**Time Series Forecasting using SARIMA (Python)**

Exploring Seasonal Autoregressive Integrated Moving Average models using Python and weather dataset

[[](https://medium.com/@ozdogar?source=post_page-----8db28f1d8cfc--------------------------------)](https://medium.com/@ozdogar?source=post_page-----8db28f1d8cfc--------------------------------)

[Can Ozdogar](https://medium.com/@ozdogar?source=post_page-----8db28f1d8cfc--------------------------------)

·

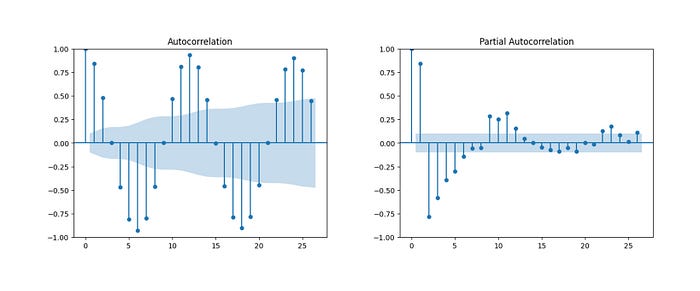
Follow

8 min read

·

Aug 11

68



ARMA models are widely used in time series forecasting. In compare to ARMA models, SARIMA models can be used even if the data is not stationary and there is a seasonality component.

There are not many Python implementation details of SARIMA models on the internet. On top of that, resources rarely compare prediction results with other methods. This article will not get into in-depth technical details of SARIMA models, but will be focusing on Python implementation, comparison and some shortcomings of SARIMA. We will be using Istanbul’s temperature data for this study.

You may be asking: does SARIMA model makes sense for temperature forecasting? The answer is yes. Autoregressive Models (AR) are suitable when data is regressing over its own lagged values. For weather, we can say it is true, this month’s temperature is somewhat dependent to previous month’s temperature. Additionally, I would expect current month’s average temperature to be affected by previous timesteps’ white noise error terms, so Moving Average(MA) addition to the model makes sense. There is an obvious seasonality, so a Seasonal ARMA model should be providing strong predictive power for monthly weather.

I would expect no meaningful trend in weather data over years, considering the dataset is only covering 32 years. However, in case some trend can be observed due to climate change or air pollution, using differencing can help us eliminate the trend (and help our stationarity assumption, which needs to hold for ARMA models). Therefore, I decided to use Seasonal ARIMA instead of Seasonal ARMA. We will be testing these assumptions during the implementation as well.

You may reach the **full Colab Notebook** that I created for this study at: <https://github.com/cnzdgr/Weather-Forecast/blob/main/Istanbul_Weather_Forecast_Using_SARIMA.ipynb>

The dataset was purchased from “[www.visualcrossing.com](http://www.visualcrossing.com/)”, therefore cannot be shared. You may replicate the study on the Colab Notebook with other time series.

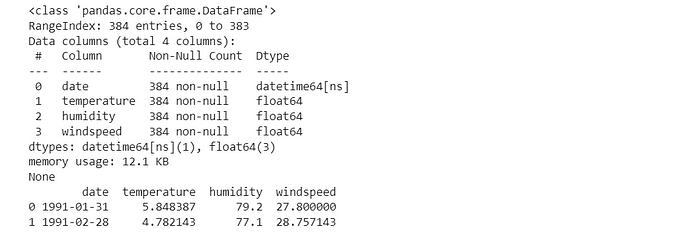
**Part 1: Data Preparation**

We need to check our weather data and convert data from daily into monthly.

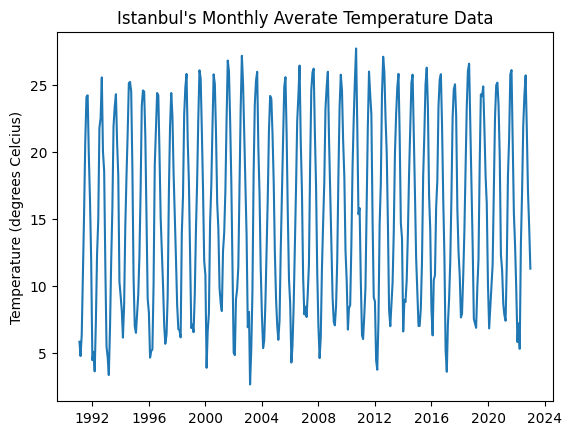
A valid question can be “why are we not using daily data as autocorrelation makes more sense in daily setting and daily weather prediction is more useful”. This is one of the shortcomings of SARIMA models, especially in Python. Using daily data means setting seasonality parameter to 365 and it inflates both RAM requirement to unfeasible levels (above 100GB level) and model fitting becomes extremely slow (each training takes more than 1 hour) in our case, where we have 28 years of data for training.

There are other people out there having the same problem. One example is: <https://datascience.stackexchange.com/questions/90327/python-sarimax-model-fits-too-slow>

That is why lecturers are using monthly or quarterly data while teaching SARIMA models and it really hurts the usability.

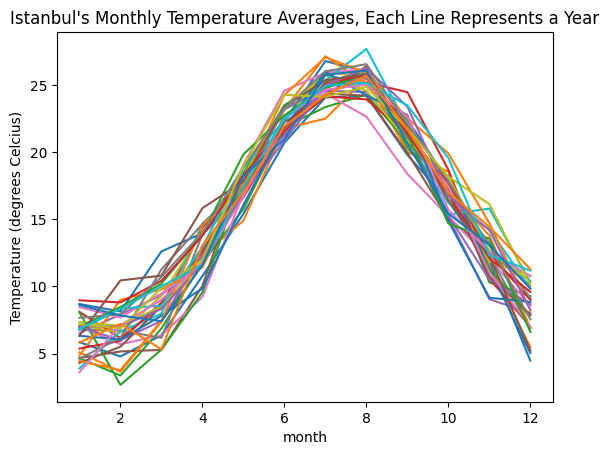


Dataset, after grouping monthly



Dataset — visualized

We can see the clear pattern above. Please note that we will only be using average temperature data (exclude humidity and wind speed) as ARMA model family (inc. SARIMA) are for single variable problems.



Each month has around 5 °C interval that observations fall into

**Part 2: Predicting with a Simple Model**

We need a benchmark to compare our SARIMA model’s performance. For this purpose, there are 2 alternatives:  
1. Compare with the naive estimation’s error. Naive estimation means: I would predict this month’s average temperature for the next month. If this month is January and the average temperature is 5 °C, I would predict 5 °C for February.  
2. Make a mean temperature prediction by using ‘month of the year’ data and find coefficients by fitting a linear model.

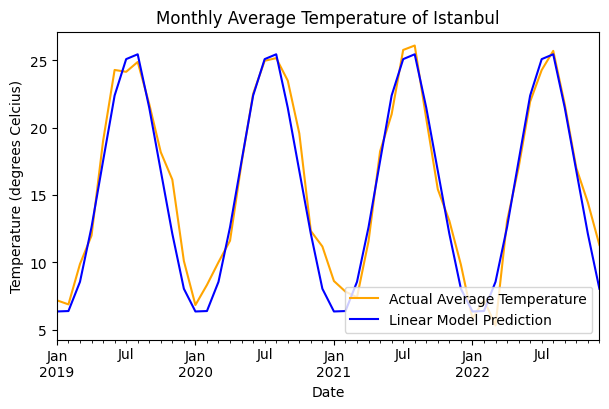
I will be using the second method. First, I will one-hot-encode the month variable. Then feed it to a linear regression and take the result as the benchmark.

Data from 1991 to 2018 will be used for training and data from 2019 to 2022 for testing. Dataset contains all days from 1-Jan-91 to 31-Dec-22.

#Taking month data from the datetime object  
weather\_linear['month'] = pd.DatetimeIndex(weather\_linear['date']).month  
  
#One-hot-encoding of month data  
weather\_linear = pd.get\_dummies(weather\_linear, columns = ['month'])  
  
#Fitting a linear regression to data from 1991 to 2018, using 2019 to 2022 as test set  
train\_data = weather\_linear[weather\_linear['date'] < '2019-01-01']  
test\_data = weather\_linear[weather\_linear['date'] >= '2019-01-01']  
regression = LinearRegression().fit(train\_data.iloc[:,-12:], train\_data['temperature'])  
  
weather\_pred = regression.predict(test\_data.iloc[:,-12:])  
test\_data['prediction'] = weather\_pred  
print("Mean squared error: %.3f" % mean\_squared\_error(test\_data['temperature'], weather\_pred))  
  
print("Coefficient of determination: %.3f" % r2\_score(test\_data['temperature'], weather\_pred))

Mean squared error: 2.292  
**Coefficient of determination: 0.947**

The result is great. The simple linear model that only knows the month of the year can predict with high accuracy and can explain 94.7% of the variance. The fit (you may see below) seems accurate. This will be our benchmark.



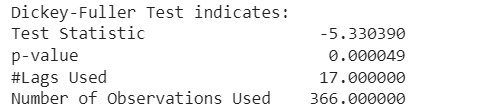
Linear model fit for the test data

**Part 3: Predicting with SARIMA**

In theory, we can implement SARIMA directly. However, the best approach consists of 4 steps.  
1. Check if the time series is stationary  
2. Check for autocorrelation  
3. Check for partial autocorrelation  
4. Decide on 7 parameters of SARIMA (p, d, q), (P,D,Q,m) by getting help from the first 3 steps

1. For stationary assumption, statsmodels has Augmented Dickey-Fuller implementation:

def ad\_fuller(timeseries):  
 print ('Dickey-Fuller Test indicates:')  
 df\_test = adfuller(timeseries, regression='ct', autolag='AIC')  
 output = pd.Series(df\_test[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])  
 print(output)  
  
print(ad\_fuller(weather['temperature']))

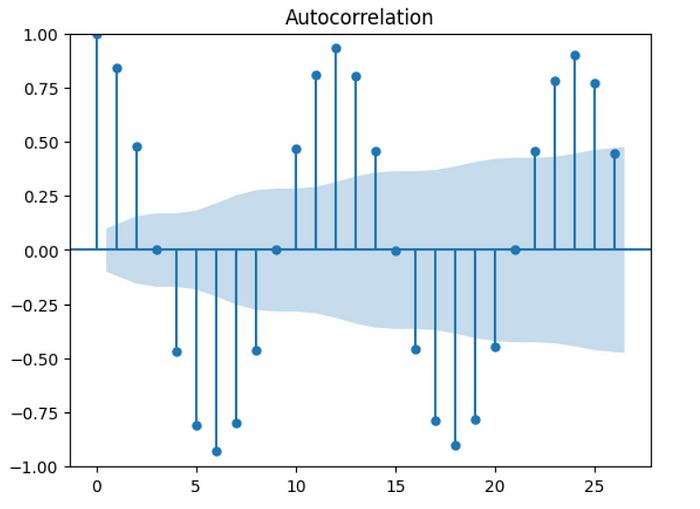


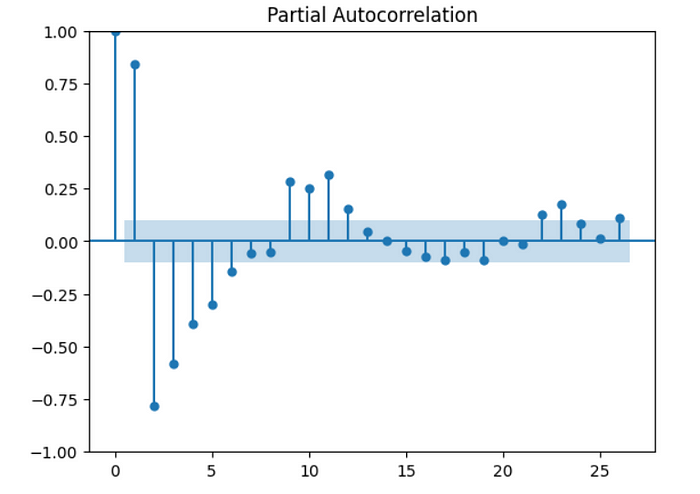
Time series is stationary

Our test has extremely small p-value, indicating that our time series is stationary.

2. and 3. For autocorrelation and partial autocorrelation, statsmodel has implementation as well: plot\_acf, plot\_pacf

plot\_acf(weather['temperature'])  
plt.show()  
plot\_pacf(weather['temperature'])  
plt.show()





We are seeing meaningful autocorrelation and partial autocorrelation.

4. At this step, we need to decide on model parameters (p,d,q)x(P,D,Q,m). It requires both inference from previous steps and some trial & error.

As Dickey-Fuller test indicated that the data is stationary, we can safely set d=0 (differencing) in our model. For D (seasonal counterpart of d), I will be using D=1 just because the model fits slightly better in that case compared to D=0.

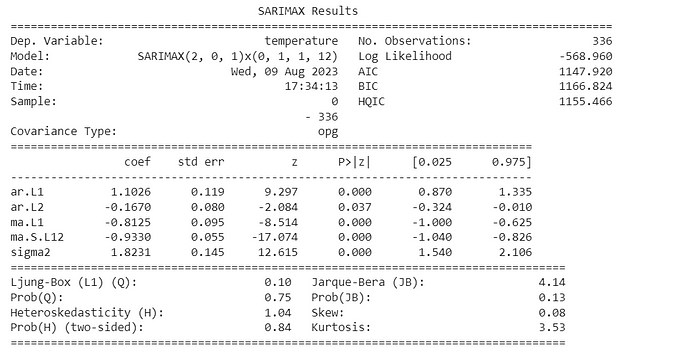
MA(q) part of the model predicts by taking a weighted average of past errors and we need to check **autocorrelation chart** to decide the q value. Looking at the high spikes, the chart suggests a value that is less than or equal to 3 (first 3 values are meaningfully large). Trying both 1, 2 and 3; we get the best result at q=1. As the seasonal part, Q=1 also works well.

We need to check for large spikes from **partial autocorrelation chart** to decide AR(p). This time the first 5 to 6 spikes are meaningfully large. Trying all values from 1 to 6; we get the best result at p=2. As long-term autoregression does not make much sense, I’ll set P to 0 without checking.

Seasonality is 12 in our monthly data.

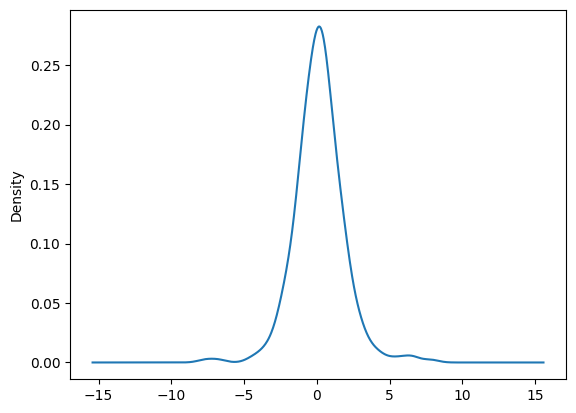
Please note that we got some help from our first 3 steps during deciding on the upper bound for p, d, q parameters. Deciding the exact values for them and their seasonal counterparts (P, D, Q) requires trial & error. You may aim for lower AIC (or BIC) or simply look at p-values of each parameter and be sure that each parameter is meaningfully away from 0.

model = sm.tsa.statespace.SARIMAX(train\_data\_SARIMA['temperature'],  
 order=(2, 0, 1),  
 seasonal\_order=(0, 1, 1, 12))  
result = model.fit()  
print(result.summary())



By both getting help from our analysis and some trial and error, we have a good fit with SARIMA(2,0,1)x(0,1,1,12). Now we will check its accuracy.

But before, it is always good to check for residual distribution to be sure that the fit is good.

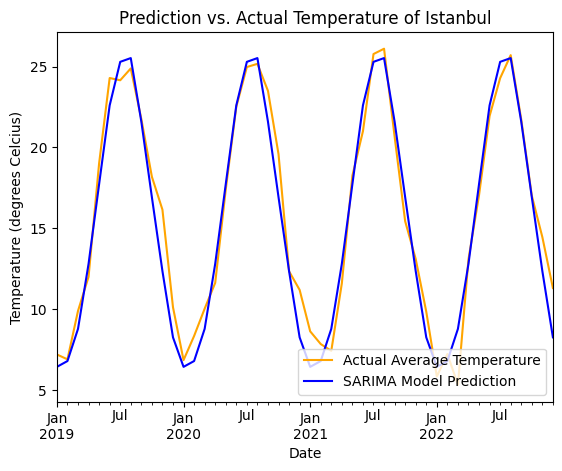


Distribution of residuals

We are seeing a good bell-shaped distribution of residuals, indicating no problems in our model fit.

Now it is time to make one-step-ahead predictions with our model. It is computationally expensive because we need to train our model over and over at each timestep using the new datapoint that the model just saw.

#Training the model again after each prediction, making one-step-ahead prediction each time  
one\_step\_predictions = []  
for i in range(48):  
 cut\_point = weather.size - 24 + i  
 model = sm.tsa.statespace.SARIMAX(train\_data\_SARIMA['temperature'][:cut\_point],  
 order=(2, 0, 1),  
 seasonal\_order=(0, 1, 1, 12),  
 enforce\_stationarity=False,  
 enforce\_invertibility=False)  
 result = model.fit()  
 one\_step\_predictions.append(result.predict(cut\_point).values[0])  
  
test\_data\_SARIMA['prediction'] = one\_step\_predictions  
test\_data\_SARIMA.set\_index('date', inplace=True)  
test\_data\_SARIMA['temperature'].plot(label='Actual Average Temperature', color='orange')  
test\_data\_SARIMA['prediction'].plot(label='SARIMA Model Prediction', color='blue')  
plt.title("Prediction vs. Actual Temperature of Istanbul")  
plt.xlabel('Date')  
plt.ylabel('Temperature (degrees Celcius)')  
  
plt.legend(loc='lower right')



Linear model fit for the test data

**Coefficient of determination: 0.952**

Our benchmark R2 was 0.947 and SARIMA achieved 0.952. It means our new model explained 0.48% more variance in temperature data. Not bad, but not very satisfactory either.

**Conclusion**

SARIMA model slightly outperformed the simple linear regression in monthly weather prediction task. It was better, yet not satisfactory and the possible reason could be: autoregressive analysis can be useful for daily data, but monthly average is extremely dependent on the given month and SARIMA model is not able to meaningfully beat it. The study showed 2 shortcomings of a SARIMA model:

1. Even though a SARIMA model might be working fine with daily data, using 365 as seasonality pushes the computing time to a non-feasible space in Python. R implementation seems more efficient. That is the possible reason that nearly all examples on the internet uses monthly or quarterly data.
2. SARIMA model is good at 1-step-ahead predictions as value at t=10 is dependent to value at t=9. That means we need to fit the model again and again for each prediction using all data before that point. It makes the process computationally expensive.