The ML Classroom

Time Series Analysis and Forecasting with ARIMA in Python

Divyesh Bhatt Follow 4 min read · Nov 3, 2023

Time series forecasting is a crucial area of machine learning that predicts future points in a series based on past data. It is especially common in economics, weather forecasting, and capacity planning, but it is applicable in many other fields. Today, we'll walk through an example of time series analysis and forecasting using the ARIMA model in Python.

Understanding ARIMA

ARIMA stands for AutoRegressive Integrated Moving Average. It is a class of models that captures a suite of different standard temporal structures in time series data. Before we apply ARIMA, we need to ensure that the series is stationary, which involves checking if its statistical properties such as mean, variance, and autocorrelation are constant over time.

The Data

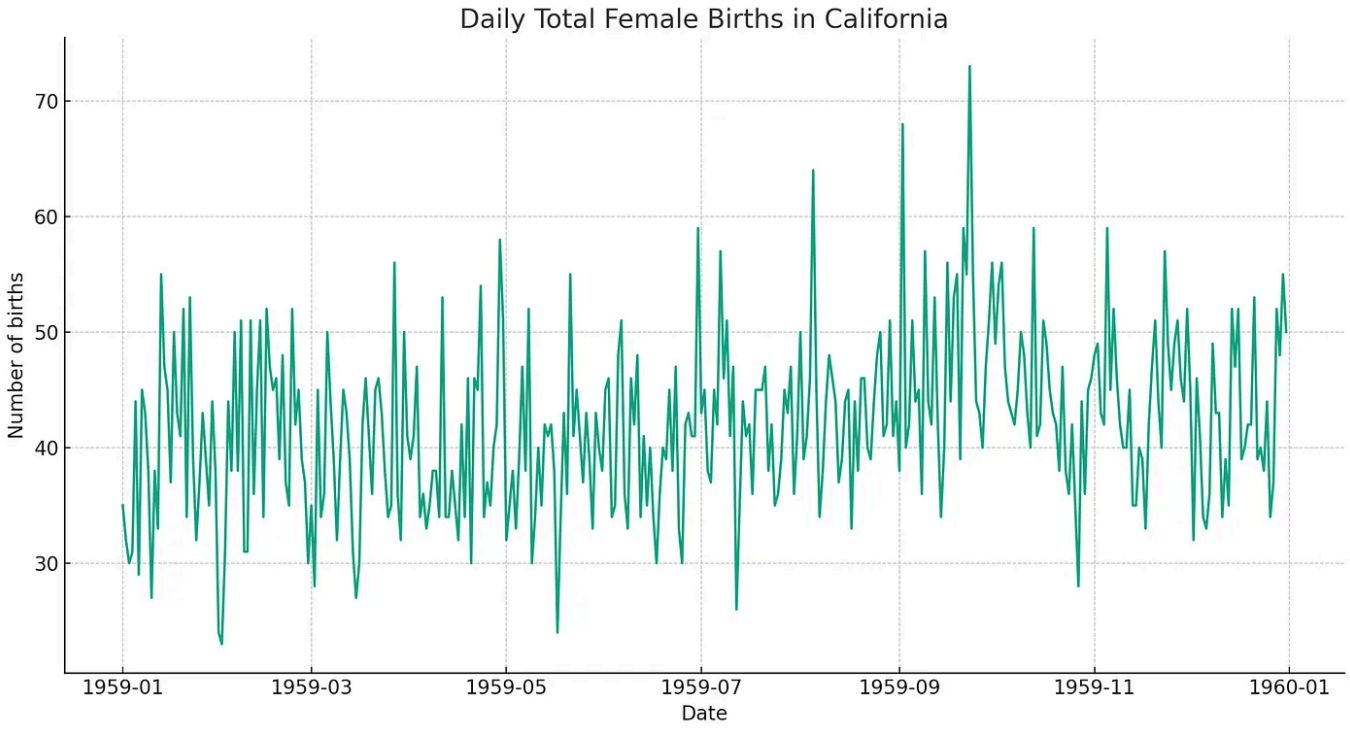
We are using a dataset that represents the daily total female births in California in 1959, which is commonly used as an example in time series analysis.

Preliminary Analysis

We start by loading the data and plotting the time series:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('daily-total-female-births-CA.csv')
data['date'] = pd.to_datetime(data['date'])
data.set_index('date', inplace=True)
# Plot the time series
plt.plot(data['births'])
plt.title('Daily Total Female Births in California')
plt.xlabel('Date')
plt.ylabel('Number of births')
plt.show()
```

The plot helps us visualize the data and provides initial insights into its properties, like trends and seasonality.



The plot helps us visualize the data and provides initial insights into its properties, like trends and seasonality.

Testing for Stationarity

Next, we use the Augmented Dickey-Fuller (ADF) test to check for stationarity:

```
from statsmodels.tsa.stattools import adfuller

adf_test = adfuller(data['births'])
# Output the results
print('ADF Statistic: %f' % adf_test[0])
print('p-value: %f' % adf_test[1])
```

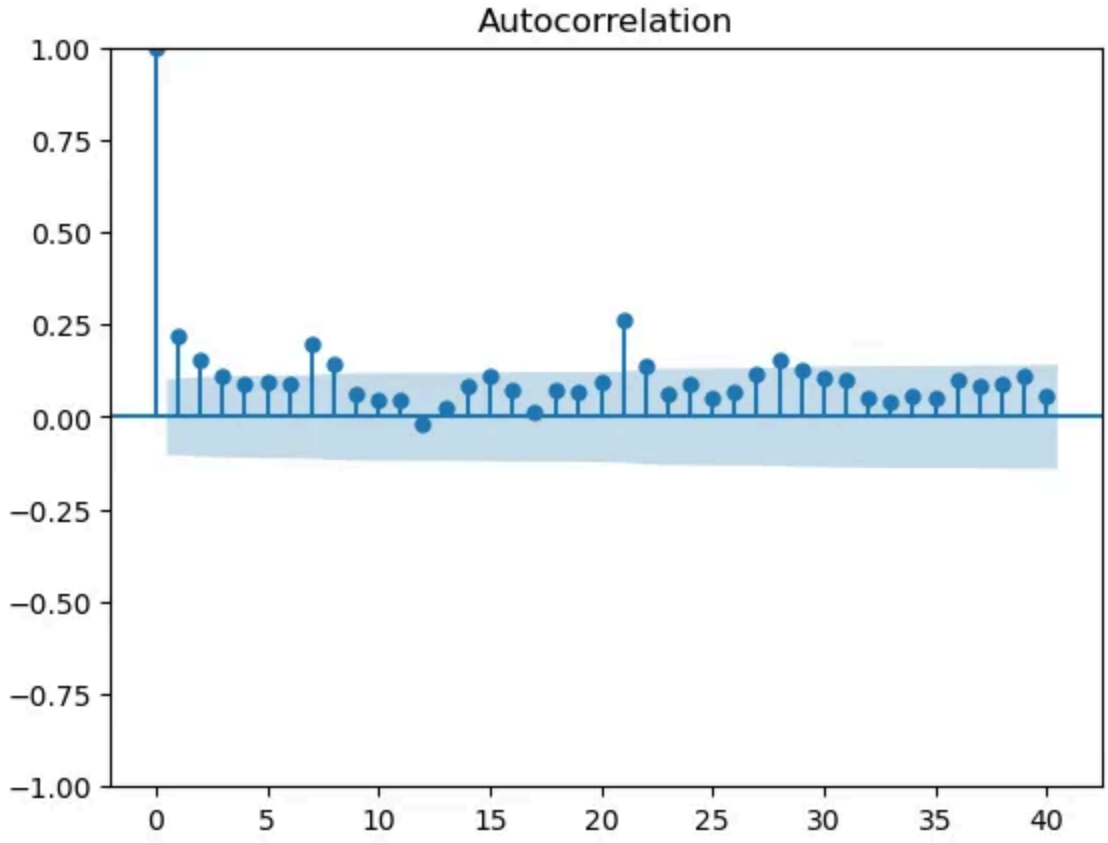
```
ADF Statistic: -4.808291
p-value: 0.000052
```

A p-value below 0.05 indicates stationarity, and our data meets this criterion, so we do not need to difference it.

Finding ARIMA Parameters

We use the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots to find the ARIMA parameters.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(data['births'], lags=40)
plot_pacf(data['births'], lags=40)
plt.show()
```



These plots suggest that an ARIMA(1, 0, 1) model may be a good starting point.

Building the ARIMA Model

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(data['births'], order=(1, 0, 1))
model_fit = model.fit()
```

Training and Forecasting

We train the model on the data and perform a forecast.

```
forecast = model_fit.get_forecast(steps=30)
```

Then we visualize the forecast alongside the historical data.

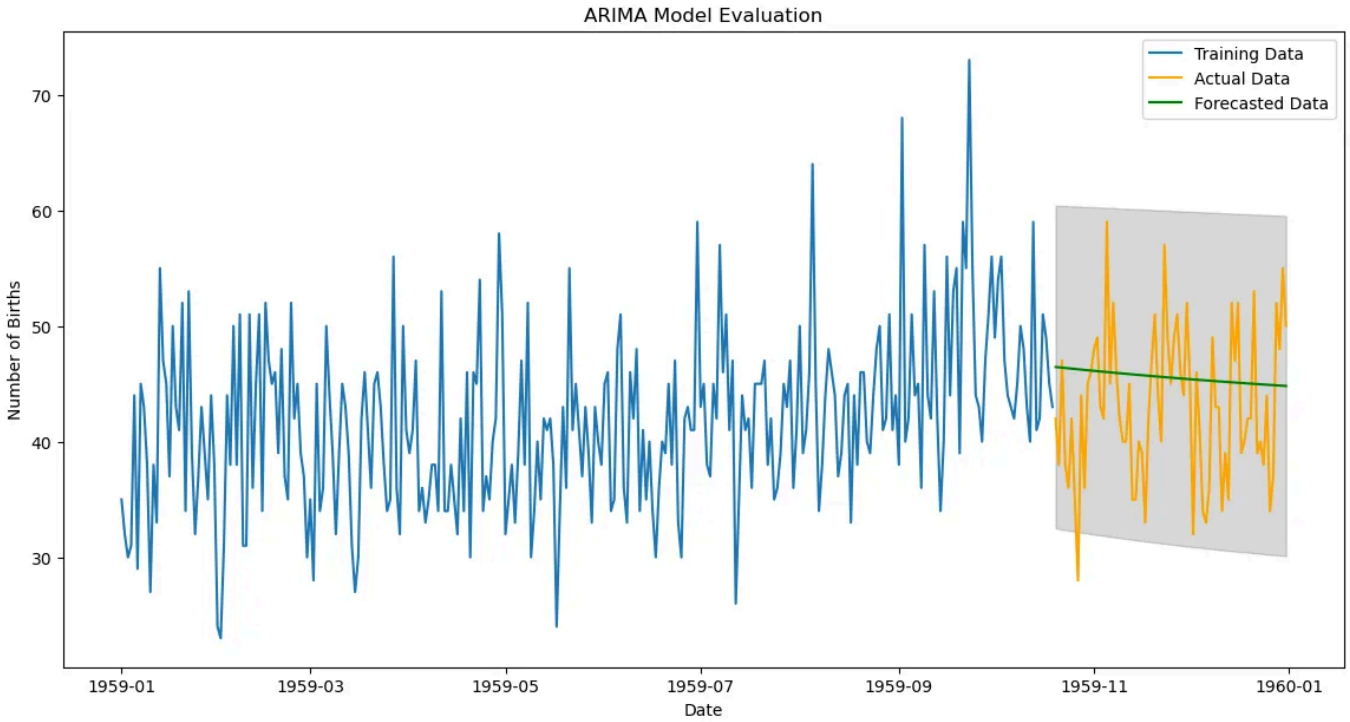
Model Evaluation

To assess the model, we perform a retrospective forecast.

```
from sklearn.metrics import mean_squared_error

# Split the data into train and test
train_size = int(len(data) * 0.8)
train, test = data[0:train_size], data[train_size:len(data)]

# Fit the ARIMA model on the training dataset
```

```
model_train = ARIMA(train['births'], order=(1, 0, 1))
model_train_fit = model_train.fit()

# Forecast on the test dataset
test_forecast = model_train_fit.get_forecast(steps=len(test))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test.index)

# Calculate the mean squared error
mse = mean_squared_error(test['births'], test_forecast_series)
rmse = mse**0.5

# Create a plot to compare the forecast with the actual test data
plt.figure(figsize=(14,7))
plt.plot(train['births'], label='Training Data')
plt.plot(test['births'], label='Actual Data', color='orange')
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.fill_between(test.index,
                 test_forecast.conf_int().iloc[:, 0],
                 test_forecast.conf_int().iloc[:, 1],
                 color='k', alpha=.15)
plt.title('ARIMA Model Evaluation')
plt.xlabel('Date')
plt.ylabel('Number of Births')
plt.legend()
plt.show()

print('RMSE:', rmse)
```

```
RMSE:  6.970853456222879
```



The retrospective forecast (backtest) compares the forecasted data against the actual data in the test set. The Root Mean Squared Error (RMSE) of the forecast is approximately 6.97 births.

The plot shows the training data, the actual values from the test set (in orange), and the forecasted values (in green), with the 95% confidence interval shown as the shaded area. The model appears to capture the central tendency of the series but does not capture any potential within-sample variability, which is not surprising given that ARIMA models are often more suited for data with trends or seasonality, which this series does not appear to exhibit.

The RMSE gives an indication of the forecast accuracy, with lower values indicating better fit. Whether this level of error is acceptable depends on the specific context in which the model is being used.

Conclusion

ARIMA models are a powerful tool for time series forecasting. By understanding the underlying patterns in the time series data and using statistical tests and plots to determine the model parameters, we can build a model that provides accurate and reliable forecasts.

This outline can be expanded upon with additional details and insights specific to the dataset and the results of the analysis.

Arima Model

Published in The ML Classroom

6 followers · Last published Mar 20, 2024

Follow

It is a capital mistake to theorize before one has data.

Written by Divyesh Bhatt

99 followers · 290 following

Follow

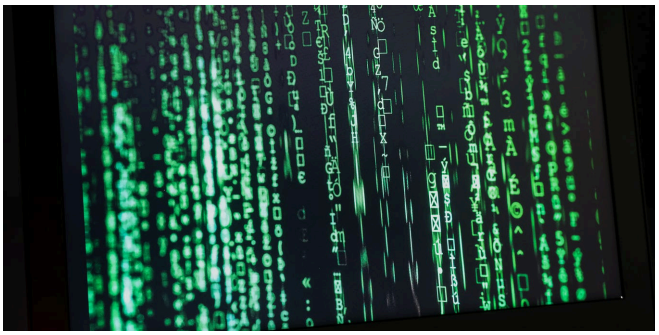
AI/ ML Practitioner

No responses yet

Write a response

What are your thoughts?

More from Divyesh Bhatt and The ML Classroom



Divyesh Bhatt

Understanding the Jacobian Matrix and Determinant in Machine...

The Jacobian matrix and determinant are fundamental mathematical concepts that pl...

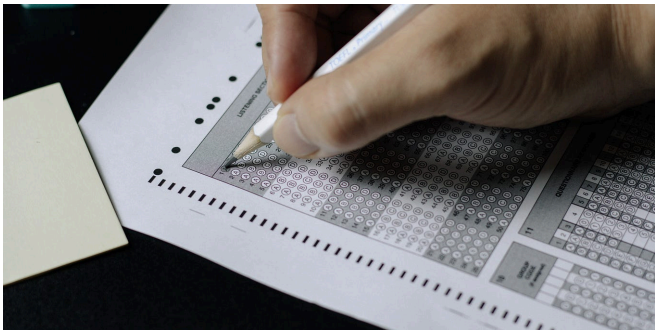
Jun 17, 2023 19

In The ML Classroom by Divyesh Bhatt

Probability Caliberation On Imbalanced Data

Probability calibration is a technique used in machine learning to adjust the predicted...

Mar 20, 2024 50



In The ML Classroom by Divyesh Bhatt

Classification : ROC vs PR Scores

As a machine learning practitioner, you may often come across the terms ROC and PR...

Mar 21, 2023

Divyesh Bhatt

Understanding Seasonality in Time Series Data

Time series data often reveals patterns when plotted over time. One of the most common...

Oct 4, 2023 1

See all from Divyesh Bhatt

See all from The ML Classroom

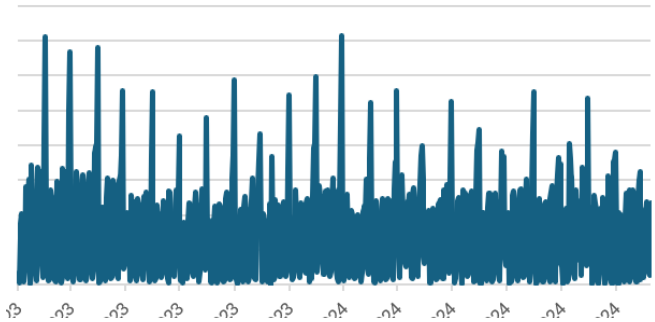
Recommended from Medium

Sarmita Majumdar

Lag & Rolling Features: Time Series Forecasting with Python

Find how to transform time series data into a supervised learning model format with lag...

Apr 14 100 1

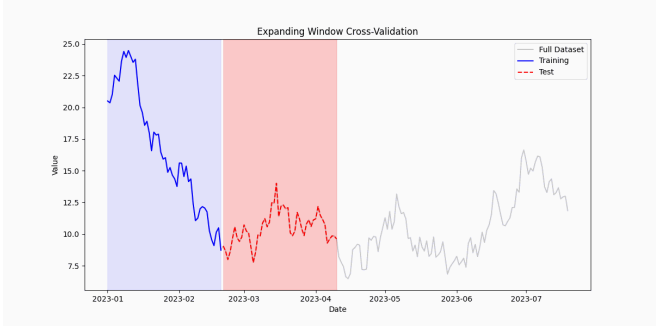


NandaKishore Joshi

Revenue Forecast With Prophet—With Seasonality, External...

Use multiple business regressors along with seasonality's to improve Revenue forecast...

Jan 14 18

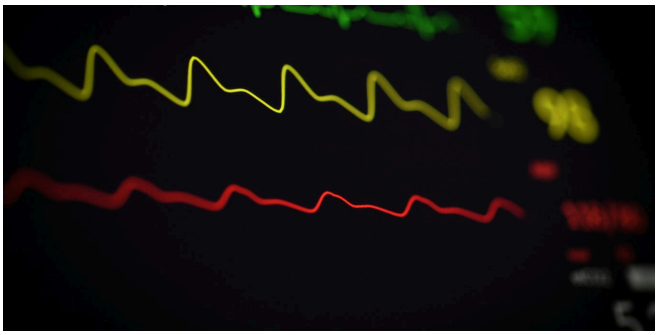


Kyle Jones

Cross-Validation for Time Series Data

Time series cross-validation differs fundamentally from standard cross-validatio...

Jan 17 91 2

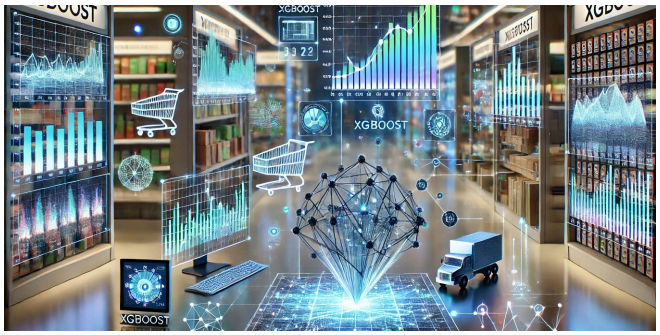


agus abdul rahman

Time Series Forecasting with ARIMA Models

A Step-by-Step Guide to Building Accurate Predictions for Your Data

Dec 8, 2024 50 1



Aditya singh

Leveraging XGBoost for Accurate Retail Time Series Predictions

One of the most common issues in retail today is the handling of time series problem...

Jan 2 30

Chris Yan

Dynamic Pricing Model for Logistics Services with Python...

This article explores how data science techniques, including machine learning...

Feb 20 5

See more recommendations