Dataset link : https://www.kaggle.com/datasets/agajorte/detroit-daily-temperatures-with-artificial-warming

Data Cleaning, Preprocessing and Exploratory Data Analysis

```
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
from numpy import sqrt
import warnings

# Suppress warnings from statsmodels
warnings.filterwarnings("ignore")

# Function to parse date
def parser(x):
    return datetime.strptime(x, '%Y-%m-%d')

# Load the dataset
file_path = '/content/weather-complete.csv'
series = pd.read_csv(file_path, header=0, parse_dates=[0], index_col=0, date_parser=parser)

# Exploratory Data Analysis
print("Data Information:")
print(series.info())
print("\nFirst Five Rows of the Dataset:")
print(series.head())
print("\nStatistical Summary:")
print(series.describe())

# Check for missing values
print("\nChecking for Missing Values:")
print(series.isnull().sum())

# Remove missing values
series.dropna(inplace=True)

# Confirm missing values are removed
print("\nChecking for Missing Values After Removal:")
print(series.isnull().sum())

# Ensure the data has a daily frequency
series = series.asfreq('D')

# Decompose the time series using additive model
result = seasonal_decompose(series.interpolate(method='linear'), model='additive', period=365)
series_trend = result.trend.dropna()

# Plot the smoothed data (trend component)
plt.figure(figsize=(12, 6))
series_trend.plot()
plt.title("Trend Component (Smoothed Data)")
plt.xlabel("Date")
plt.ylabel("Temperature")
plt.show()

# Use trend component for ARIMA model
smoothed_series = series_trend

# Split data into training and test sets
train_data = smoothed_series[:'2016-12-31']
test_data = smoothed_series['2017-01-01':]
```

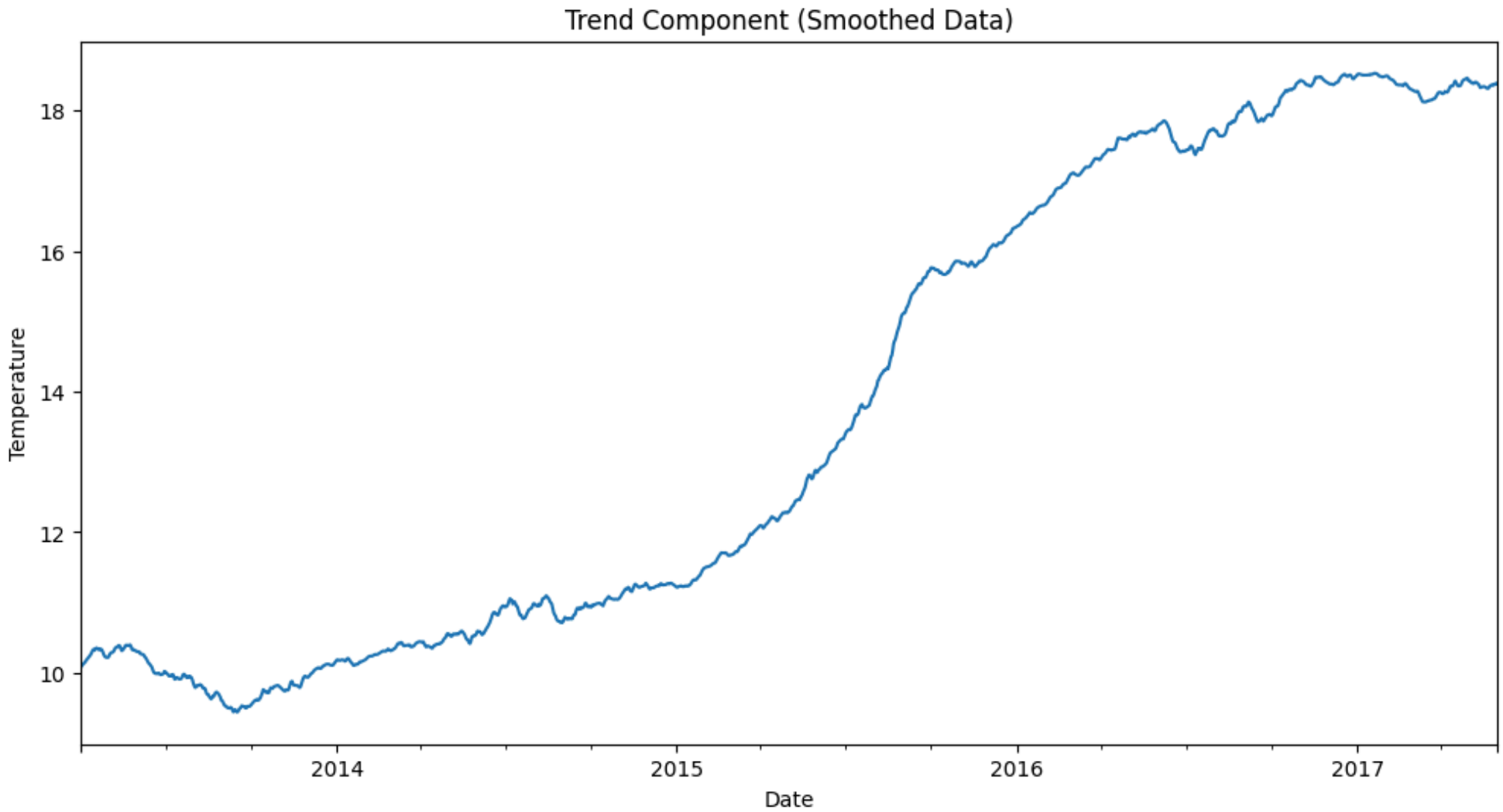
```
Data Information:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1886 entries, 2012-10-01 to 2017-11-29
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   temperature  1885 non-null   float64
dtypes: float64(1)
memory usage: 29.5 KB
None

First Five Rows of the Dataset:
            temperature
date                   
2012-10-01    11.036840
2012-10-02    14.340558
2012-10-03    14.518382
2012-10-04    16.820351
2012-10-05    16.948431

Statistical Summary:
       temperature
count  1885.000000
mean     13.782933
std      11.418935
min     -20.568680
25%       5.289972
50%      14.602483
75%      23.146501
max      35.738109

Checking for Missing Values:
temperature    1
dtype: int64

Checking for Missing Values After Removal:
temperature    0
dtype: int64
```



```
# Function to evaluate ARIMA model
def evaluate_arima_model(train, test, arima_order):
    model = ARIMA(train, order=arima_order)
    results = model.fit()
    start = len(train)
    end = start + len(test) - 1
    predictions = results.predict(start=start, end=end, dynamic=False)
    mse = mean_squared_error(test, predictions)
    return mse, predictions

# Function to evaluate multiple ARIMA models
def evaluate_models(train, test, p_values, d_values, q_values):
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p, d, q)
                try:
                    mse, _ = evaluate_arima_model(train, test, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.3f' % (order, mse))
                except:
                    continue
    print('Best ARIMA=%s MSE=%.3f' % (best_cfg, best_score))
    return best_cfg
```

```
# Parameter ranges for ARIMA model
p_values = range(0, 10)
d_values = range(0, 3)
q_values = range(0, 3)
warnings.filterwarnings("ignore")
best_cfg = evaluate_models(train_data, test_data, p_values, d_values, q_values)
```

```
ARIMA(3, 1, 2) MSE=0.043
ARIMA(3, 2, 0) MSE=1.495
ARIMA(3, 2, 1) MSE=0.274
ARIMA(3, 2, 2) MSE=0.229
ARIMA(4, 0, 0) MSE=0.017
ARIMA(4, 0, 1) MSE=0.034
ARIMA(4, 0, 2) MSE=0.028
ARIMA(4, 1, 0) MSE=0.043
ARIMA(4, 1, 1) MSE=0.043
ARIMA(4, 1, 2) MSE=0.042
ARIMA(4, 2, 0) MSE=1.074
ARIMA(4, 2, 1) MSE=0.912
ARIMA(4, 2, 2) MSE=0.225
ARIMA(5, 0, 0) MSE=0.011
ARIMA(5, 0, 1) MSE=0.694
ARIMA(5, 0, 2) MSE=0.012
ARIMA(5, 1, 0) MSE=0.043
```

```
ARIMA(9, 0, 0) MSE=0.394
ARIMA(9, 0, 2) MSE=0.404
ARIMA(9, 1, 0) MSE=0.043
ARIMA(9, 1, 1) MSE=0.043
ARIMA(9, 1, 2) MSE=0.042
ARIMA(9, 2, 0) MSE=0.296
ARIMA(9, 2, 1) MSE=0.281
ARIMA(9, 2, 2) MSE=0.283
Best ARIMA=(5, 0, 0) MSE=0.011
```

```
# Fit ARIMA model with the best configuration
model = ARIMA(train_data, order=best_cfg)
results = model.fit()

# Generate predictions
predictions = results.predict(start=len(train_data), end=len(smoothed_series)-1, dynamic=False)

last_date_index = smoothed_series.index[-1]

# Generate forecast for the next 20 days beyond the dataset timeline
forecast_period = 60
forecast_start_date = last_date_index + pd.Timedelta(days=1)  # Start forecasting from the day after the last date in the index
forecast_index = pd.date_range(start=forecast_start_date, periods=forecast_period, freq='D')
forecast = results.forecast(steps=forecast_period)
forecast.index = forecast_index

# Print the forecast
print("Forecast (steps = 60):")
print(forecast)

# Calculate MSE and RMSE
mse = mean_squared_error(test_data, predictions[:len(test_data)])
rmse = sqrt(mse)
print("MSE: ", mse)
print("RMSE: ", rmse)

print(results.summary())
```

```
Forecast (steps = 60):
2017-06-01    18.491203
2017-06-02    18.487299
2017-06-03    18.471813
2017-06-04    18.465702
2017-06-05    18.456530
2017-06-06    18.460952
2017-06-07    18.461359
2017-06-08    18.470572
2017-06-09    18.469934
2017-06-10    18.473800
2017-06-11    18.466599
2017-06-12    18.465626
2017-06-13    18.457023
2017-06-14    18.457979
2017-06-15    18.453217
2017-06-16    18.457931
2017-06-17    18.455321
2017-06-18    18.459968
2017-06-19    18.455553
2017-06-20    18.457718
2017-06-21    18.451336
2017-06-22    18.452743
2017-06-23    18.446922
2017-06-24    18.449693
2017-06-25    18.445344
2017-06-26    18.449034
2017-06-27    18.444802
2017-06-28    18.447886
2017-06-29    18.442743
2017-06-30    18.445024
2017-07-01    18.439526
2017-07-02    18.441936
2017-07-03    18.436930
2017-07-04    18.439893
2017-07-05    18.435287
2017-07-06    18.438332
2017-07-07    18.433545
2017-07-08    18.436246
2017-07-09    18.431152
2017-07-10    18.433679
2017-07-11    18.428616
2017-07-12    18.431301
2017-07-13    18.426461
2017-07-14    18.429300
2017-07-15    18.424521
2017-07-16    18.427296
```

```
2017-07-17    18.422404
2017-07-18    18.425047
2017-07-19    18.420090
2017-07-20    18.422723
2017-07-21    18.417831
2017-07-22    18.420536
2017-07-23    18.415718
2017-07-24    18.418442
2017-07-25    18.413618
2017-07-26    18.416291
2017-07-27    18.411427
```

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Define xlabel and ylabel
xlabel = 'Date'
ylabel = 'Temperature'

# Define the formatter
formatter = ticker.FuncFormatter(lambda x, _: f'{x:.2f}')

# Plotting
ax = smoothed_series.plot(legend=True, figsize=(12, 6), label='Trend Component')
predictions.index = test_data.index  # Align predictions with test data index
predictions.plot(legend=True, label='Predictions', ax=ax)
forecast.plot(legend=True, label='Forecast')

ax.axvline(x=test_data.index[0], color='gray')

ax.autoscale(axis='x', tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
ax.yaxis.set_major_formatter(formatter)
plt.legend()
plt.show()
```