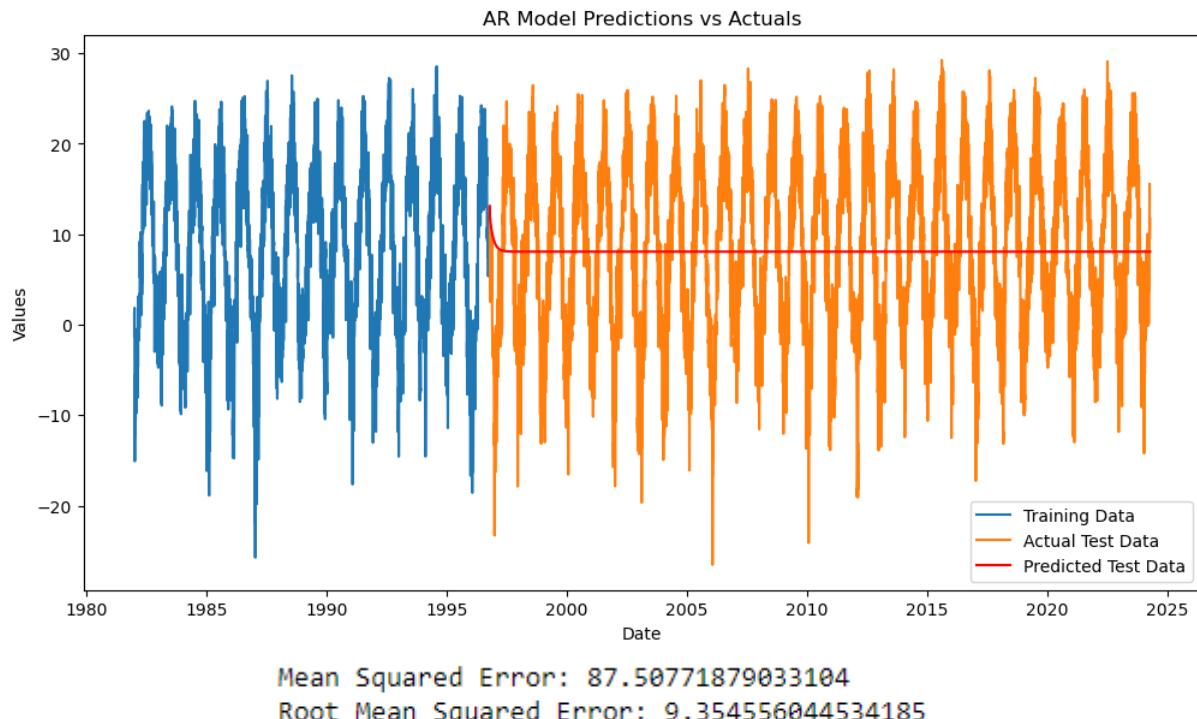
    plt.figure(figsize=(12, 6))
    plt.plot(train_data.index, train_data, label='Training Data')
    plt.plot(test_data.index, test_data, label='Actual Test Data')
    plt.plot(test_data.index, predictions, label='Predicted Test Data', color='red')
    plt.title('AR Model Predictions vs Actuals')
    plt.xlabel('Date')
    plt.ylabel('Values')
    plt.legend()
    plt.show()

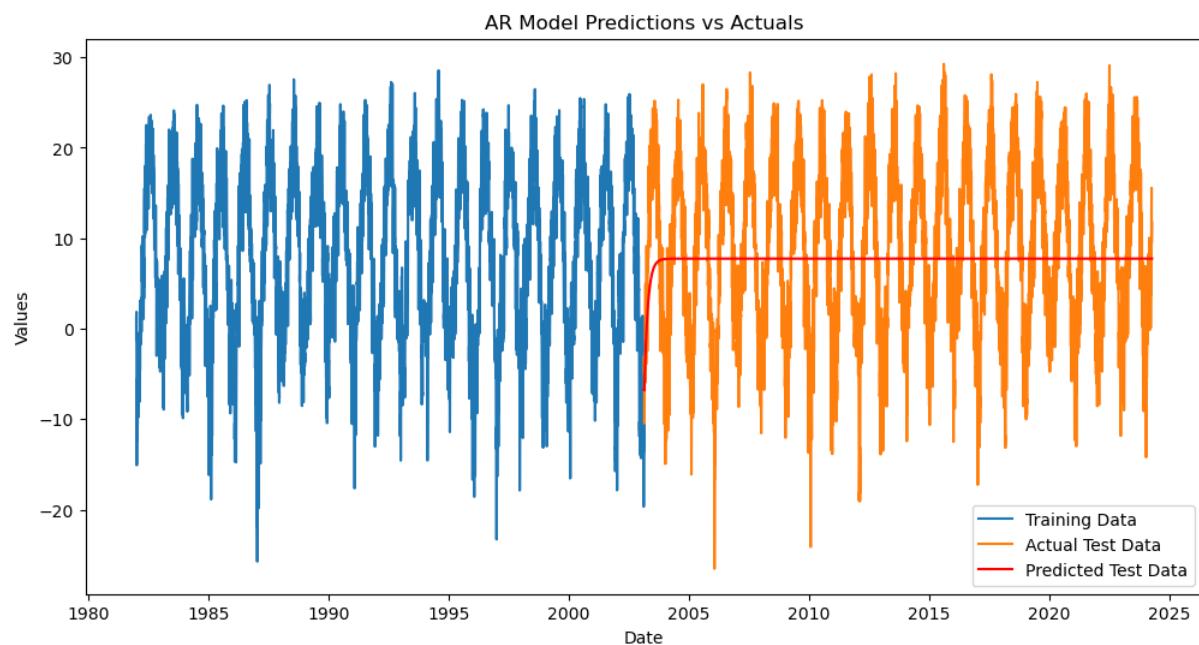
    return predictions
```

Sprawdzamy dokładność naszego modelu dzięki obliczaniu średniego błędu i pierwiastka średniego błędu jaki wygenerował model.

```
In [19]: # Funkcja sprawdzająca dokładność modelu
def evaluate_forecast(actuals, predictions):
    mse = mean_squared_error(actuals, predictions)
    rmse = sqrt(mse)
    print(f"Mean Squared Error: {mse}")
    print(f"Root Mean Squared Error: {rmse}")
```

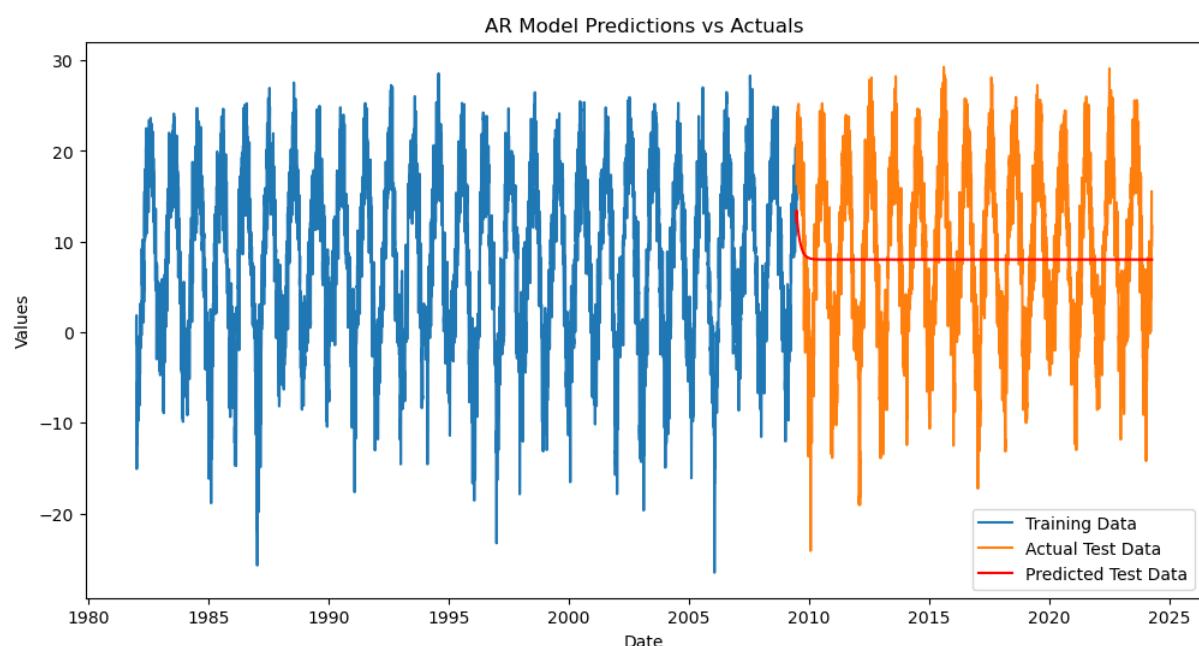
Wyniki Modelu AR





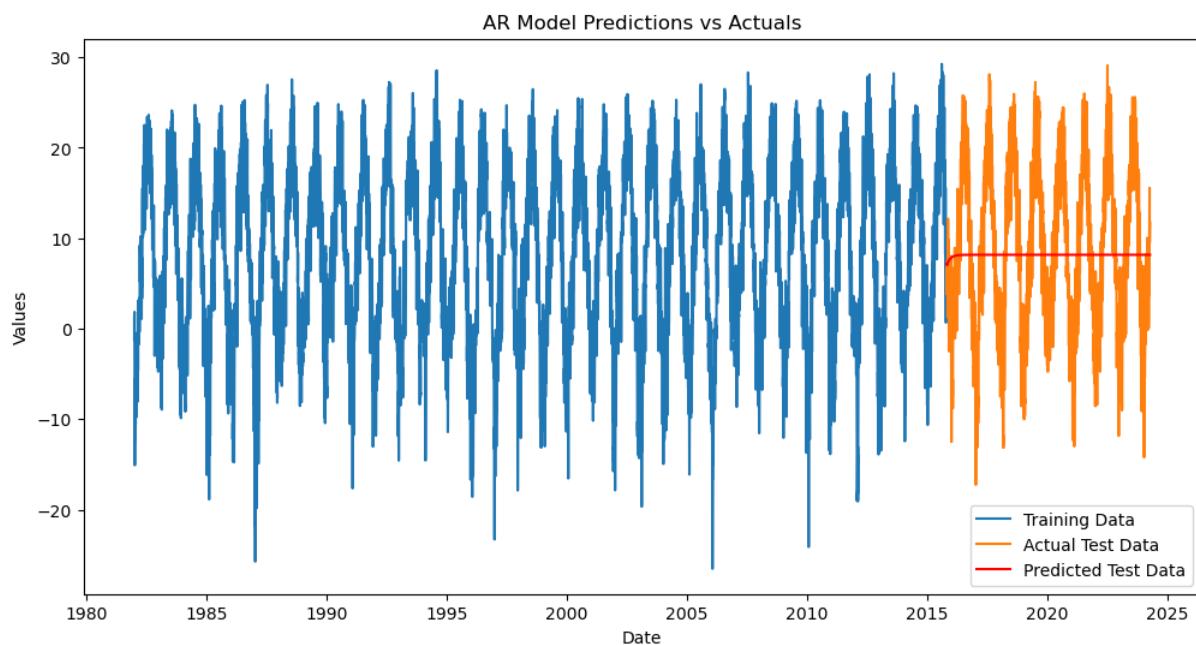
Mean Squared Error: 86.76824473713931

Root Mean Squared Error: 9.314947382413886



Mean Squared Error: 86.37048842746907

Root Mean Squared Error: 9.293572425470686



Mean Squared Error: 81.39212057363432

Root Mean Squared Error: 9.021758175302324

Interpretacja wyników

Niestety model AR jest zbyt prosty by poradzić sobie ze złożonością naszych danych. Proporcje zestawów train/test nie mają większego znaczenia w dokładności wyników. Możemy domyślać się, że problemem jest stacjonarność (średnia i wariancja nie zmieniają się w czasie) naszych danych i sezonowość. Model AR jest dobry dla danych posiadających silny trend, który w naszych danych nie występuje.

Model MA

Wstęp techniczny do modelu MA

Model średniej ruchomej (MA) w szeregu czasowym jest kolejnym kluczowym narzędziem w analizie i prognozowaniu danych czasowych. W odróżnieniu od modelu autoregresyjnego, model MA opisuje wartość zmiennej jako funkcję bieżących i poprzednich błędów prognozy, co pozwala na modelowanie szeregów czasowych z "szumami" lub fluktuacjami.

Równanie modelu średniej ruchomej $MA(q)$, gdzie q oznacza rząd modelu (czyli liczbę uwzględnianych opóźnień błędu), można przedstawić w następujący sposób:

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q}$$

gdzie:

- X_t to wartość szeregu w czasie t ,
- μ to średnia szeregu,
- $\varepsilon_t, \varepsilon_{t-1}, \dots, \varepsilon_{t-q}$ to błędy losowe w czasie $t, t-1, \dots, t-q$ odpowiednio,
- $\theta_1, \theta_2, \dots, \theta_q$ to parametry modelu odpowiadające odpowiednio za wpływ błędów z $1, 2, \dots, q$ okresów wcześniej.

Podobnie jak w poprzednim przypadku model posiada tylko jeden parametr: (0,0,q)(0,0,0)(0)

Wybieranie parametrów modelu MA

Proces wybierania parametrów jest bardzo podobny, jedyną zmianą jest to że iterujemy po ilości lagów (0,0,q).

```
In [32]: # Funkcja do wybierania najlepszego modelu
def evaluate_ma_model(data, max_lag):
    best_aic = np.inf
    best_order = None
    best_model = None

    for q in range(1, max_lag+1):
        try:
            # Specyfikacja modelu MA(q) z dodatkowymi ustawieniami początkowymi
            model = ARIMA(data, order=(0, 0, q), enforce_stationarity=False, enforce_invertibility=True)
            results = model.fit(method='innovations_mle', low_memory=True) # Metoda innovations MLE może być bardziej stabilna
            if results.aic < best_aic and np.all(np.abs(results.maparams) < 1): # Dodatkowa weryfikacja odwrocalności
                best_aic = results.aic
                best_order = q
                best_model = results
        except Exception as e:
            print(f"Failed to fit MA({q}) model: {e}")
            continue

    return best_order, best_aic, best_model
```

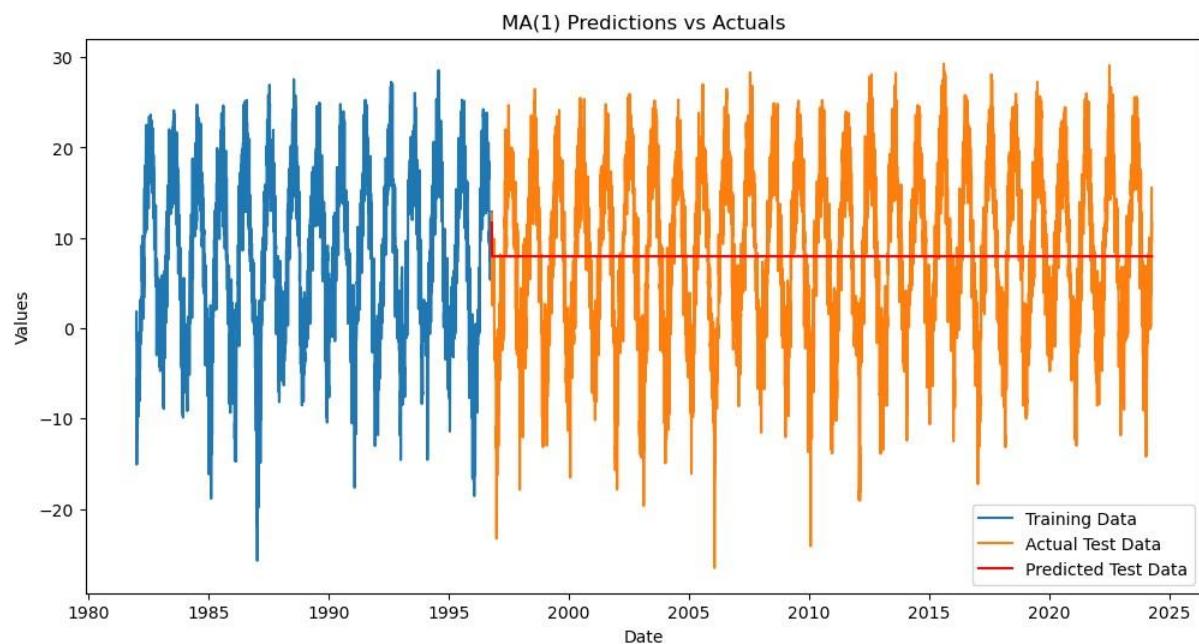
Nasz model jest zapisywany i podawany do funkcji wizualizującej i oceniającej poprawność modelu.

```
In [33]: # Funkcja do wizualizacji i oceny modelu
def plot_and_evaluate(model, train_data, test_data, order):
    # Prognozowanie
    start = len(train_data)
    end = start + len(test_data) - 1
    predictions = model.predict(start=start, end=end)

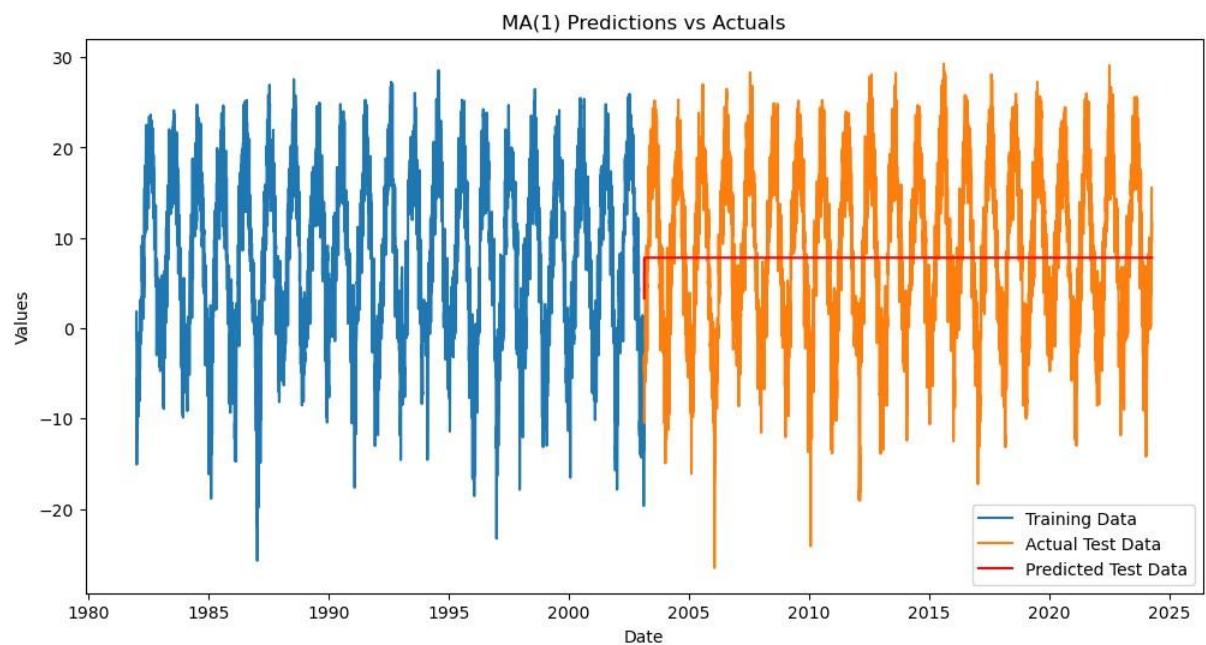
    # Wizualizacja
    plt.figure(figsize=(12, 6))
    plt.plot(train_data.index, train_data, label='Training Data')
    plt.plot(test_data.index, test_data, label='Actual Test Data')
    plt.plot(test_data.index, predictions, label='Predicted Test Data', color='red')
    plt.title(f'MA({order}) Predictions vs Actuals')
    plt.xlabel('Date')
    plt.ylabel('Values')
    plt.legend()
    plt.show()

    # Obliczanie i drukowanie błędów
    mse = mean_squared_error(test_data, predictions)
    rmse = sqrt(mse)
    print(f"Mean Squared Error for MA({order}): {mse}")
    print(f"Root Mean Squared Error for MA({order}): {rmse}")
```

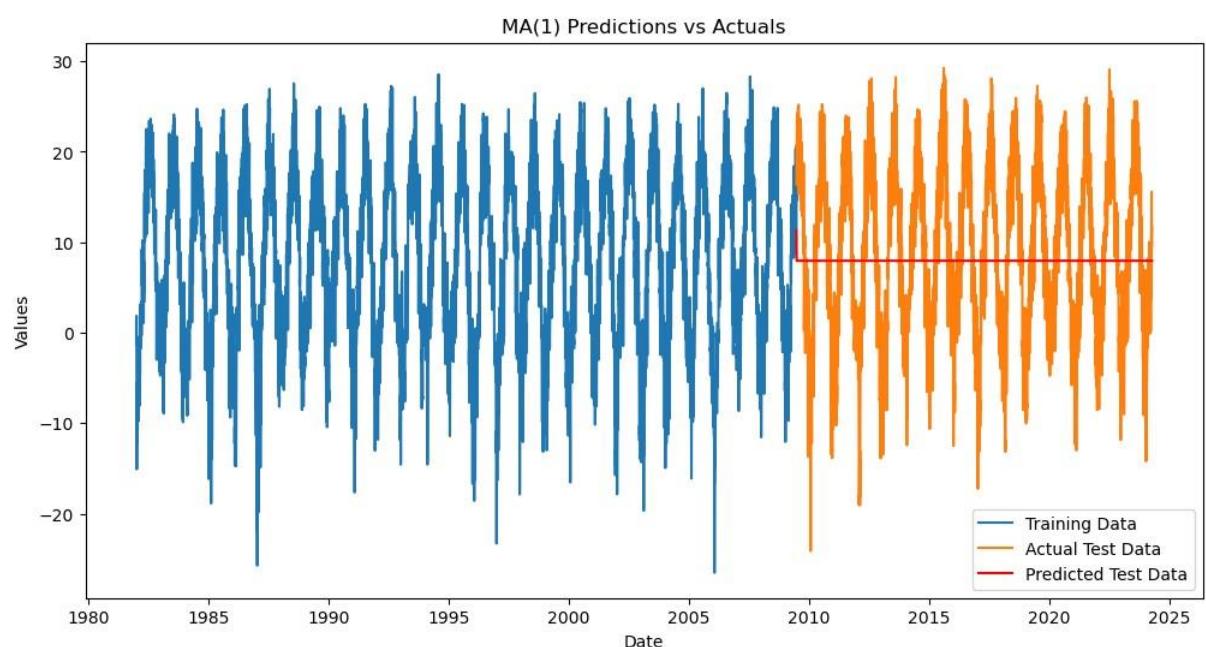
Wyniki Modelu MA



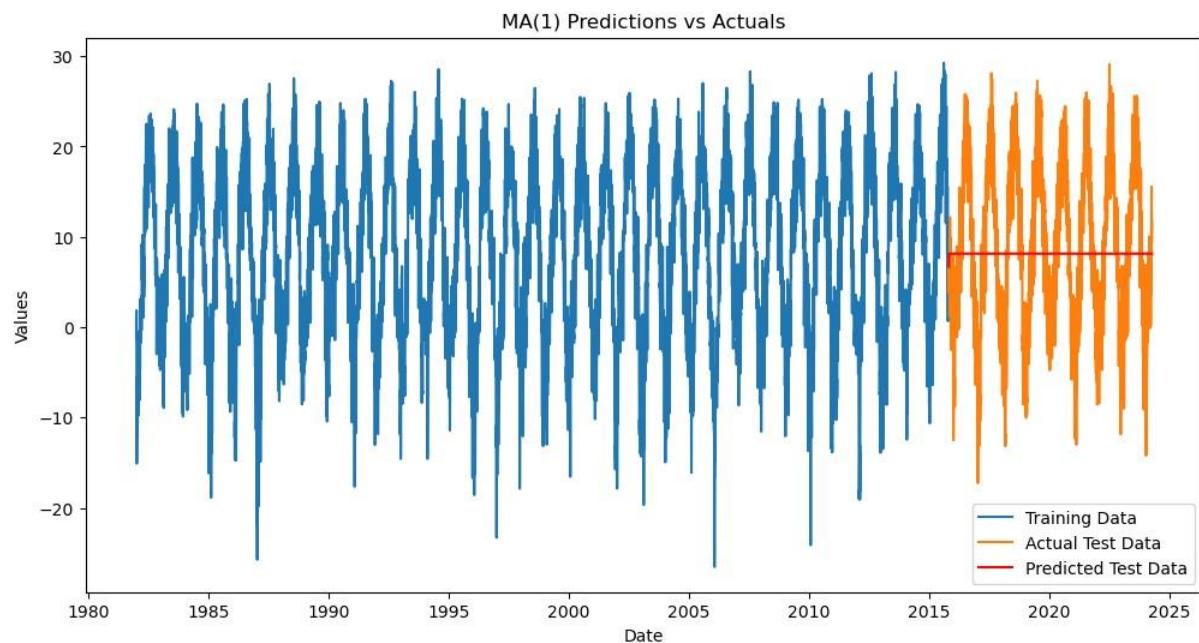
Mean Squared Error for MA(1): 87.28828652189236
Root Mean Squared Error for MA(1): 9.342820051884354



Mean Squared Error for MA(1): 86.51826765556727
Root Mean Squared Error for MA(1): 9.301519642271755



Mean Squared Error for MA(1): 87.32347286019576
Root Mean Squared Error for MA(1): 9.34470293054818



Mean Squared Error for MA(1): 81.61005887563951
Root Mean Squared Error for MA(1): 9.033828583476637

Interpretacja wyników

Niestety i w tym przypadku poprawność modelu jest niezadowalająca. Model MA jest użyteczny, gdy struktura autokorelacji szeregu czasowego szybko zanika. W naszym przypadku autokorelacja funkcji jest utrzymywana bardzo długo. Do tego średnia wartość naszych danych mimo dużych ich fluktuacji jest stała, bardzo to zakłocha działanie modelu. Proporcje zestawów train/test nie mają większego znaczenia.

Model ARIMA

Wstęp techniczny do modelu ARIMA

Model ARIMA, czyli autoregresyjny model ze średnią ruchomą zintegrowany (AutoRegressive Integrated Moving Average), jest zaawansowaną formą modeli AR i MA, zaprojektowaną specjalnie do modelowania szeregów czasowych, które nie są stacjonarne. Model ARIMA łączy różnicowanie szeregu czasowego (aby uczynić go stacjonarnym), autoregresję (AR) oraz średnią ruchomą (MA).

Równanie ogólne dla modelu ARIMA(p, d, q), gdzie:

- p to rząd komponentu autoregresyjnego,
- d to stopień różnicowania potrzebny do osiągnięcia stacjonarności szeregu,
- q to rząd komponentu średniej ruchomej,

można zapisać w następujący sposób:

$$(1 - \phi_1 B - \phi_2 B^2 - \dots - \phi_p B^p)(1 - B)^d X_t = (1 + \theta_1 B + \theta_2 B^2 + \dots + \theta_q B^q)\varepsilon_t$$

gdzie:

- B to operator przesunięcia wstecz (tzw. backshift operator), gdzie $B^k X_t = X_{t-k}$,
- $\phi_1, \phi_2, \dots, \phi_p$ to parametry komponentu AR,
- $\theta_1, \theta_2, \dots, \theta_q$ to parametry komponentu MA,
- ε_t to biały szum (szereg stochastyczny o zerowej średniej i stałej wariancji, niezależny dla różnych t).

Proces różnicowania $(1 - B)^d X_t$ służy do transformacji szeregu niestacjonarnego w stacjonarny poprzez usuwanie trendów i sezonowości, co umożliwia zastosowanie technik AR i MA do przetworzonego szeregu.

W naszym przypadku dane są już stacjonarne, więc możemy ostatecznie mówić o użyciu modelu ARMA o parametrach: (p,0,q)(0,0,0)(0)

Proces dobieranie parametrów dla modelu ARIMA

Do wybrania parametrów modelu użyjemy funkcji `auto_arima` z pakietu `statsmodels`. Ten sposób podejścia do procesu wybierania parametrów jest bardziej wygodny niż manualne pisanie pętli `for` tak jak zrobiliśmy to w poprzednich modelach. Wystarczy wywołać gotową funkcję która poda nam najlepsze parametry bez naszej ingerencji.

```
In [42]: # Funkcja do wybierania najlepszego modelu
def find_best_arima(train_data, seasonal=False):
    # Użycie auto_arima do automatycznego znalezienia najlepszego modelu ARIMA
    arima_model = auto_arima(train_data, seasonal=seasonal, trace=True, suppress_warnings=True, stepwise=True)
    return arima_model
```

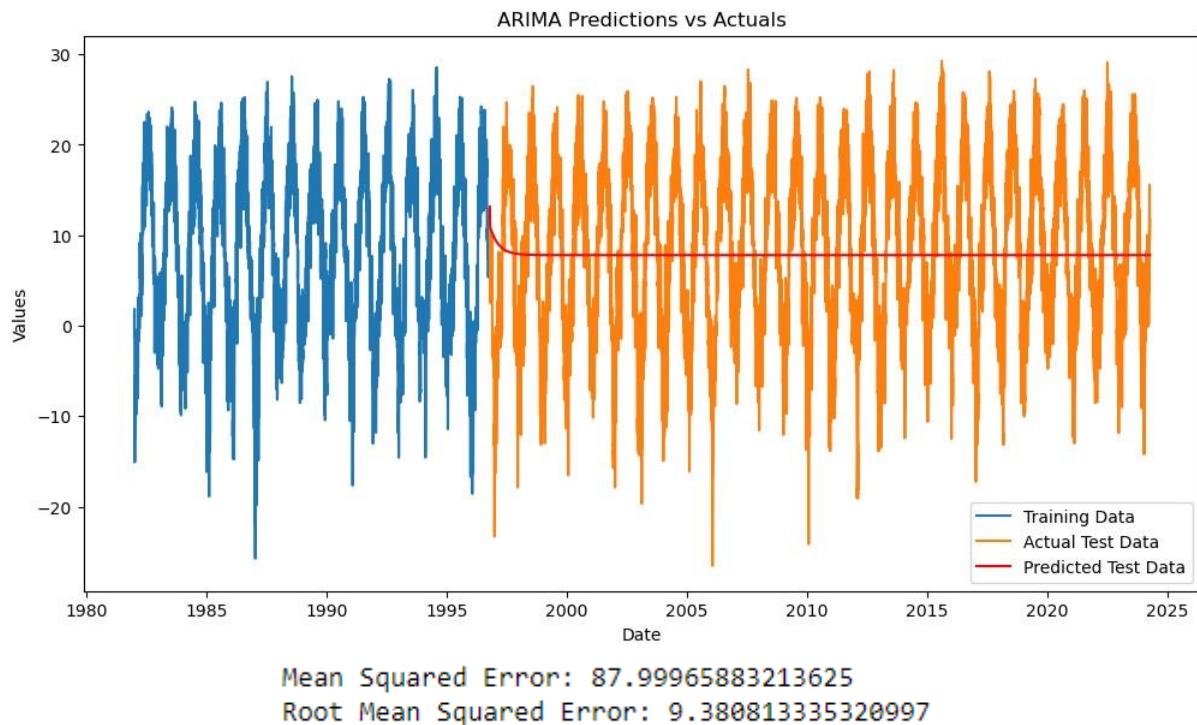
Nasz otrzymany model możemy podać do funkcji wizualizującej

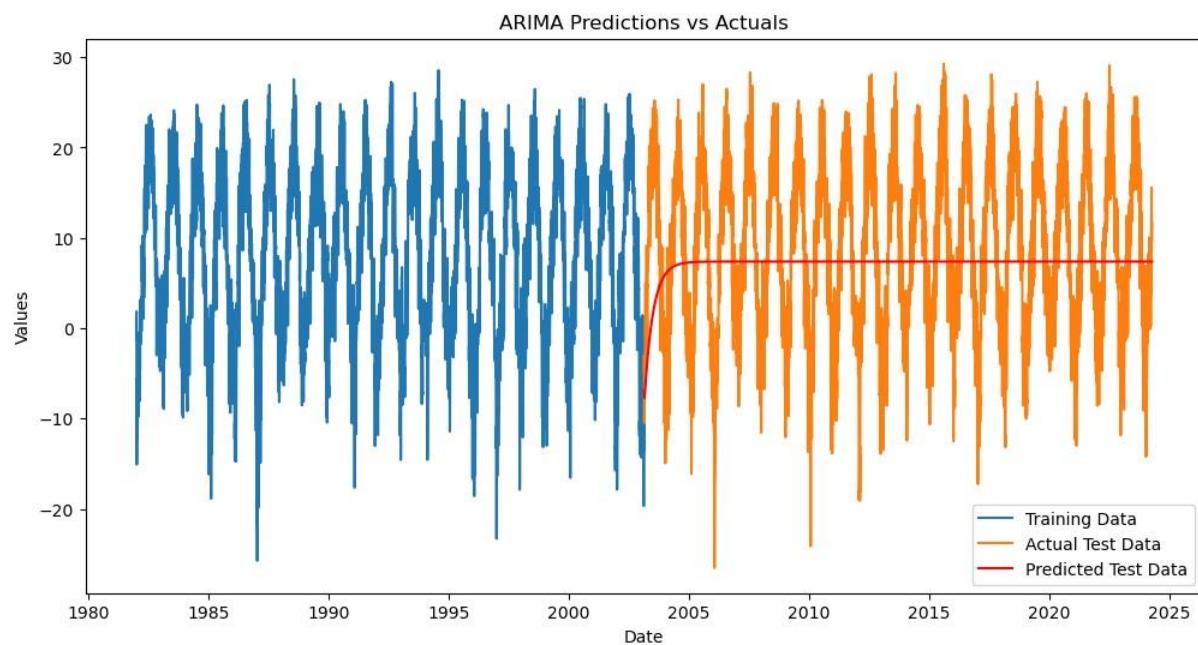
```
In [43]: # Funkcja do wizualizacji i sprawdzenia modelu
def plot_and_evaluate_arima(model, train_data, test_data):
    # Prognozowanie
    predictions = model.predict(n_periods=len(test_data))

    # Wizualizacja
    plt.figure(figsize=(12, 6))
    plt.plot(train_data.index, train_data, label='Training Data')
    plt.plot(test_data.index, test_data, label='Actual Test Data')
    plt.plot(test_data.index, predictions, label='Predicted Test Data', color='red')
    plt.title('ARIMA Predictions vs Actuals')
    plt.xlabel('Date')
    plt.ylabel('Values')
    plt.legend()
    plt.show()

    # Obliczanie i drukowanie błędów
    mse = mean_squared_error(test_data, predictions)
    rmse = sqrt(mse)
    print(f"Mean Squared Error: {mse}")
    print(f"Root Mean Squared Error: {rmse}")
```

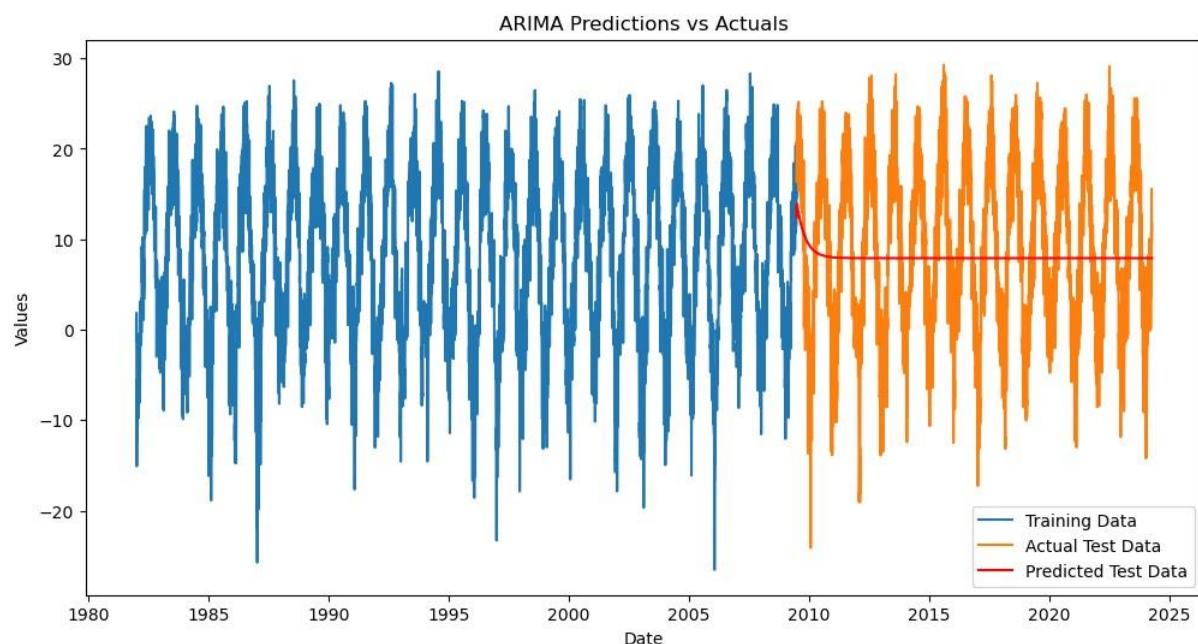
Wyniki Modelu ARIMA





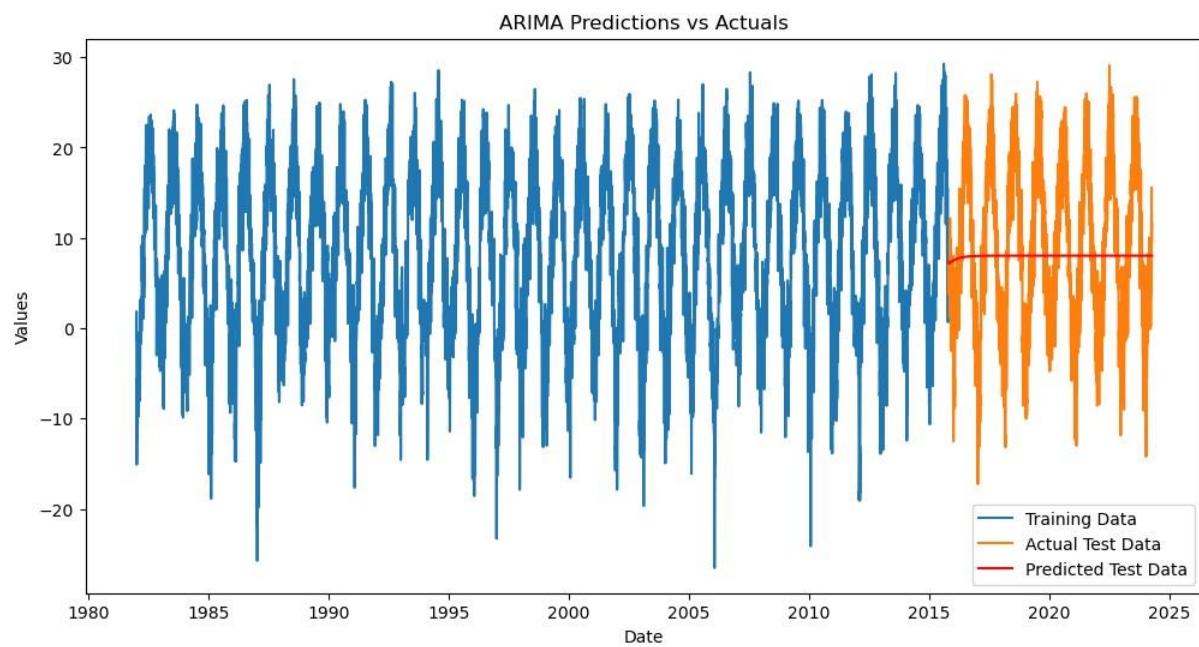
Mean Squared Error: 90.2747408899534

Root Mean Squared Error: 9.50130206287293



Mean Squared Error: 86.65455927316489

Root Mean Squared Error: 9.308843068457266



Mean Squared Error: 81.61429675369094

Root Mean Squared Error: 9.034063136468049

Interpretacja wyników

Model ARIMA też napotyka problemy w dokładnym przewidywaniu temperatury. Połączenie modeli AR i MA nie przyniosło pozytywnym skutków. Dane są zbyt skomplikowane naszego modelu i zostaje on przeładowany informacjami takimi jak sezonowość. Proporcje zestawów train/test nie mają dużego znaczenia.

Model SARIMA

Model SARIMA, rozszerzenie modelu ARIMA, uwzględnia sezonowość w danych czasowych. Model SARIMA, czyli Seasonal AutoRegressive Integrated Moving Average, pozwala na modelowanie zarówno niestacjonarności jak i sezonowości w szeregu czasowym. Jest to szczególnie przydatne w analizie danych, które wykazują wzory powtarzające się w regularnych odstępach czasu, takich jak dane miesięczne dotyczące sprzedaży czy temperatury.

Równanie dla modelu SARIMA($p, d, q)(P, D, Q)_s$, gdzie:

- p, d, q są parametrami niesezonowego komponentu ARIMA, opisującego autoregresję, różnicowanie i średnią ruchomą,
- P, D, Q są parametrami sezonowego komponentu modelu, opisującymi autoregresję, różnicowanie i średnią ruchomą na poziomie sezonowym,
- s to długość sezonu (np. 12 dla miesięcznych danych z rocznym cyklem sezonowym),

można zapisać jako:

$$(1 - \Phi_1 B^s - \Phi_2 B^{2s} - \dots - \Phi_P B^{Ps})(1 - \phi_1 B - \phi_2 B^2 - \dots - \phi_p B^p)(1 - B)^d(1 - B^s)^D X$$

gdzie:

- B jest operatorem przesunięcia wstecz,
- Φ_i i Θ_i to parametry sezonowe odpowiednio dla AR i MA,
- ϕ_i i θ_i to niesezonowe parametry AR i MA,
- ε_t oznacza biały szum.

Model SARIMA pozwala na oddzielne modelowanie dynamiki sezonowej i niesezonowej, co jest jego dużą zaletą w przypadkach, gdy sezonowość odgrywa znaczącą rolę. Również tutaj, podobnie jak w ARIMA, używa się różnicowania, aby uczynić szereg stacjonarnym, zarówno na poziomie ogólnym, jak i sezonowym.

Model SARIMA może być dobrym rozwiązaniem na problemy z niezadowalającą jakością predykcji. Jest on w stanie zrozumieć takie zawiłości jak sezonowość, która uniemożliwiała na dokładne estymowanie predykcji.

Próba użycie modelu

Niestety próby użycia modelu SARIMA okazały się bezowocne. Problemem jest ilość danych w naszym sezonie, jest to liczba dni w roku, czyli 365. Przy tak dużym parametrze s próba stworzenia nawet prostego modelu staje się niemożliwa ze względu na ograniczenia pamięci RAM.

Próba użycia modelu SARIMA skończyła się niepowodzeniem. Model dla najmniejszego zestawu train miał wielkość 5 GB. Jest to zdecydowanie za dużo.

```
In [53]: # # Określenie parametrów
# order = (1, 0, 1) # Parametry niesezonowe: (p, d, q)
# seasonal_order = (1, 0, 1, 365) # Parametry sezonowe: (P, D, Q, S)

# # Założymy, że masz już wcześniej przygotowane i podzielone dane: datasets['train_35'] i datasets['test_35']
# fit_and_forecast_sarima(datasets['train_35']['T2M'], datasets['test_35']['T2M'], order, seasonal_order)
```

Model Facebook Prophet

Odpowiedzią na nasze problemy może być użycie modelu Prophet stworzonego przez Facebook'a. Model Facebook Prophet to nowoczesne narzędzie do prognozowania szeregów czasowych, zaprojektowane przez zespół z Facebooka, aby umożliwić analitykom i programistom efektywne modelowanie i przewidywanie danych czasowych.

Prophet został specjalnie stworzony, by radzić sobie z danymi posiadającymi silne sezonowości oraz brakujące dane, co czyni go szczególnie przydatnym w praktycznych zastosowaniach, gdzie tradycyjne metody modelowania szeregów czasowych mogą mieć trudności.

Właśnie w takiej sytuacji się znajdujemy więc użycie tego modelu powinno dać dobre wyniki.

Kod potrzebny do działania modelu

By nasz kod działał poprawnie na początku musimy usunąć w danych indeksowanie co było potrzebne w poprzednich modelach. Następnie wybieramy, że w naszych danych występuje sezonowość a dane były zbierane dziennie.

```
In [54]: def forecast_with_prophet(train_data, test_data):
    # Prepare data for Prophet
    train_data_reset = train_data.reset_index()
    prophet_train = train_data_reset.rename(columns={'DATE': 'ds', 'T2M': 'y'})

    # Create and fit the Prophet model
    model = Prophet(daily_seasonality=True, yearly_seasonality=True)
    model.fit(prophet_train)

    # Create future dataframe for predictions
    future_dates = model.make_future_dataframe(periods=len(test_data), freq='D')

    # Forecast
    forecast = model.predict(future_dates)
    forecast_filtered = forecast[['ds', 'yhat']].tail(len(test_data))

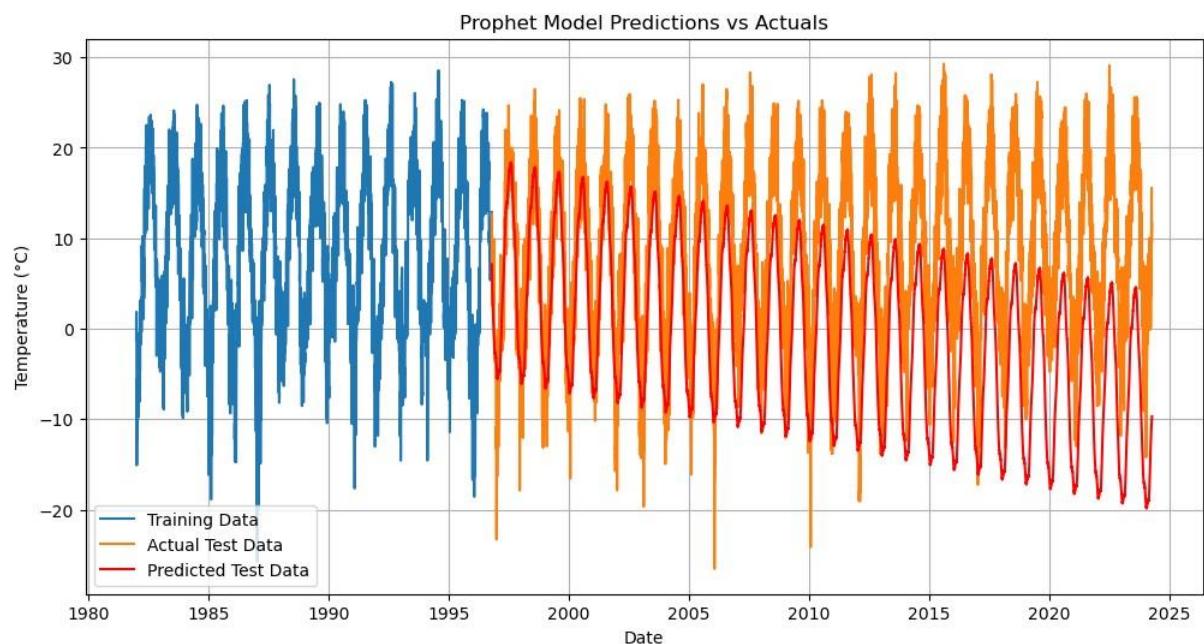
    return forecast_filtered.set_index('ds'), model
```

Następnie używamy funkcji tworzącej nam wizualizacje i mierzącej dokładność przewidynań.

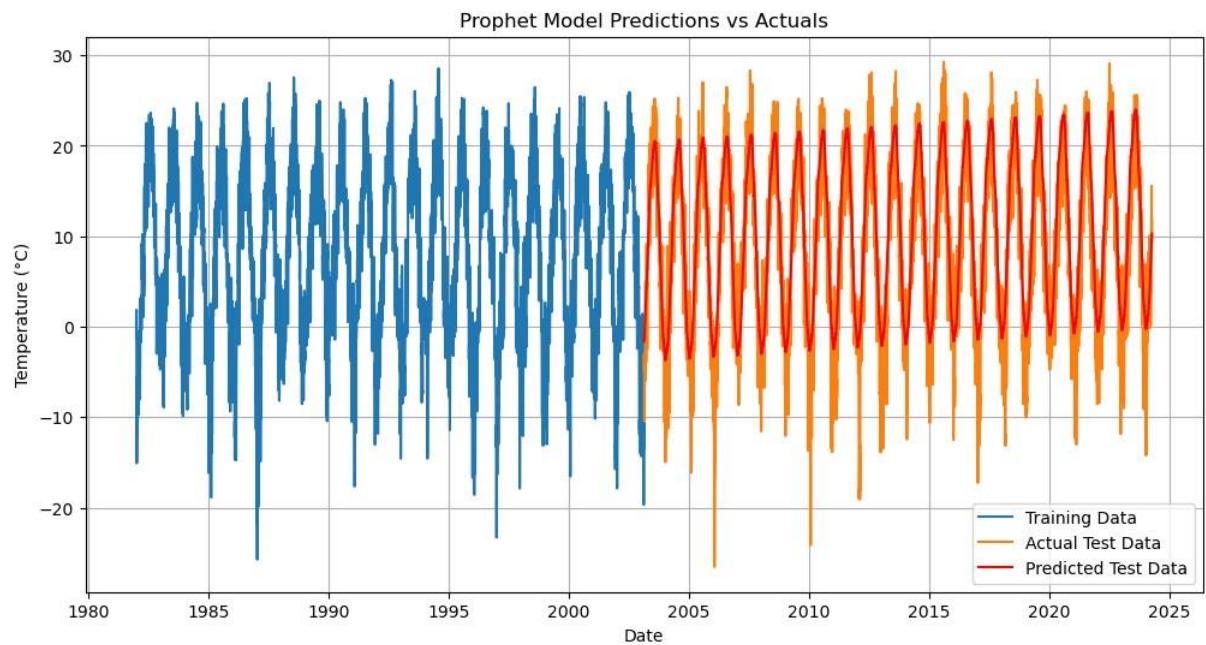
```
In [55]: def plot_and_evaluate_prophet(train_data, test_data, forecast):
    # Plotting
    plt.figure(figsize=(12, 6))
    plt.plot(train_data.index, train_data, label='Training Data')
    plt.plot(test_data.index, test_data, label='Actual Test Data')
    plt.plot(forecast.index, forecast['yhat'], label='Predicted Test Data', color='red')
    plt.title('Prophet Model Predictions vs Actuals')
    plt.xlabel('Date')
    plt.ylabel('Temperature (°C)')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Evaluate forecast
    mse = mean_squared_error(test_data, forecast['yhat'])
    rmse = sqrt(mse)
    print(f"Mean Squared Error: {mse}")
    print(f"Root Mean Squared Error: {rmse}")
```

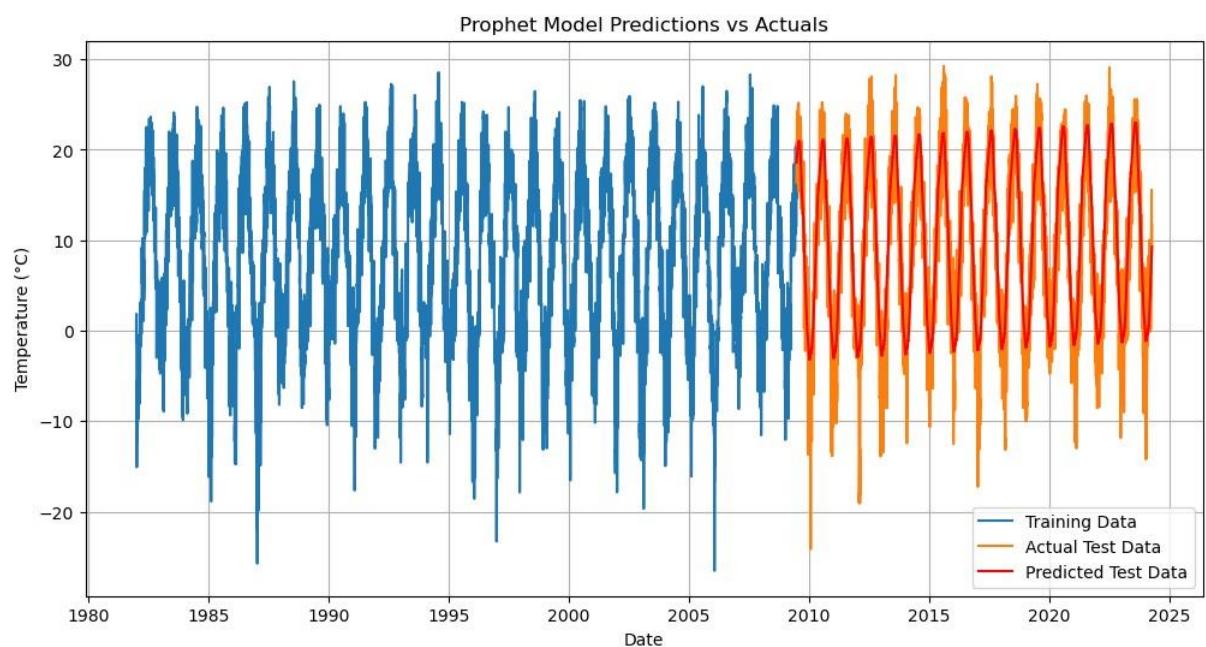
Wyniki Modelu Facebook Prophet



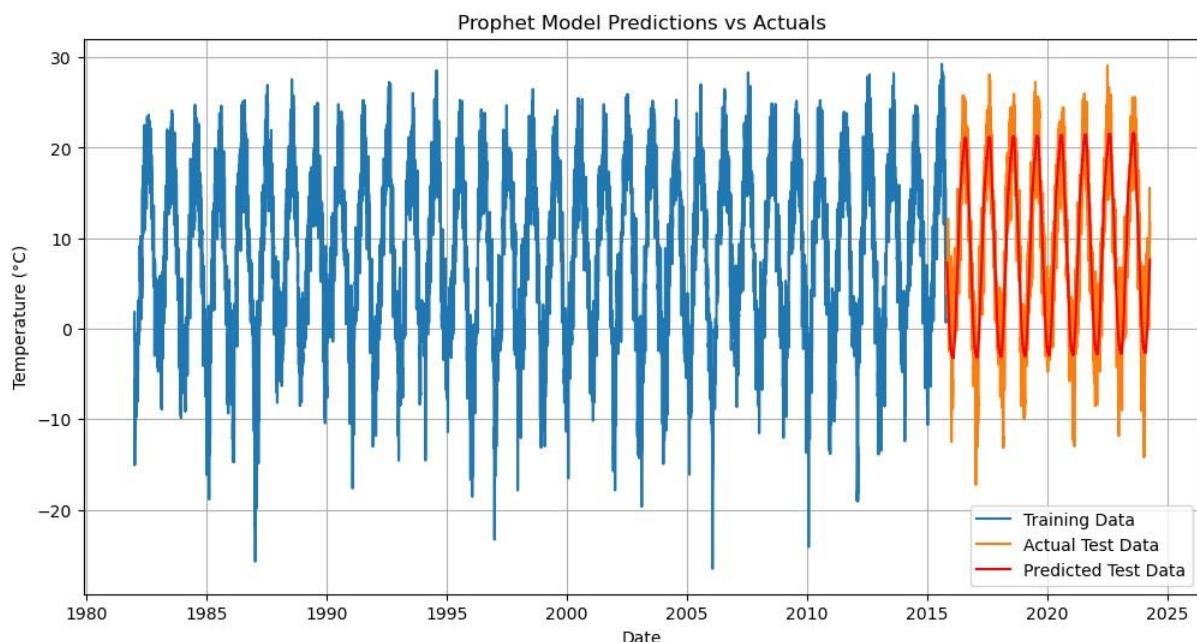
Mean Squared Error: 125.76875103264008
Root Mean Squared Error: 11.214666782060004



Mean Squared Error: 17.080908152091627
 Root Mean Squared Error: 4.1329055338939975



Mean Squared Error: 15.801860225983248
 Root Mean Squared Error: 3.9751553713010073



Mean Squared Error: 13.560817368413813
Root Mean Squared Error: 3.6825015096281786

Interpretacja wyników

Działanie modelu Facebook Prophet jest zadowalające. Model dobrze radzi sobie z przewidywaniem sezonowości.

Proporcje zestawów train/test ma znaczenie, pierwszy model który miał proporcje 35%/65% błędnie myślał, że wartości temperatury zaczynają spadać, ponieważ koniec części train był okresem, kiedy temperatura małały, co zmyliło model. Model 65%/35% już poradził sobie z podobną sytuacją i przewidział, że temperatura wróci do swojego naturalnego cyklu.

Mylące może być patrzenie tylko na MSE. Wartości niego są dość wysokie, ponieważ model ma problemy z predykcją najbardziej odstających wartości, jednak nie wpływa to bardzo na używalność naszego modelu. Nie ma to dla nas dużego znaczenia czy model przewidzi temperaturę -10 stopni Celsjusza, czy -30 stopni Celsjusza. W obydwu przypadkach i tak będziemy chcieli zostać w domu. Lepszym miernikiem poprawności jest RMSE.

Używanie Modeli AR, MA i ARIMA na trendzie naszych danych

Wstęp techniczny

Modele AR, MA i ARIMA nie dały nam zadowalających wyników, na podstawie tej informacji moglibyśmy dojść do wniosku, że są one bezużyteczne przy predykcji pogody. Nie jest to jednak prawda. Z naszych oryginalnych danych możemy wyciągnąć trend, czyli ogólną informację, czy nasze dane rosną czy maleją. Po prowadzeniu nowych danych do modeli możemy uzyskać predykcje ich ogólnej zmiany w czasie.

Modyfikowanie danych regresja liniowa

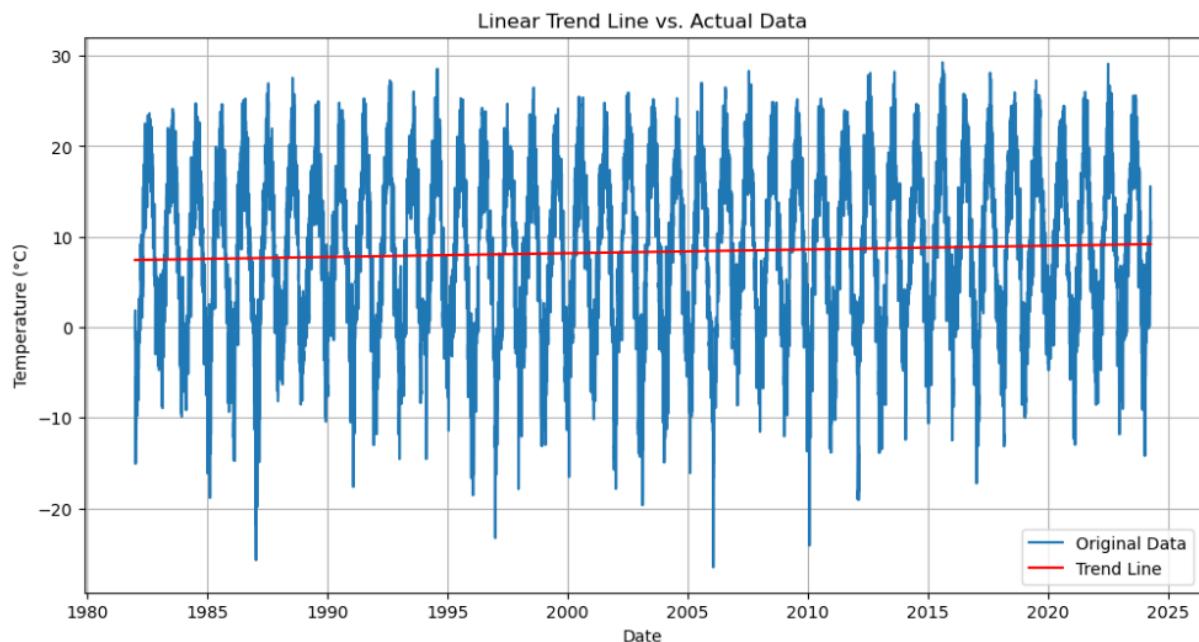
Jednym ze sposobów uzyskania linii trendu jest użycie regresji liniowej.

```
In [60]: # Przygotowanie danych dla regresji liniowej
data_lin_trend = data_xxx.copy()
data_lin_trend['Time'] = np.arange(len(data_lin_trend.index))
data_lin_trend['T2M'].interpolate(method='linear', inplace=True)

# Model regresji liniowej
model_lin = LinearRegression()
model_lin.fit(data_lin_trend['Time'].values.reshape(-1, 1), data_lin_trend['T2M'].values.reshape(-1, 1))
data_lin_trend['Trend_Linear'] = model_lin.predict(data_lin_trend['Time'].values.reshape(-1, 1))

# Wyodrębnienie danych trendu
data_lin_trend = data_lin_trend[['Trend_Linear']]
```

Po przeprowadzeniu transformacji tak wyglądają nasze nowe dane.



Modyfikowanie danych seasonal decompose

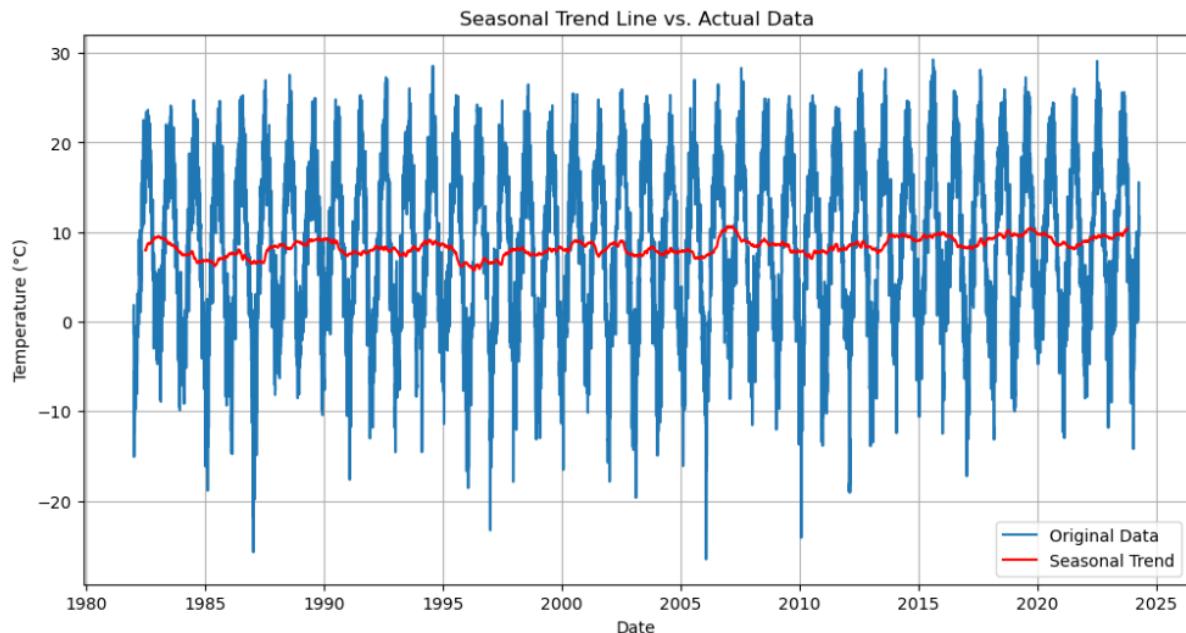
Drugim rodzajem wydobycia trendu jakim się posłużymy będzie seasonal_decompose z paczki statsmodels.tsa.seasonal.

```
In [61]: # Interpolacja danych
data_seasonal_trend = data_xxx.copy()
data_seasonal_trend['T2M'].interpolate(method='linear', inplace=True)

# Dekompozycja sezonowa
decomposition = seasonal_decompose(data_seasonal_trend['T2M'], model='additive', period=365)
data_seasonal_trend['Trend_Seasonal'] = decomposition.trend

# Wyodrębnienie danych trendu
data_seasonal_trend = data_seasonal_trend[['Trend_Seasonal']].dropna()
```

Po przeprowadzeniu transformacji tak wyglądają nasze nowe dane.



Informacje o różnicy otrzymanych trendach

Regresją liniową cechuje się większą łatwością w zarejestrowaniu ogólnego trendu naszych danych i powiedzeniu nam czy poprzeć cały zakres naszych danych rosy one czy malały. Jak widać na powyższym wykresie, występuje tendencja wzrostowa.

Dekompozycja sezonowa, jak w przykładzie z wykorzystaniem funkcji `seasonal_decompose`, to zaawansowana technika analizy szeregów czasowych, która pozwala na oddzielenie danych na trzy główne składniki: trend, sezonowość i resztę. Podczas gdy regresja liniowa skupia się głównie na wyznaczeniu ogólnego kierunku (trendu) w danych, dekompozycja sezonowa pozwala na głębsze zrozumienie struktury danych, wskazując jak poszczególne składniki przyczyniają się do obserwowanych wzorców. W tym przypadku będziemy operować tylko na trendzie wyodrębnionym przez tą funkcję. Należy też pamiętać że dekompozycja sezonowa nie jest nieomylna w wyznaczaniu trendu i w naszych danych musza znajdować się szczegółki szumu białego i sezonowości

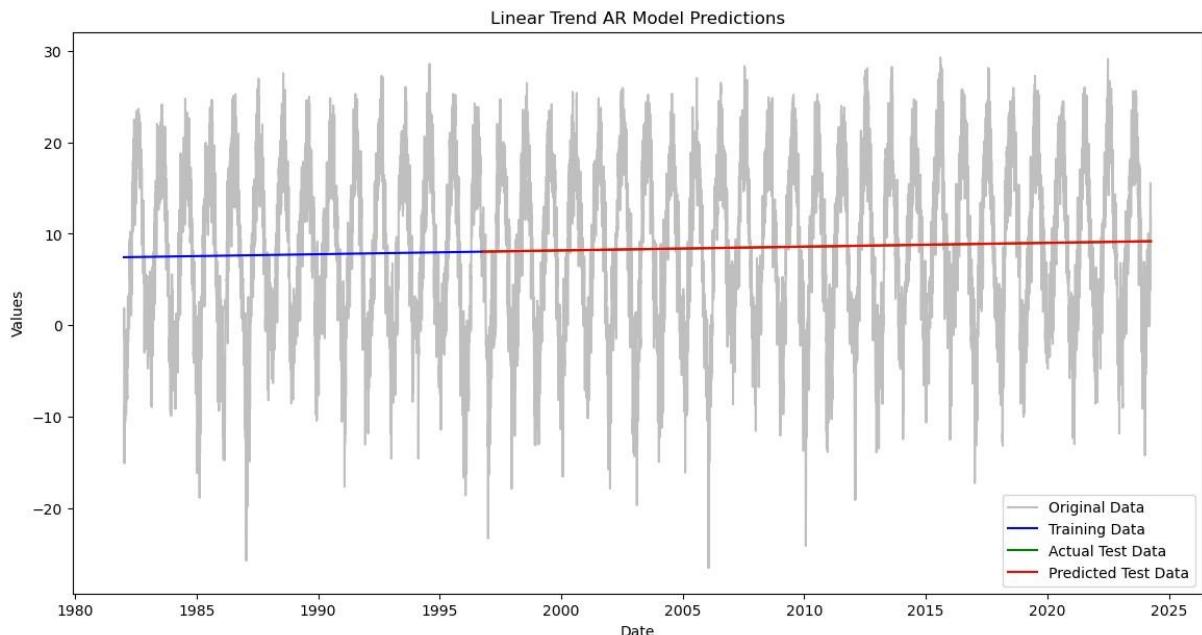
Dla porównanie sprawdzimy jak modele działają obu rodzajów uzyskania trendu

Kody modeli

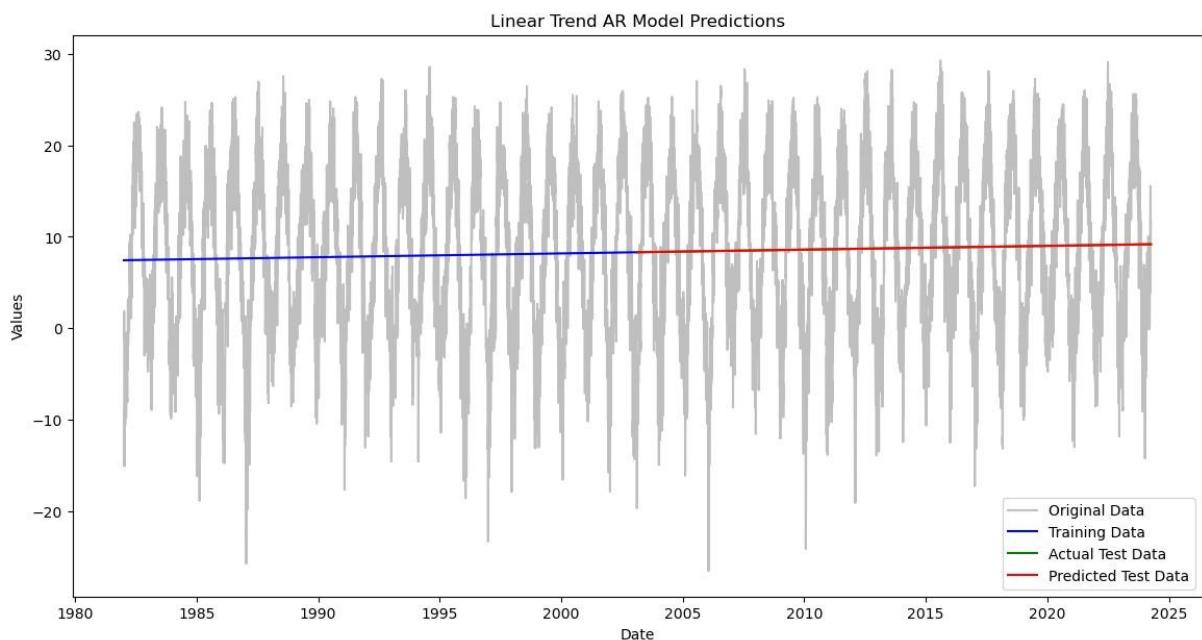
Kody modeli są adekwatne z kodem modeli podanym wcześniej, z symbolicznymi zmianami.

Regresja linienna

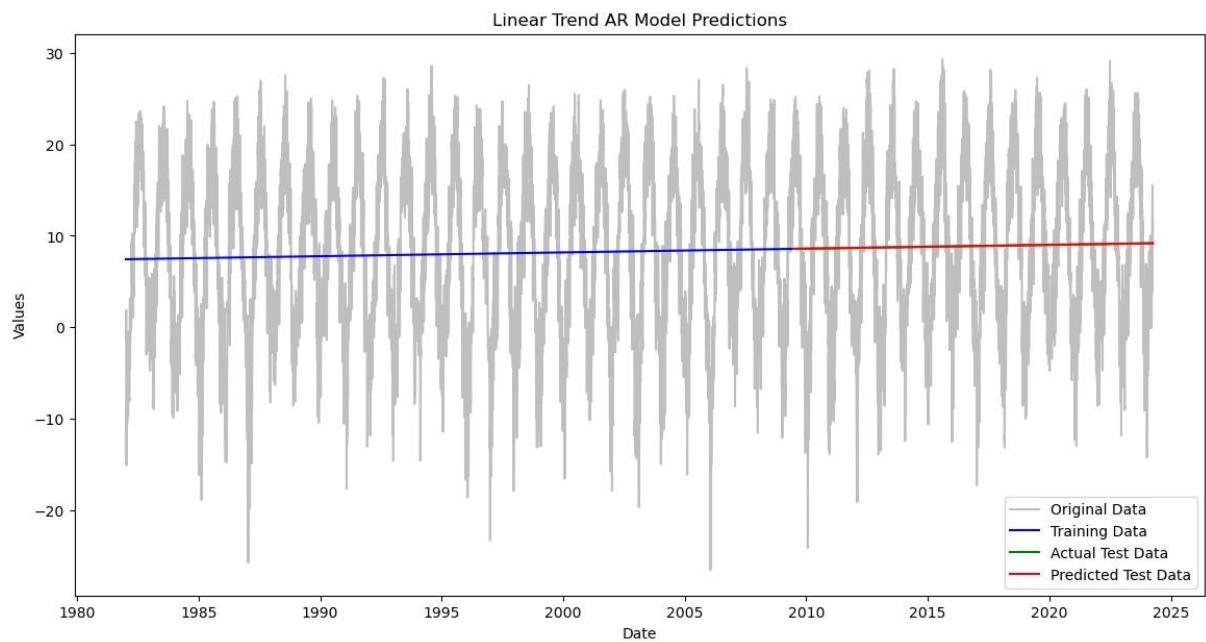
Wyniki model AR



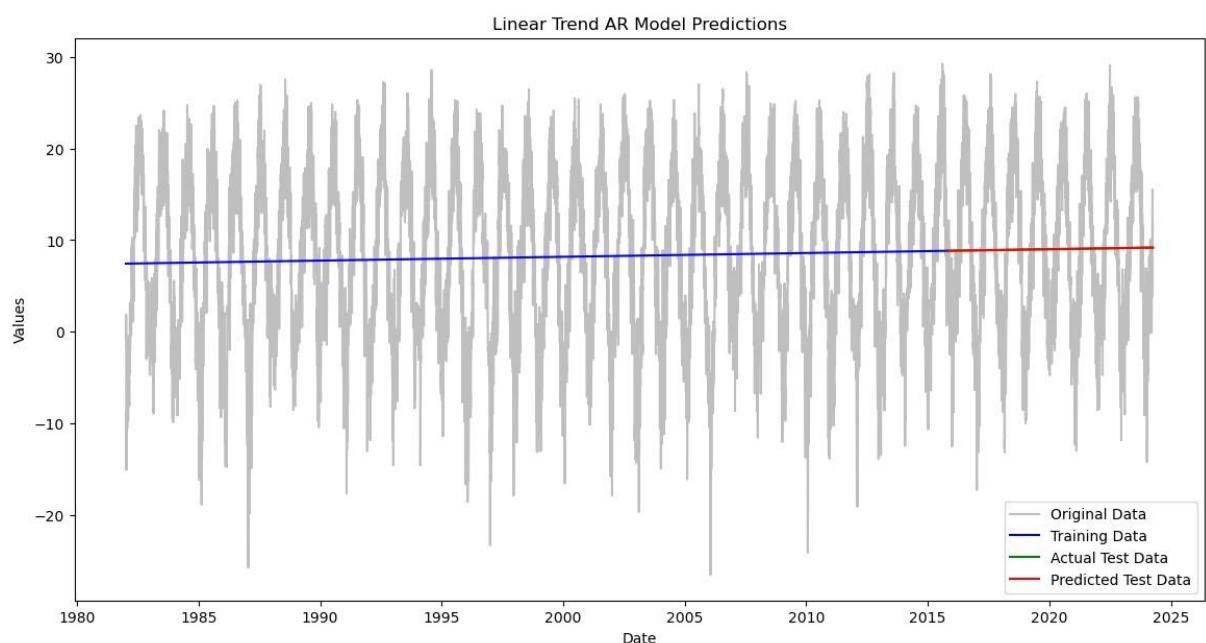
Mean Squared Error: 3.660783622556691e-21
Root Mean Squared Error: 6.050440994305036e-11



Mean Squared Error: 4.2155994496904205e-22
Root Mean Squared Error: 2.0531925018590977e-11

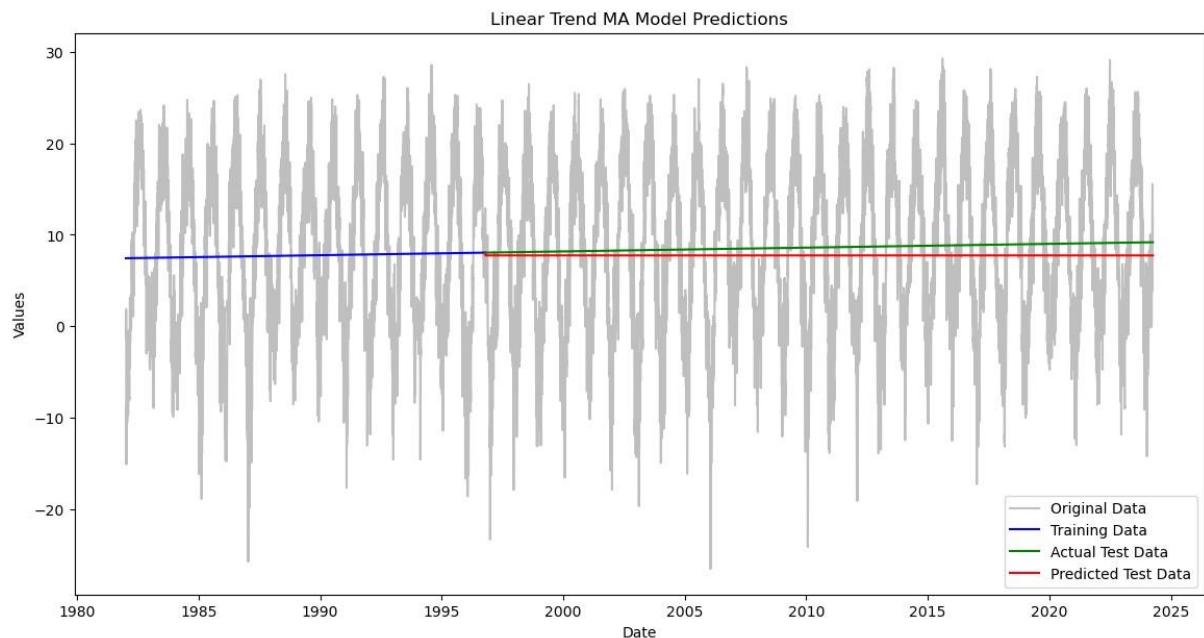


Mean Squared Error: 1.7079478547109303e-22
Root Mean Squared Error: 1.3068847901444603e-11



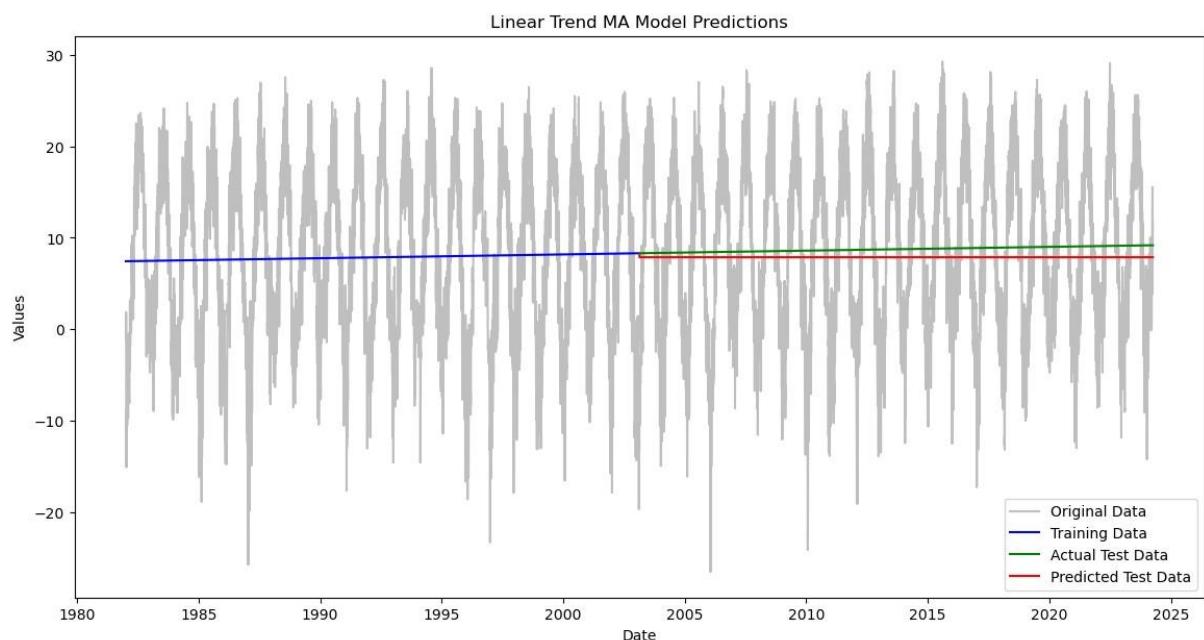
Mean Squared Error: 6.798359585249046e-24
Root Mean Squared Error: 2.6073664079390618e-12

Wyniki model MA



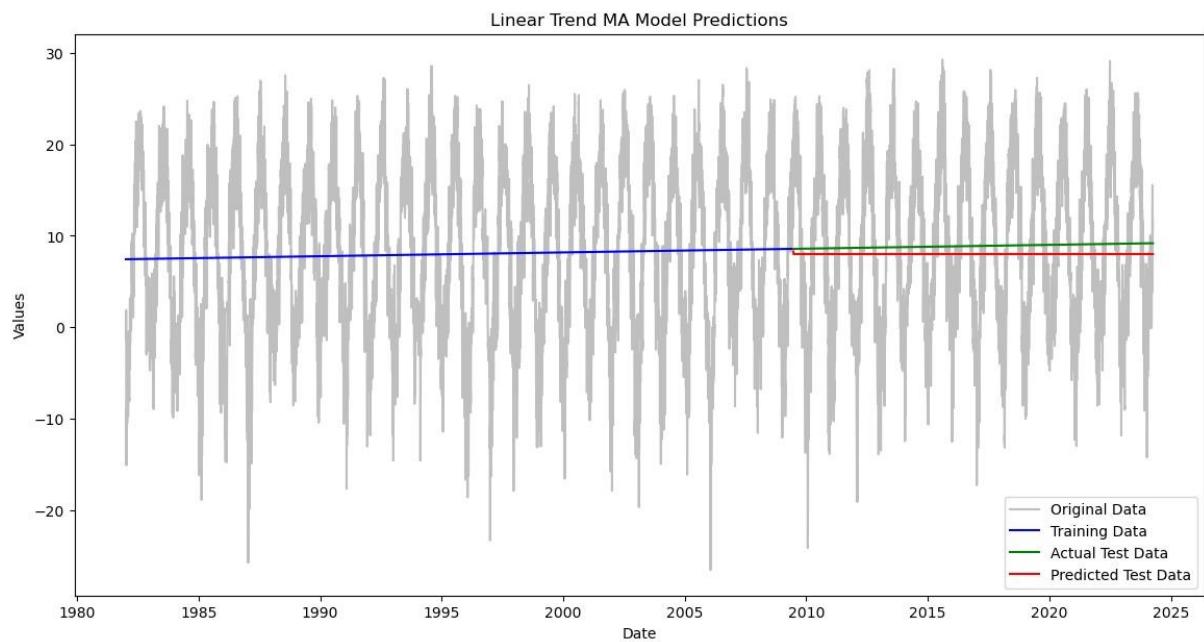
Mean Squared Error: 0.8706934138773824

Root Mean Squared Error: 0.9331095401277293



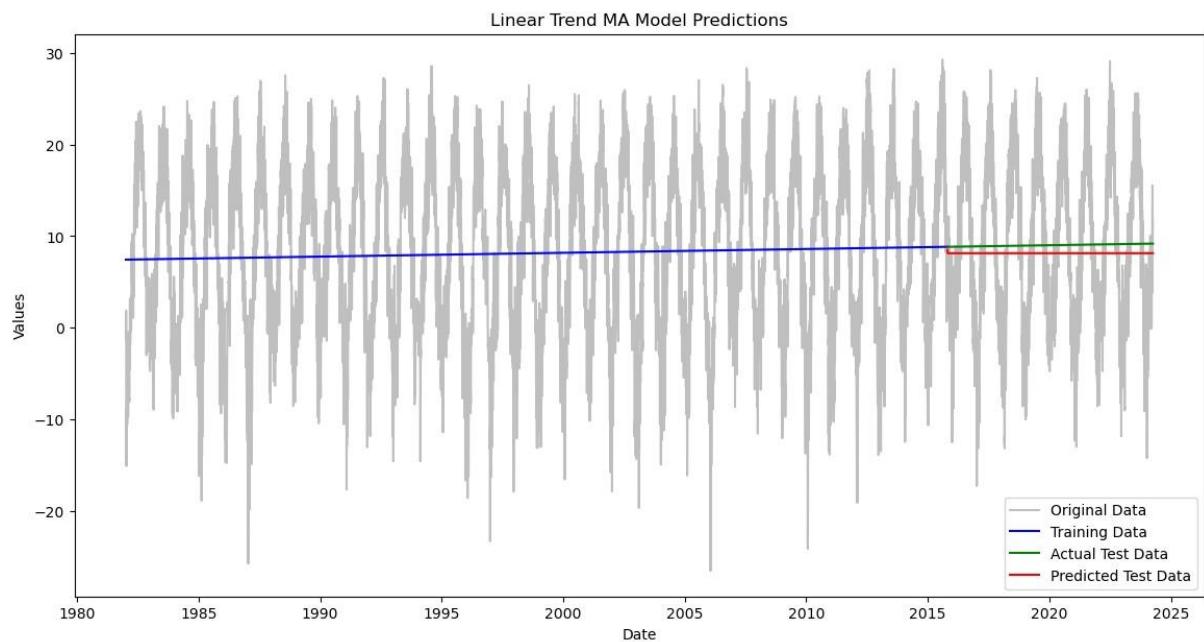
Mean Squared Error: 0.8267909313667344

Root Mean Squared Error: 0.90928044703861



Mean Squared Error: 0.7943364263323517

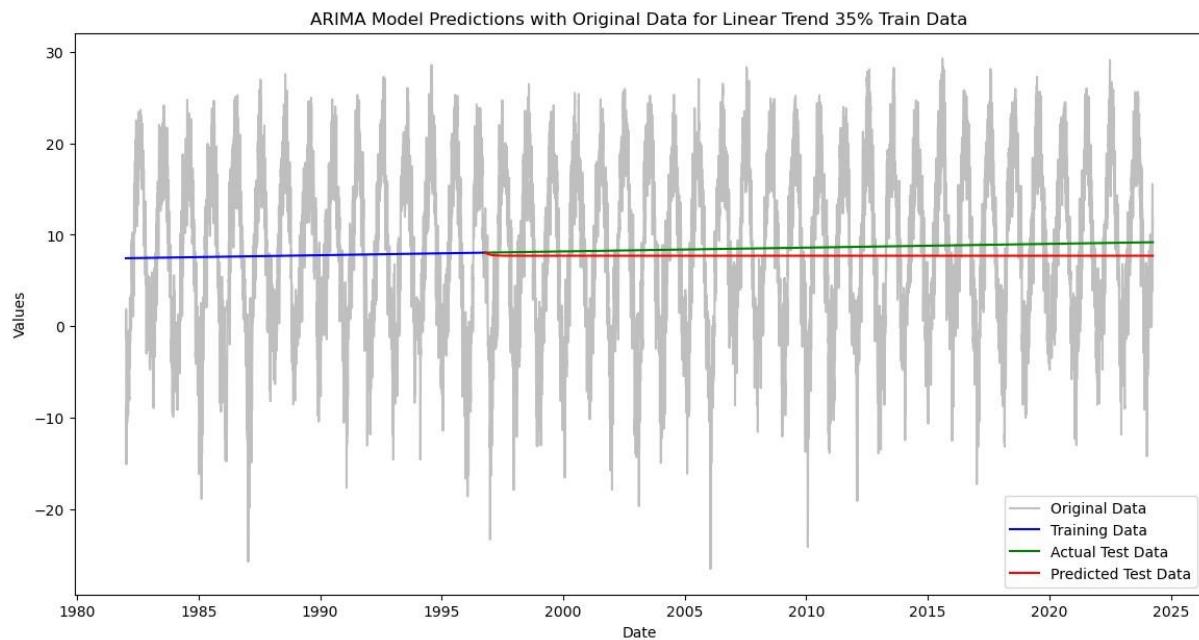
Root Mean Squared Error: 0.8912555336896101



Mean Squared Error: 0.7732688634117834

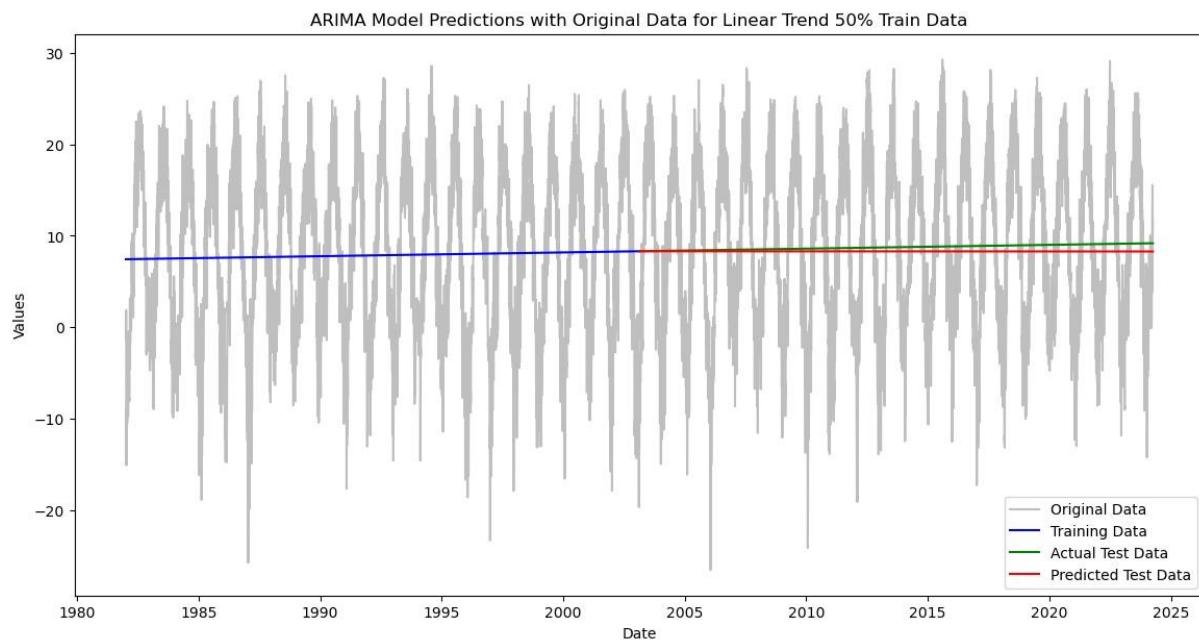
Root Mean Squared Error: 0.8793570738964822

Wyniki model ARIMA



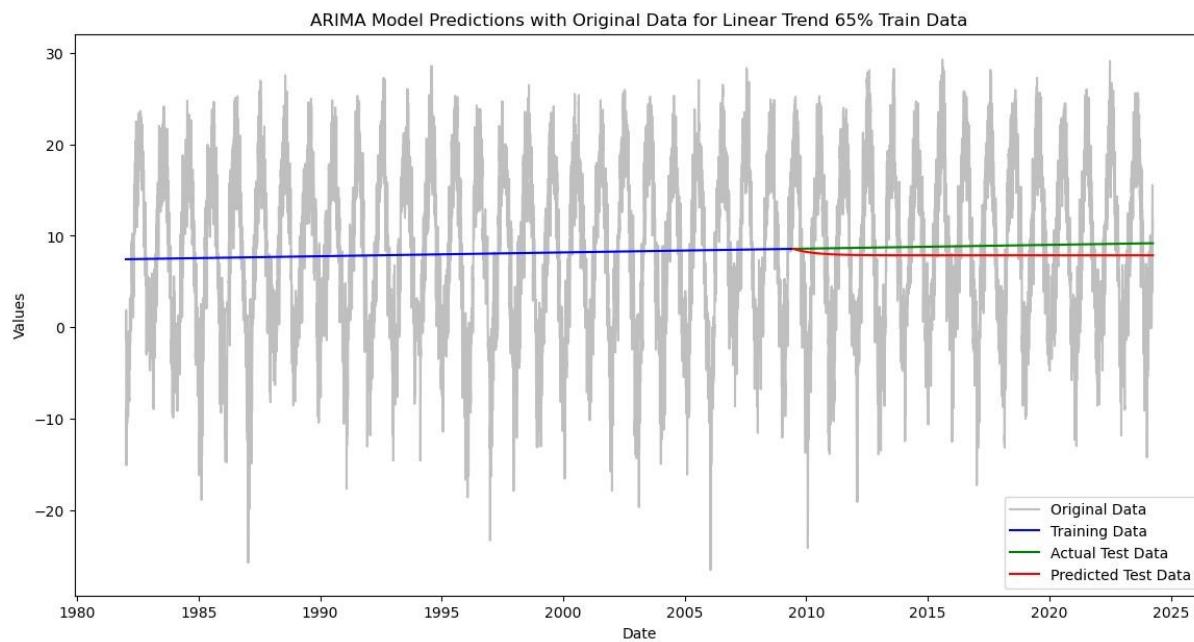
Mean Squared Error: 0.932279530744826

Root Mean Squared Error: 0.9655462343900606

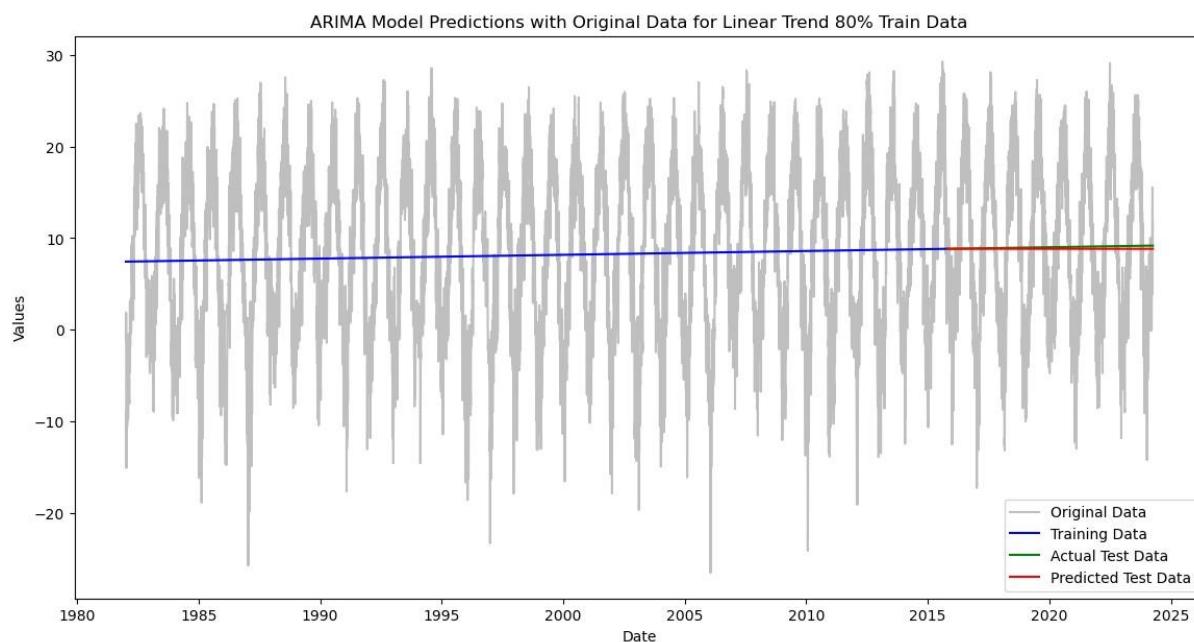


Mean Squared Error: 0.27035824639530837

Root Mean Squared Error: 0.5199598507532176



Mean Squared Error: 1.0082388024726974
 Root Mean Squared Error: 1.0041109512761512



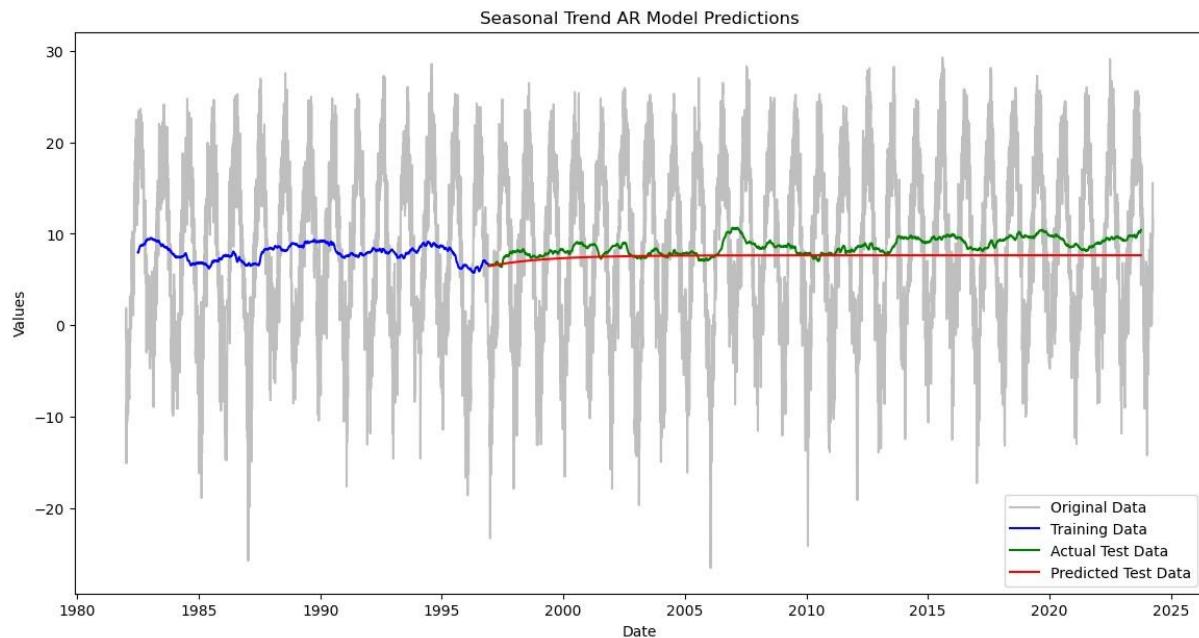
Mean Squared Error: 0.041414129306052566
 Root Mean Squared Error: 0.2035046174072042

Interpretacja wyników

Jak widać wszystkie model radzą sobie dobrze z predykcją wyników, jednak najlepiej poradził sobie model AR. Można się było tego spodziewać ponieważ dane zostały stworzone przy pomocy regresji liniowej. Najwięcej problemów miał model MA, który notorycznie zaniżał dane w porównaniu z danymi realnymi. Podobny problem miał model ARIMA lecz w mniejszym stopniu.

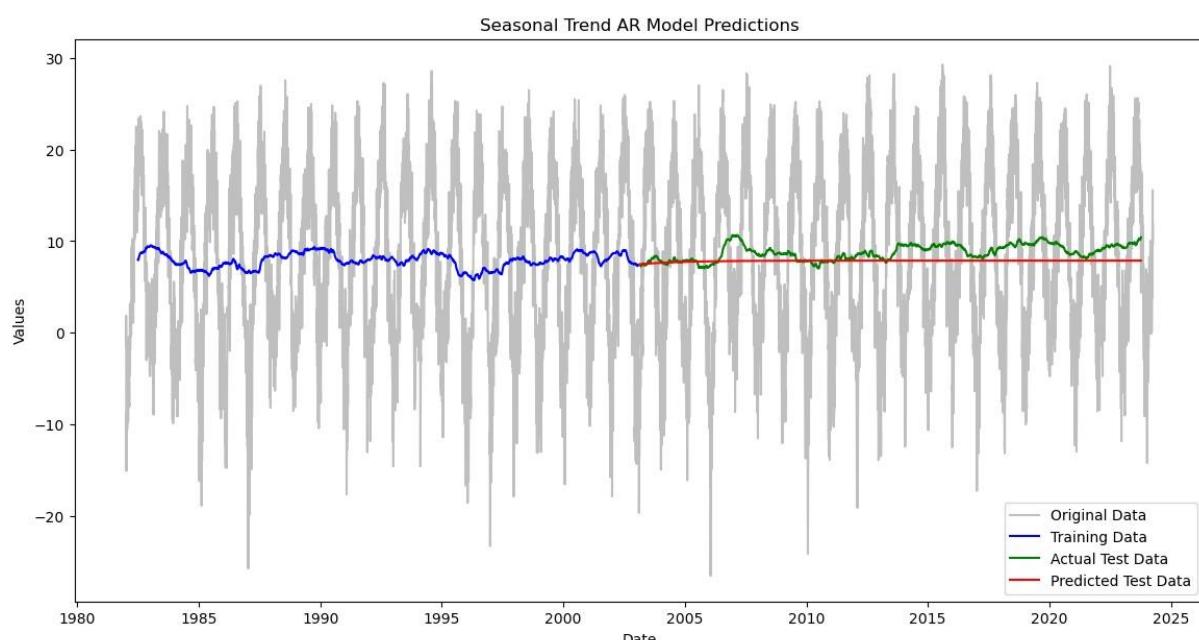
Seasonal decompose

Wyniki model AR



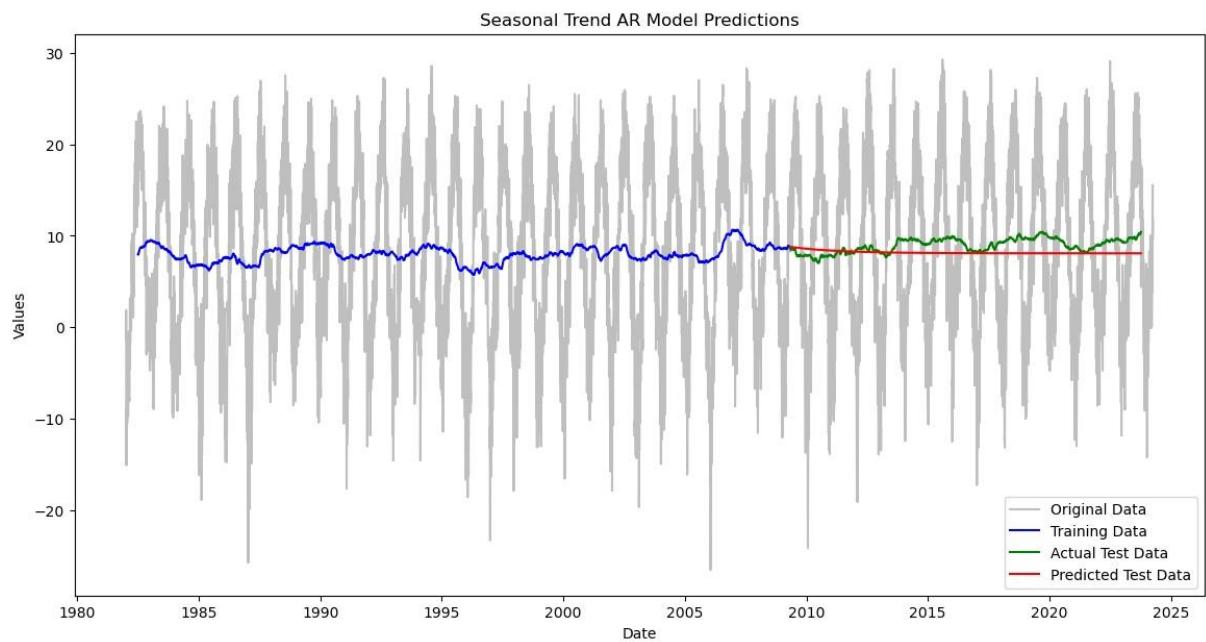
Mean Squared Error: 1.7191782257155306

Root Mean Squared Error: 1.3111743689210564

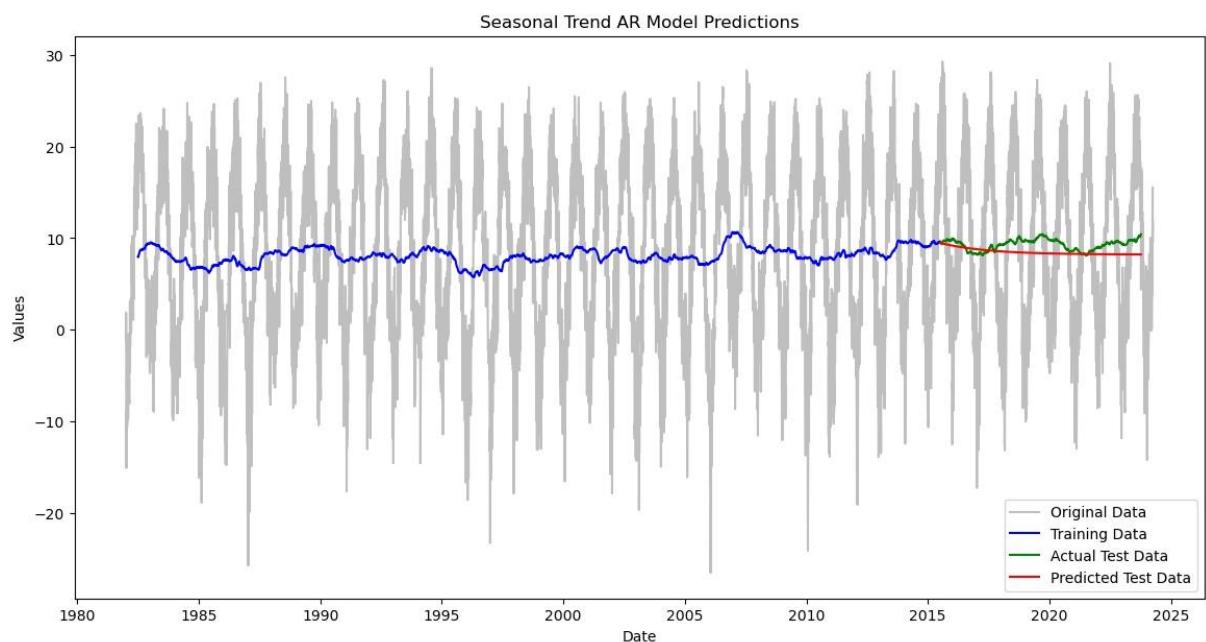


Mean Squared Error: 1.5564922490750737

Root Mean Squared Error: 1.2475945852219277

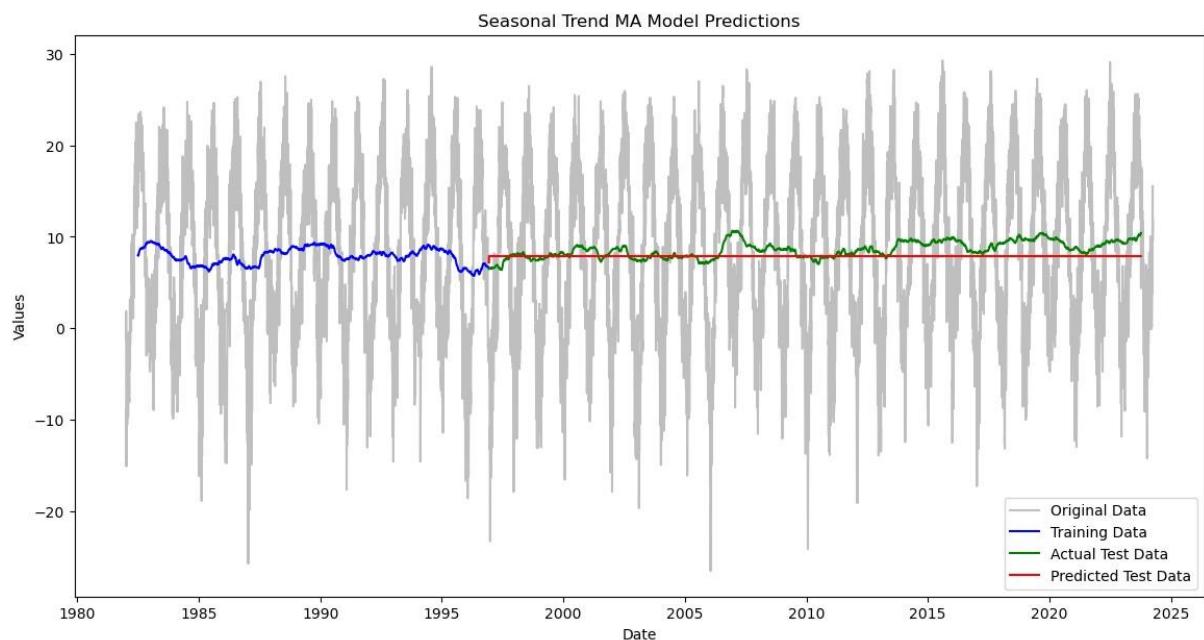


Mean Squared Error: 1.3363544926427737
Root Mean Squared Error: 1.1560079985202412



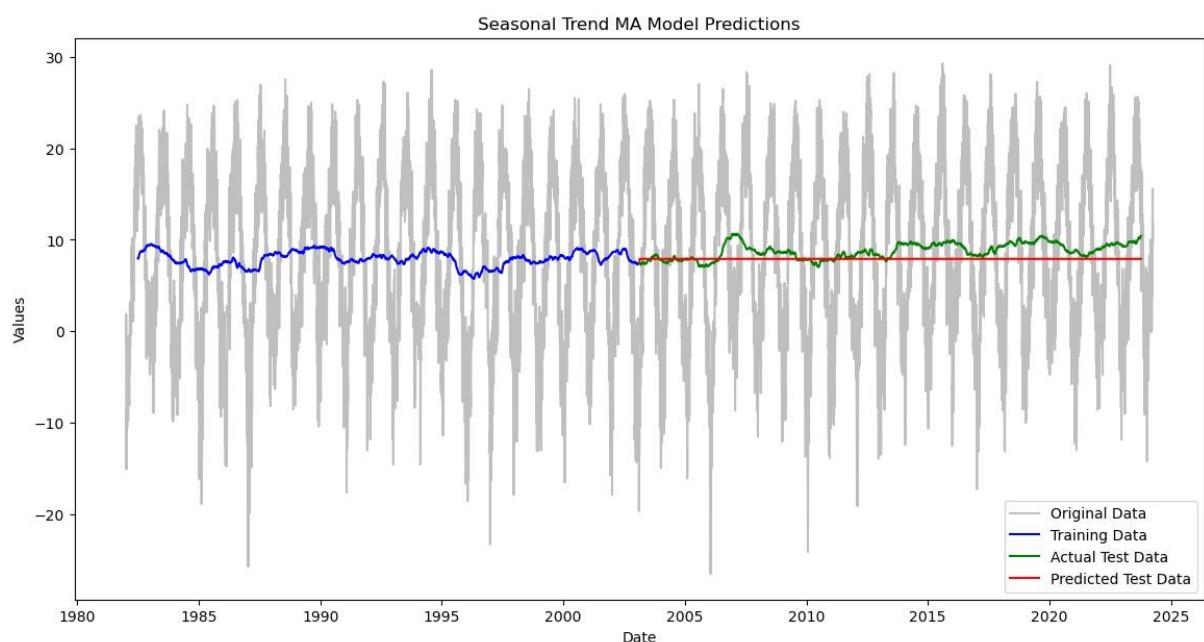
Mean Squared Error: 1.089048900292324
Root Mean Squared Error: 1.0435750573352758

Wyniki model MA



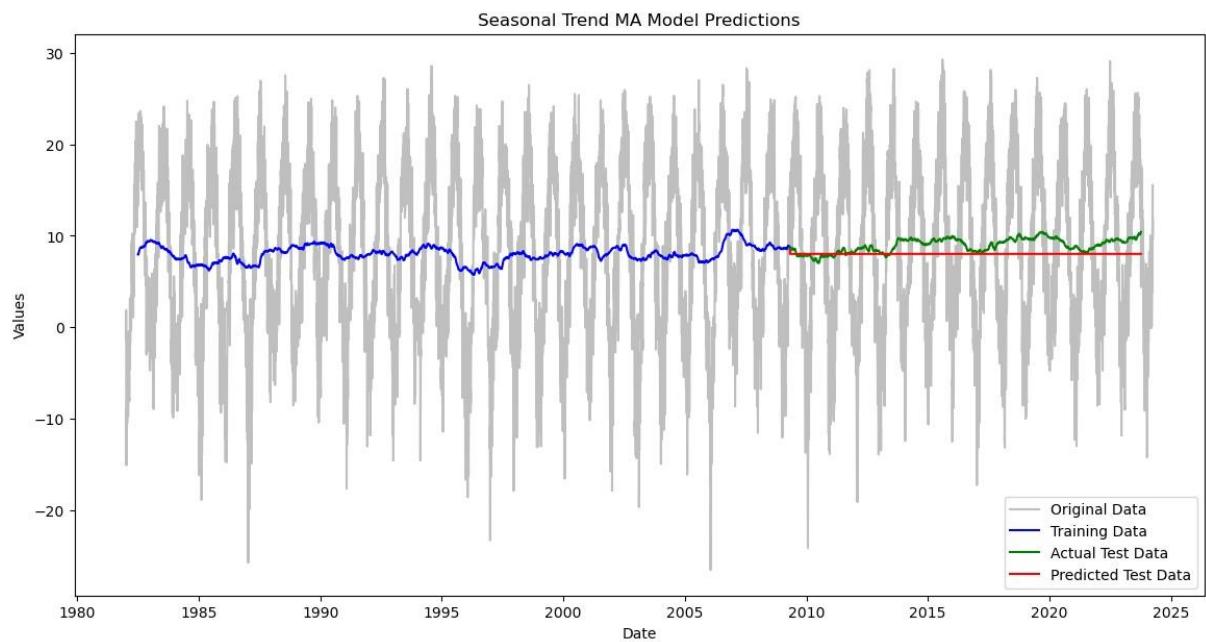
Mean Squared Error: 1.2663095435693852

Root Mean Squared Error: 1.1253042004584295



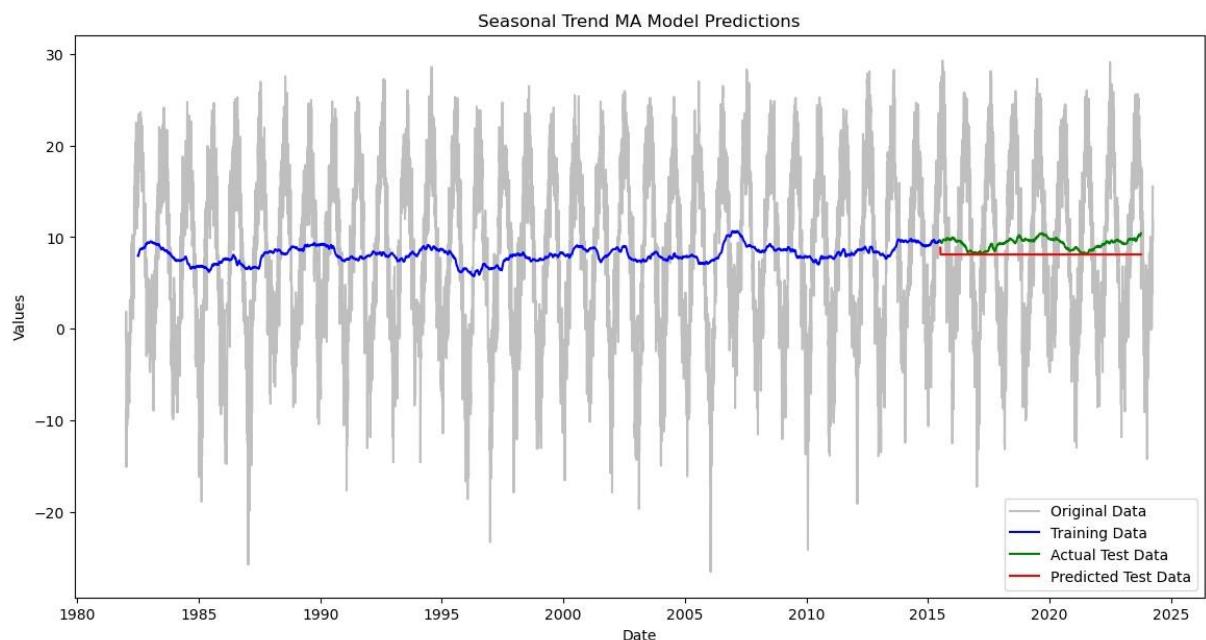
Mean Squared Error: 1.4783875336266152

Root Mean Squared Error: 1.2158896058551596



Mean Squared Error: 1.4330465100208665

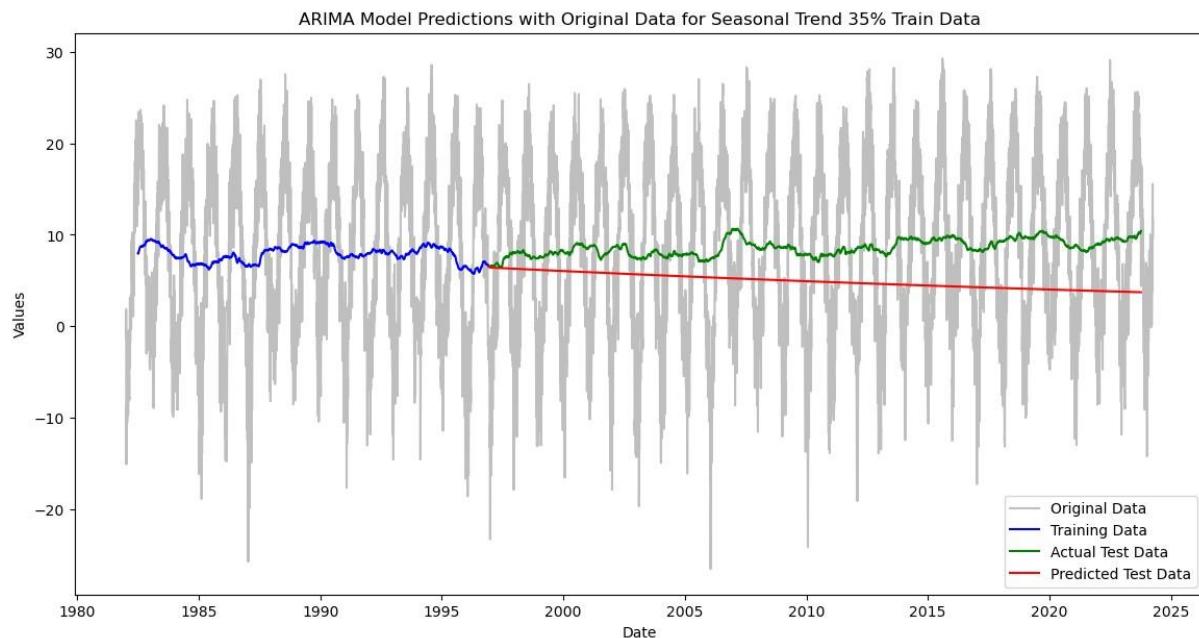
Root Mean Squared Error: 1.1970992064239565



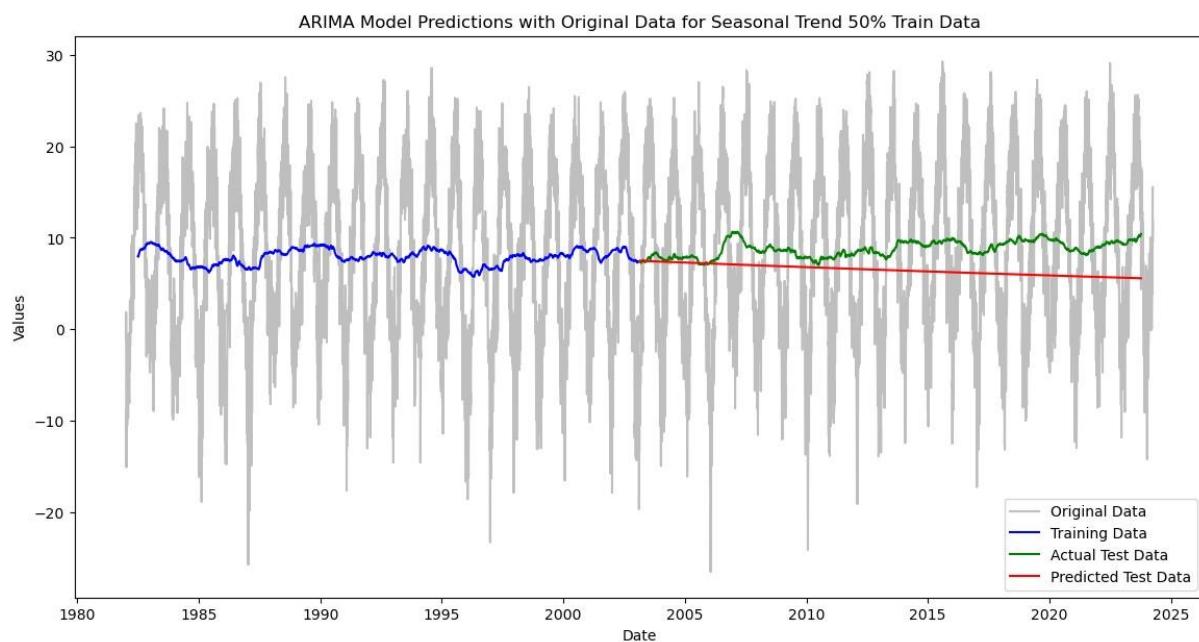
Mean Squared Error: 1.728645707646718

Root Mean Squared Error: 1.3147797182976007

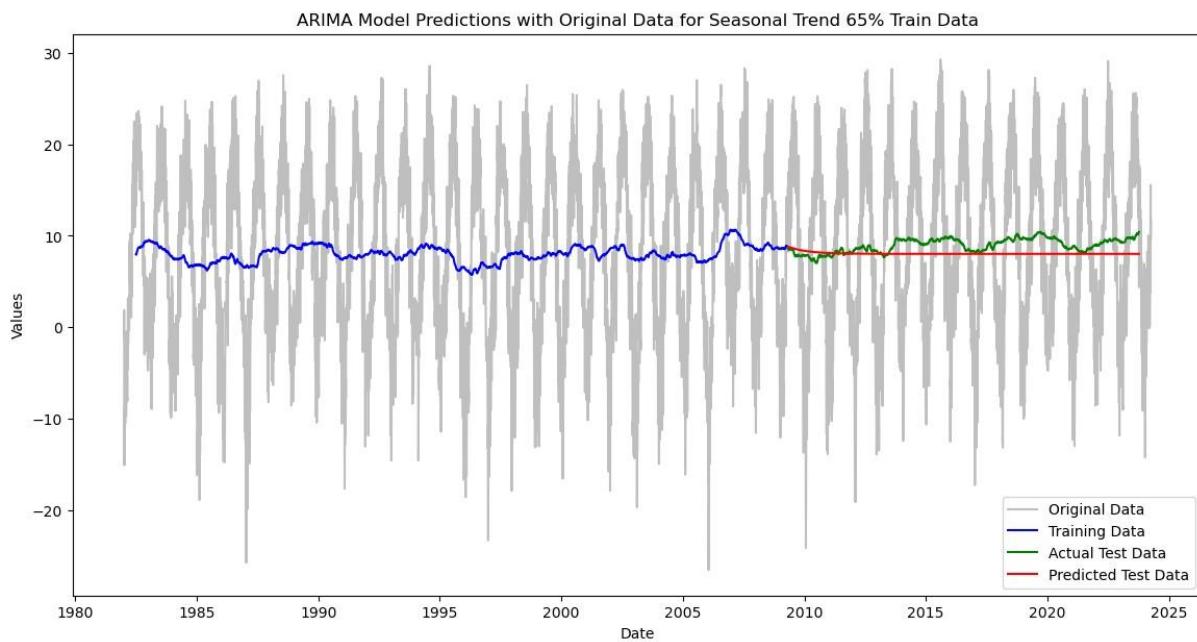
Wyniki model ARIMA



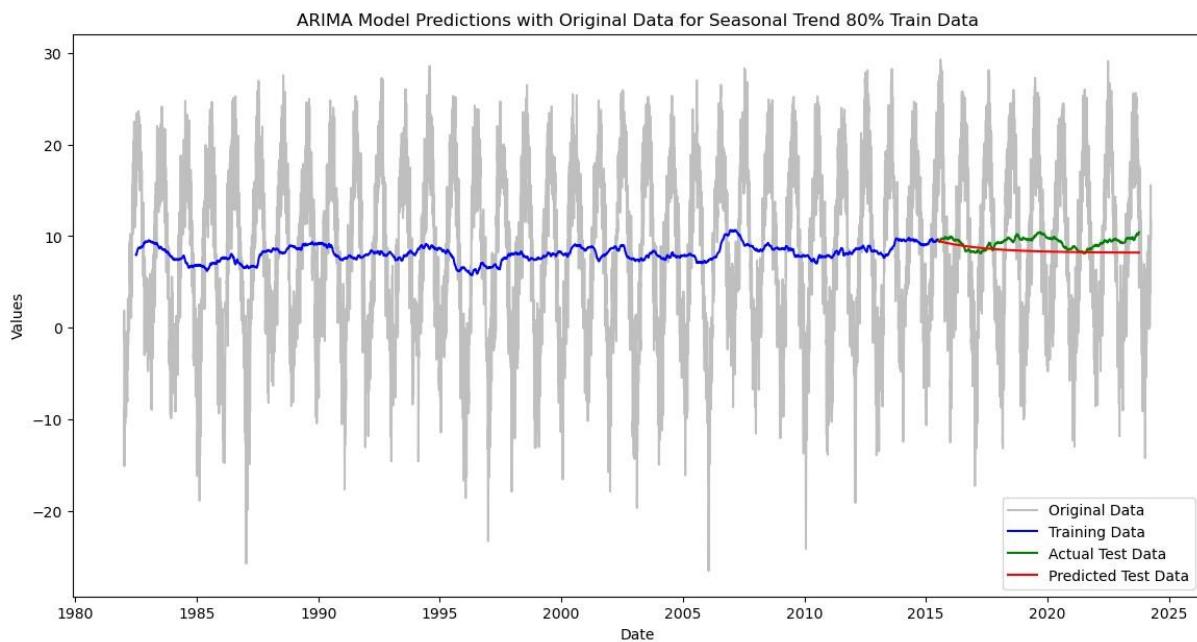
Mean Squared Error: 15.493686846872805
Root Mean Squared Error: 3.936202084099952



Mean Squared Error: 6.823677005757853
Root Mean Squared Error: 2.6122168757126296



Mean Squared Error: 1.4461889786382784
 Root Mean Squared Error: 1.2025759762436128



Mean Squared Error: 1.1103304383166896
 Root Mean Squared Error: 1.0537221827012515

Interpretacja wyników

Dane z dekompozycji sezonowej okazały się trudniejsze do przewidzenia od danych z regresji liniowej, dlatego że są znacznie bardziej złożone. Należy też pamiętać że dekompozycja sezonowa posiada resztki szumu, które pogarszają jakość naszych danych. W modelu ARIMA duże znaczenie miały proporcje train/test. Model MA jak zwykle miał problemy z danymi z dużą autokorelacją, co

skutkowało mało dokładną predykcją. Ogólnie wynikom brakowało dokładności w przewidywaniach i nie były one w stanie przewidzieć fluktuacji sezonowych.