

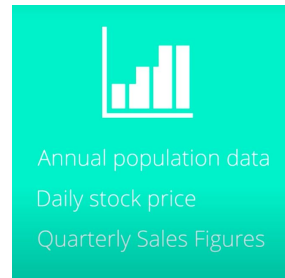
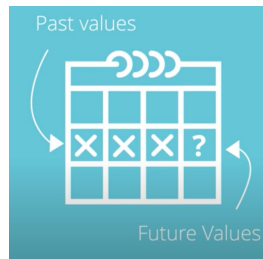
Time Series Forecasting

By Andrés Carvajal





What are the Time Series



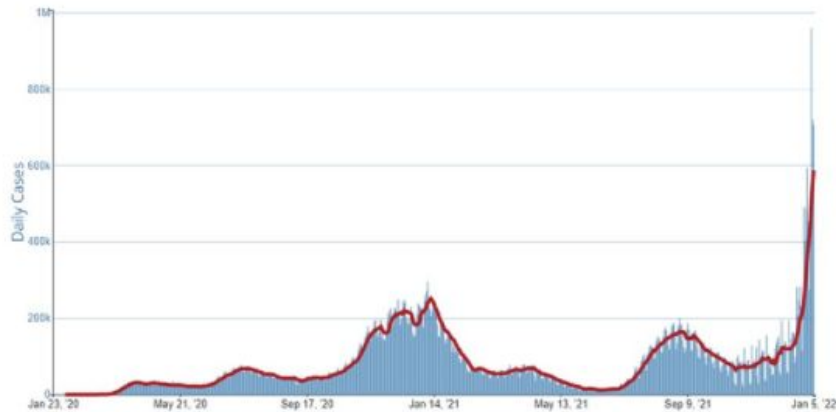
Time series data is data collected on the same subject at different points in time, such as GDP of a country by year, a stock price of a particular company over a period of time, or your own heartbeat recorded at each second.

Any data that you can capture continuously at different time-intervals is a form of time series data.

On the other hand, more conventional datasets which store information at a single point in time, such as information on customers, products, and companies, etc., are known as cross-sectional data.

The objectives are different when analyzing time-series and cross-sectional data, and a real-world dataset is likely to be a hybrid of both time-series as well as cross-sectional data.

Daily Trends in Number of COVID-19 Cases in the United States Reported to CDC



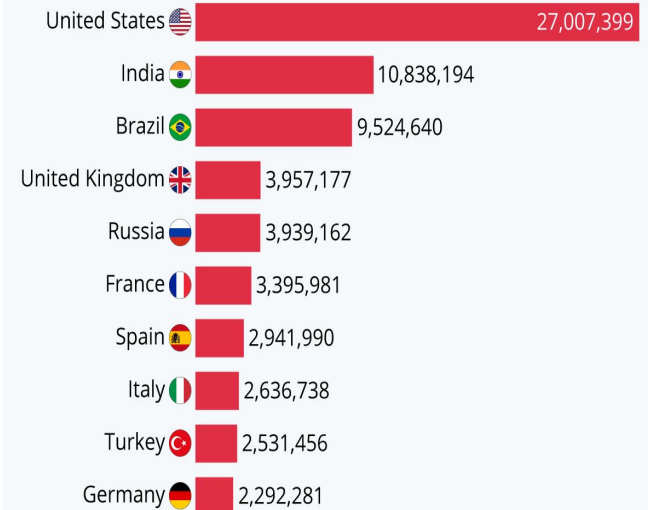
Find the latest data on
CDC's COVID Data Tracker

Data as of January 6, 2022
Source: COVID Data Tracker - Daily Trends



The Countries With The Most COVID-19 Cases

Total number of confirmed COVID-19 cases, by country*



* As of February 8, 2021 at 04:30 EST
Source: Johns Hopkins University

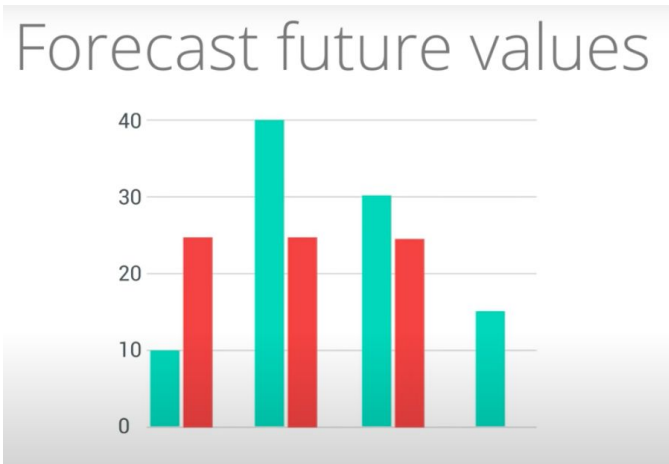
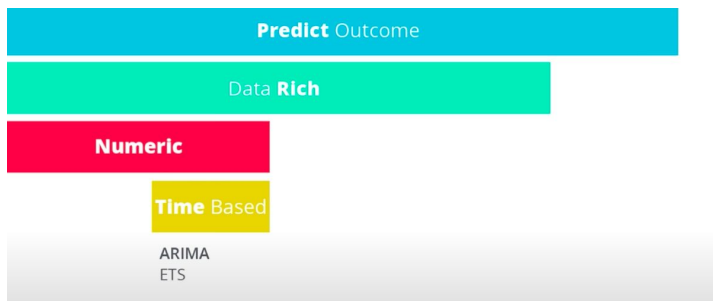


statista



What is the Time Series Forecasting?

Time series forecasting is exactly what it sounds like; predicting unknown values. Time series forecasting involves the collection of historical data, preparing it for algorithms to consume, and then predicting the future values based on patterns learned from the historical data.





Univariate Forecast

A univariate time series, as the name suggests, is a series with a single time-dependent variable. For example, if you are tracking hourly temperature values for a given region and want to forecast the future temperature using historical temperatures, this is univariate time series forecasting. Your data may look like this:

Time	Temperature
5:00 am	59 °F
6:00 am	59 °F
7:00 am	58 °F
8:00 am	58 °F
9:00 am	60 °F
10:00 am	62 °F
11:00 am	64 °F



Multivariate Forecast

On the other hand, a Multivariate time series has more than one time-dependent variable. Each variable depends not only on its past values but also has some dependency on other variables. This dependency is used for forecasting future values.

Time	Temperature	cloud cover	dew point	humidity	wind
5:00 am	59 °F	97%	51 °F	74%	8 mph SSE
6:00 am	59 °F	89%	51 °F	75%	8 mph SSE
7:00 am	58 °F	79%	51 °F	76%	7 mph SSE
8:00 am	58 °F	74%	51 °F	77%	7 mph S
9:00 am	60 °F	74%	51 °F	74%	7 mph S
10:00 am	62 °F	74%	52 °F	70%	8 mph S
11:00 am	64 °F	76%	52 °F	65%	8 mph SSW
12:00 pm	66 °F	80%	52 °F	60%	8 mph SSW

Univariate

Multivariate



Index	Variable 1	Target Variable
..
..

Index	Variable 1	Variable 2	Variable n	Target Variable
..
..

When we are dealing with multivariate time series forecasting, the input variables can be of two types:

- Exogenous: Input variables that are not influenced by other input variables and on which the output variable depends.
- Endogenous: Input variables that are influenced by other input variables and on which the output variable depends.

In situations like these, machine learning models come to the rescue as you can model any time series forecasting problem with regression.



Time Series Forecasting Methods

Time series forecasting can broadly be categorized into the following categories:

- Classical / Statistical Models — Moving Averages, Exponential Smoothing, ARIMA, SARIMA, TBATS
- Machine Learning — Linear Regression, XGBoost, Random Forest, or any ML model with reduction methods
- Deep Learning — RNN, LSTM



Statistical Models

When it comes to time series forecasting using statistical models, there are quite a few popular and well-accepted algorithms. Each of them has different mathematical modalities and they come with a different set of assumptions that must be satisfied.



ARIMA

ARIMA is one of the most popular classical methods for time series forecasting. It stands for autoregressive integrated moving average and is a type of model that forecasts given time series based on its own past values, that is, its own lags and the lagged forecast errors. ARIMA consists of three components:

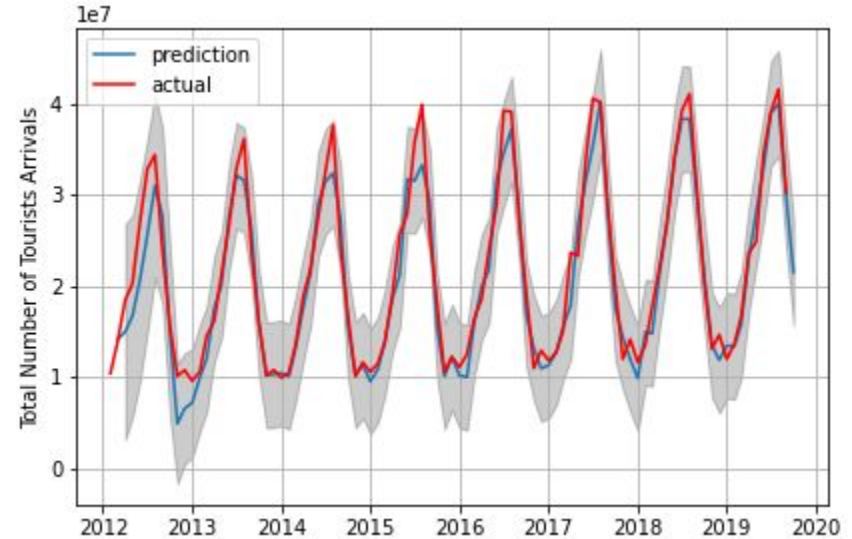
- Autoregression (AR): refers to a model that shows a changing variable that regresses on its own lagged, or prior, values.
- Integrated (I): represents the differencing of raw observations to allow for the time series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).
- Moving average (MA): incorporates the dependency between an observation and a residual error from a moving average model applied to lagged observations.

The "AR" part of [ARIMA](#) indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior observed) values. The "MA" part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The "I" (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

SARIMA



An extension to ARIMA that supports the direct modeling of the seasonal component of the series is called SARIMA. A problem with the ARIMA model is that it does not support seasonal data. That is a time series with a repeating cycle. ARIMA expects data that is either not seasonal or has the seasonal component removed, e.g. seasonally adjusted via methods such as seasonal differencing. SARIMA adds three new hyperparameters to specify the autoregression (AR), differencing (I), and moving average (MA) for the seasonal component of the series.



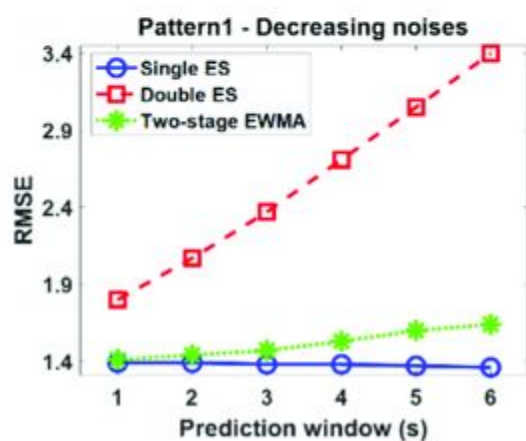
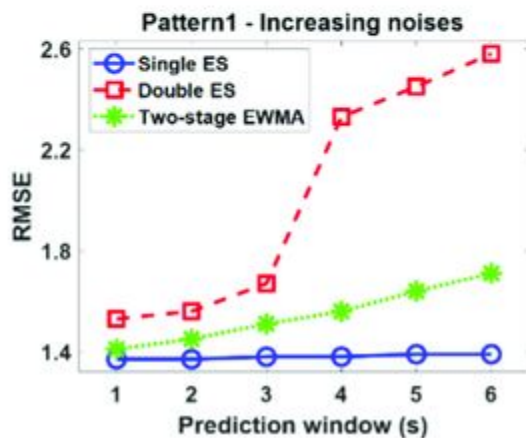
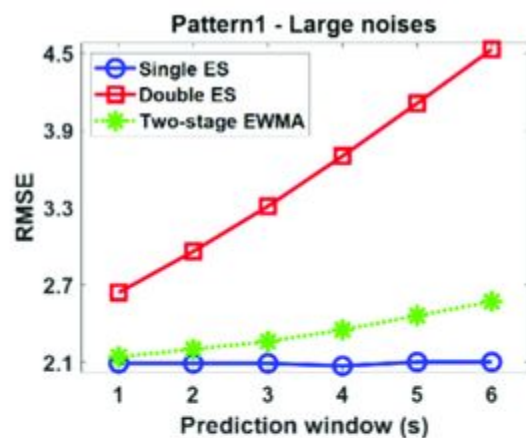
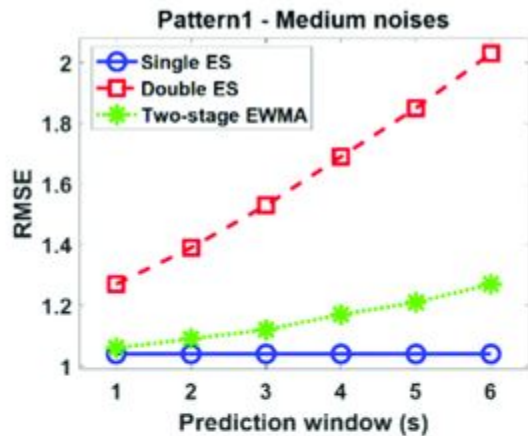
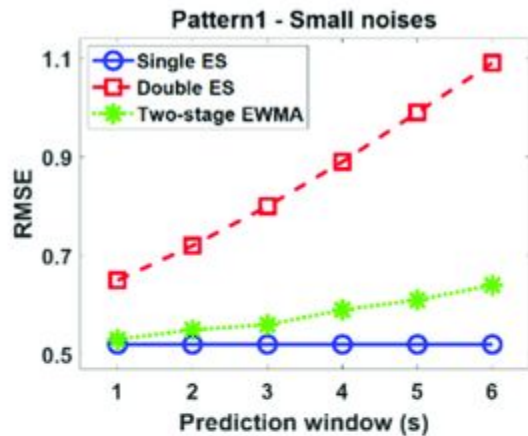


Exponential Smoothing (ETS)

Exponential smoothing is a time series forecasting method for univariate data. It can be extended to support data with a trend or seasonal component. It can be used as an alternative to the popular ARIMA family of models.

Exponential smoothing of time series data assigns exponentially decreasing weights for newest to oldest observations. The older the data, the less weight the data is given, whereas newer data is given more weight

- Simple (single) exponential smoothing uses a weighted moving average with exponentially decreasing weights.
- Holt's exponential smoothing is usually more reliable for handling data that shows trends.
- Triple exponential smoothing (also called the Multiplicative Holt-Winters) is more reliable for parabolic trends or data that shows trends and seasonality.

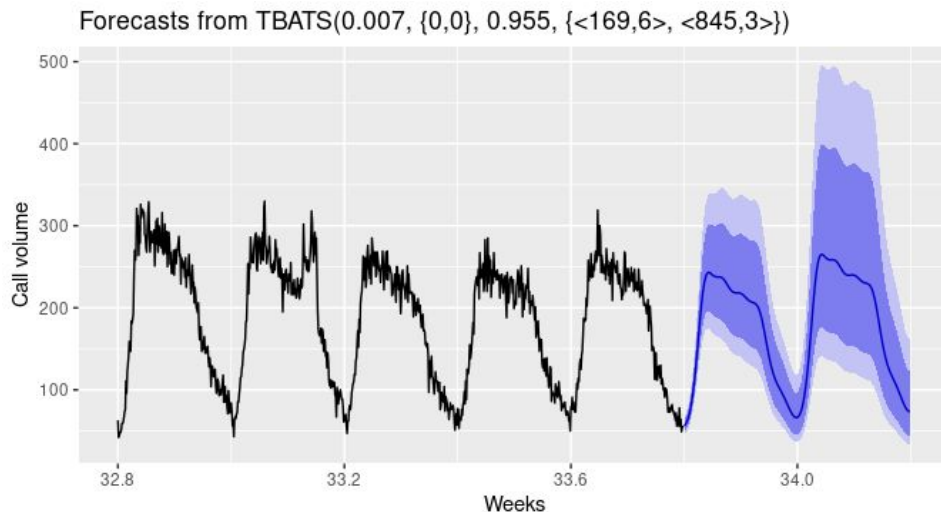




TBATS

TBATS models are for time series data with multiple seasonality. For example, retail sales data may have a daily pattern and weekly pattern, as well as an annual pattern.

In TBATS, a Box-Cox transformation is applied to the original time series, and then this is modeled as a linear combination of an exponentially smoothed trend, a seasonal component, and an ARMA component.

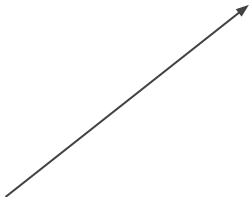




Machine Learning

If you don't want to use statistical models or they are not performing well, you can try this method. Machine learning is an alternative way of modeling time-series data for forecasting. In this method, we extract features from the data to add to our "X variable" and the value of the time-series is "y variable". Let's see an example:

We can extract features from the "Date" column such as a month, year, week of the year, etc.
See example:



	Date	Passengers
0	1949-01-01	112
1	1949-02-01	118
2	1949-03-01	132
3	1949-04-01	129
4	1949-05-01	121

	Series	Year	Month	Passengers
0	1	1949	1	112
1	2	1949	2	118
2	3	1949	3	132
3	4	1949	4	129
4	5	1949	5	121



	Series	Year	Month	Passengers
0	1	1949	1	112
1	2	1949	2	118
2	3	1949	3	132
3	4	1949	4	129
4	5	1949	5	121

```
# extract month and year from dates
data['Month'] = [i.month for i in data['Date']]
data['Year'] = [i.year for i in data['Date']]
```

```
# create a sequence of numbers
data['Series'] = np.arange(1,len(data)+1)
```

```
# drop unnecessary columns and re-arrange
data.drop(['Date', 'MA12'], axis=1,
inplace=True)
data = data[['Series', 'Year', 'Month',
'Passengers']]
```

```
# check the head of the dataset
data.head()
```




Something to note here is that the train-test-split for time-series data is special. Because you cannot change the order of the table, you have to ensure that you don't sample randomly as you want your test data to contain points that are in the future from the points in the train data (time always moves forward):

Now that we have done the train-test-split, we are ready to train a machine learning model on the train data, score it on the test data and evaluate the performance of our model. In this example, I will use PyCaret; an open-source, low-code machine learning library in Python that automates machine learning workflows. To use PyCaret, you have to install it using pip.

```
# split data into train-test set
train = data[data['Year'] < 1960]
test = data[data['Year'] >= 1960]
```

```
# check shape
train.shape, test.shape
>>> ((132, 4), (12, 4))
```

```
# install pycaret
pip install pycaret
```

```
# import the regression module
from pycaret.regression import *
```

```
# initialize setup
s = setup(data = train, test_data = test, target = 'Passengers',
fold_strategy = 'timeseries', numeric_features = ['Year', 'Series'],
fold = 3, transform_target = True, session_id = 123)
```

```
best = compare_models(sort = 'MAE')
```



	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lar	Least Angle Regression	22.3980	923.8654	28.2855	0.5621	0.0878	0.0746	0.0100
lr	Linear Regression	22.3981	923.8749	28.2856	0.5621	0.0878	0.0746	1.2500
huber	Huber Regressor	22.4274	892.3078	27.9491	0.5981	0.0880	0.0749	0.0200
br	Bayesian Ridge	22.4783	932.2165	28.5483	0.5611	0.0884	0.0746	0.0133
ridge	Ridge Regression	23.1976	1003.9426	30.0410	0.5258	0.0933	0.0764	0.8433
lasso	Lasso Regression	38.4188	2413.5109	46.8468	0.0882	0.1473	0.1241	1.0333
en	Elastic Net	40.6486	2618.8759	49.4048	-0.0824	0.1563	0.1349	0.0133
omp	Orthogonal Matching Pursuit	44.3054	3048.2658	53.8613	-0.4499	0.1713	0.1520	0.0133
xgboost	Extreme Gradient Boosting	46.7192	3791.0476	59.9683	-0.5515	0.1962	0.1432	0.2500
gbr	Gradient Boosting Regressor	50.1217	4032.0567	61.2306	-0.6189	0.2034	0.1538	0.0200
rf	Random Forest Regressor	52.3637	4647.0635	65.2883	-0.7726	0.2131	0.1578	0.2133
catboost	CatBoost Regressor	53.6141	4414.8319	64.3184	-0.7792	0.2161	0.1653	0.8133
et	Extra Trees Regressor	54.6312	4500.5115	64.0882	-0.7207	0.2146	0.1675	0.1700
ada	AdaBoost Regressor	55.0753	5128.1587	68.9577	-0.9915	0.2277	0.1667	0.0333
dt	Decision Tree Regressor	57.9293	6230.5556	70.9838	-0.9553	0.2265	0.1700	0.0167
knn	K Neighbors Regressor	64.1165	7098.4735	78.7031	-1.4511	0.2582	0.1882	0.0533
lightgbm	Light Gradient Boosting Machine	76.8521	8430.4943	91.0063	-2.9097	0.3379	0.2490	0.0200
llar	Lasso Least Angle Regression	129.0182	21858.5806	138.1309	-6.5554	0.5446	0.3958	0.0133
par	Passive Aggressive Regressor	156.1775	95107.3645	210.3616	-93.7884	0.4304	0.6643	0.0167



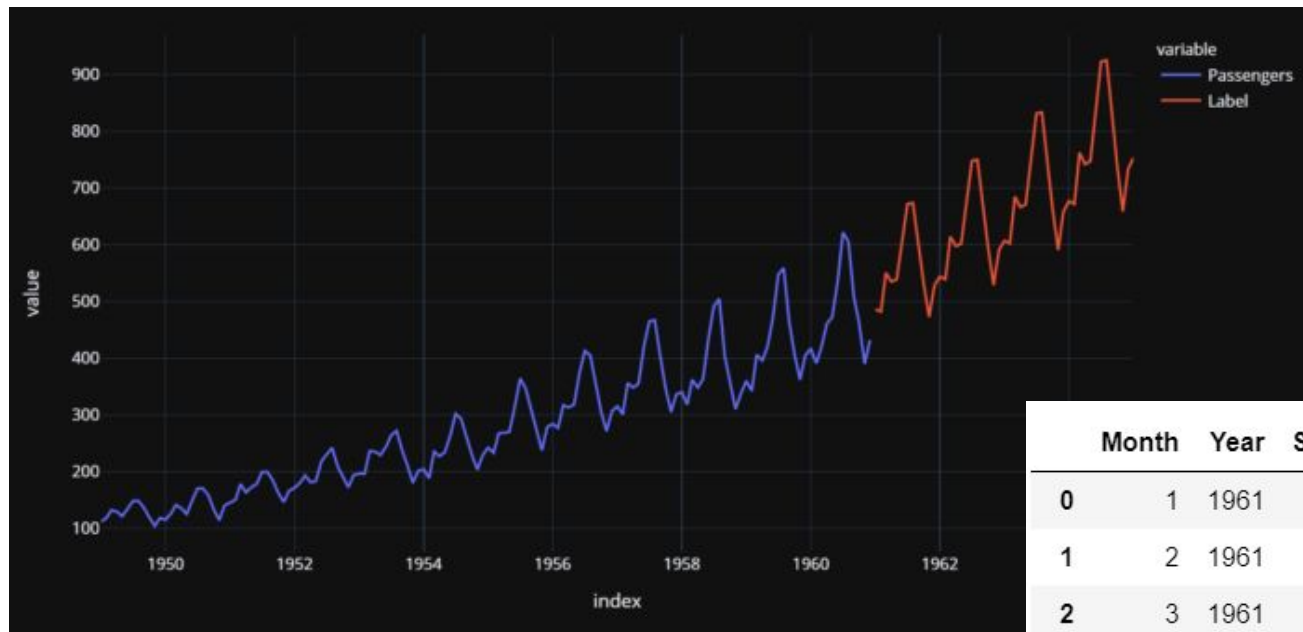
The best model using 3 fold cross-validation based on Mean Absolute Error (MAE) is Leased Angle Regression. We can now use this model to forecast the future. For that, we have to create "X variables" in the future. This can be done by creating future dates and then extracting features from them.

Since we have trained our model on the data until 1960, let's predict five years out in the future through 1965. To use our final model to generate future predictions, we first need to create a dataset consisting of the Month, Year, Series column on the future dates. This code below creates the future "X" dataset.

```
future_dates = pd.date_range(start = '1961-01-01', end = '1965-01-01', freq = 'MS')
future_df = pd.DataFrame()
future_df['Month'] = [i.month for i in future_dates]
future_df['Year'] = [i.year for i in future_dates]
future_df['Series'] = np.arange(145,(145+len(future_dates)))
future_df.head()
```

	Month	Year	Series
0	1	1961	145
1	2	1961	146
2	3	1961	147
3	4	1961	148
4	5	1961	149

```
predictions_future = predict_model(best, data=future_df)
predictions_future.head()
```



	Month	Year	Series	Label
0	1	1961	145	486.278268
1	2	1961	146	482.208186
2	3	1961	147	550.485953
3	4	1961	148	535.187166
4	5	1961	149	538.923776



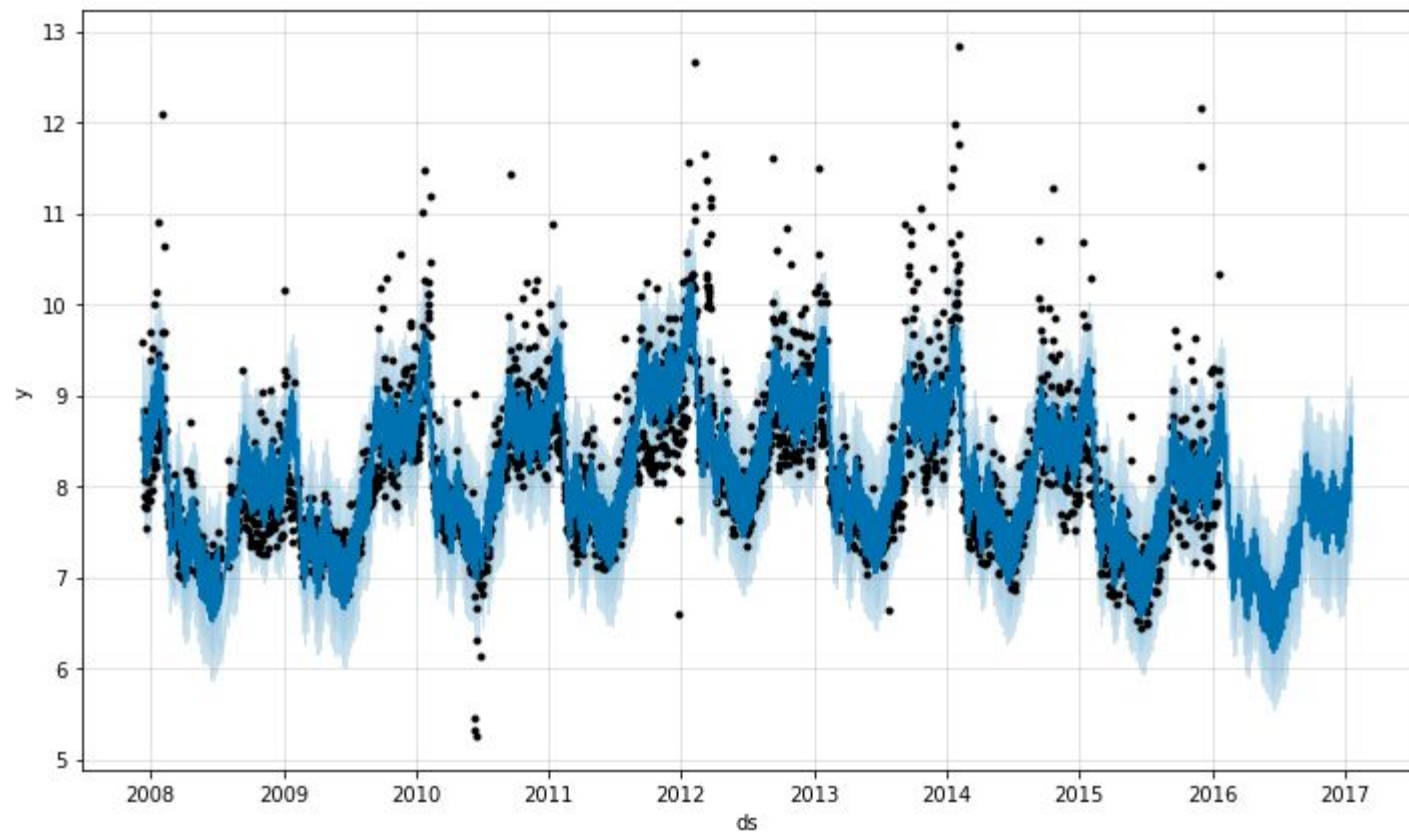
There are a few important elements to note here. Whenever you are dealing with univariate time series you can always convert them into regression problems and solve them as in this example. However, you have to be careful about cross-validation. You cannot do random cross-validation on time-series models and you must use time-series appropriate techniques.



Python Frameworks for Forecasting

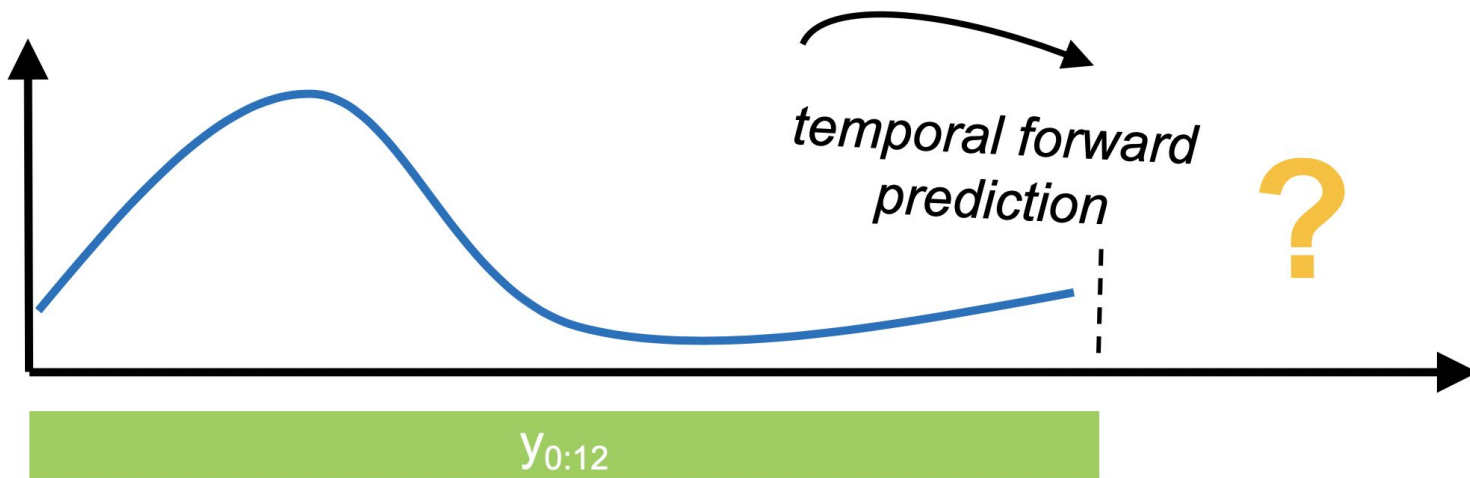
Prophet is open-source software released by Facebook's Core Data Science team. It is available for download on CRAN and PyPI.

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well



sktime

sktime is an open-source, unified framework for machine learning with time series. It provides an easy-to-use, flexible and modular platform for a wide range of time series machine learning tasks. It offers scikit-learn compatible interfaces and model composition tools, with the goal to make the ecosystem more usable and interoperable as a whole.





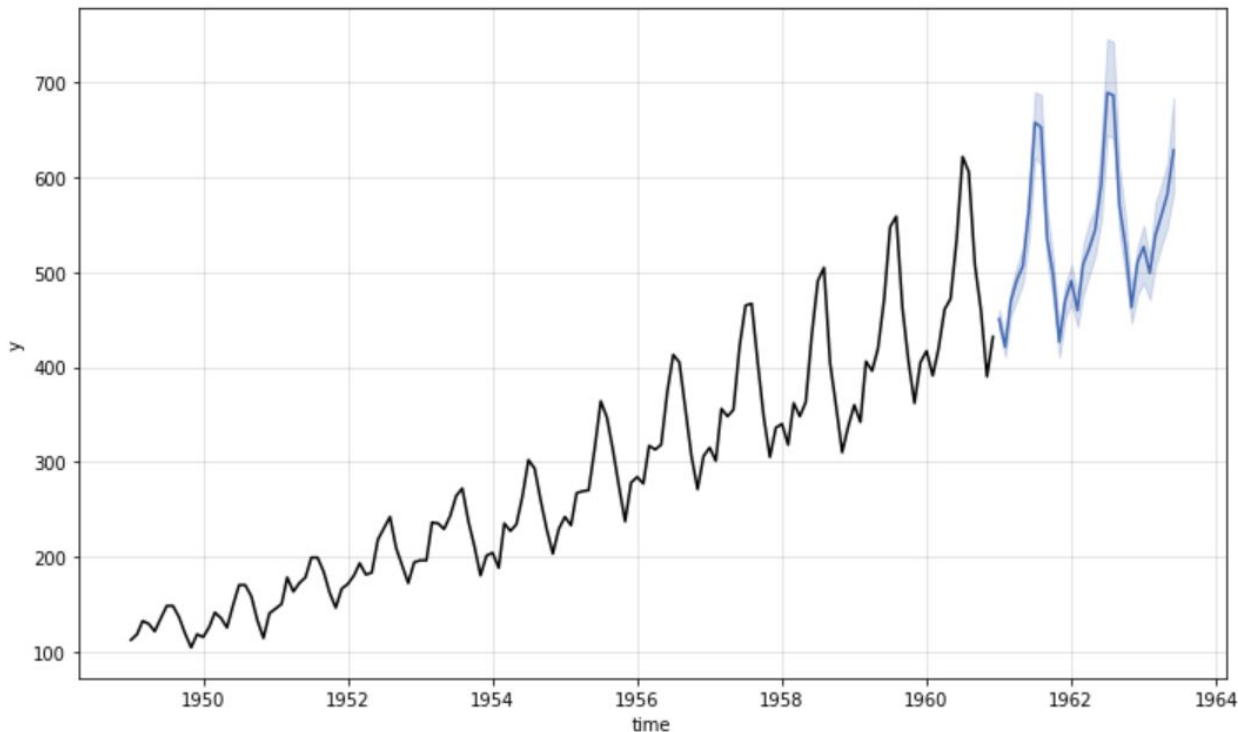
pmdarima

Pmdarima is a statistical library designed to fill the void in Python's time series analysis capabilities. This includes:

- The equivalent of R's `auto.arima` functionality
- A collection of statistical tests of stationarity and seasonality
- Seasonal time series decompositions
- Cross-validation utilities
- A rich collection of built-in time series datasets for prototyping and examples



Kats

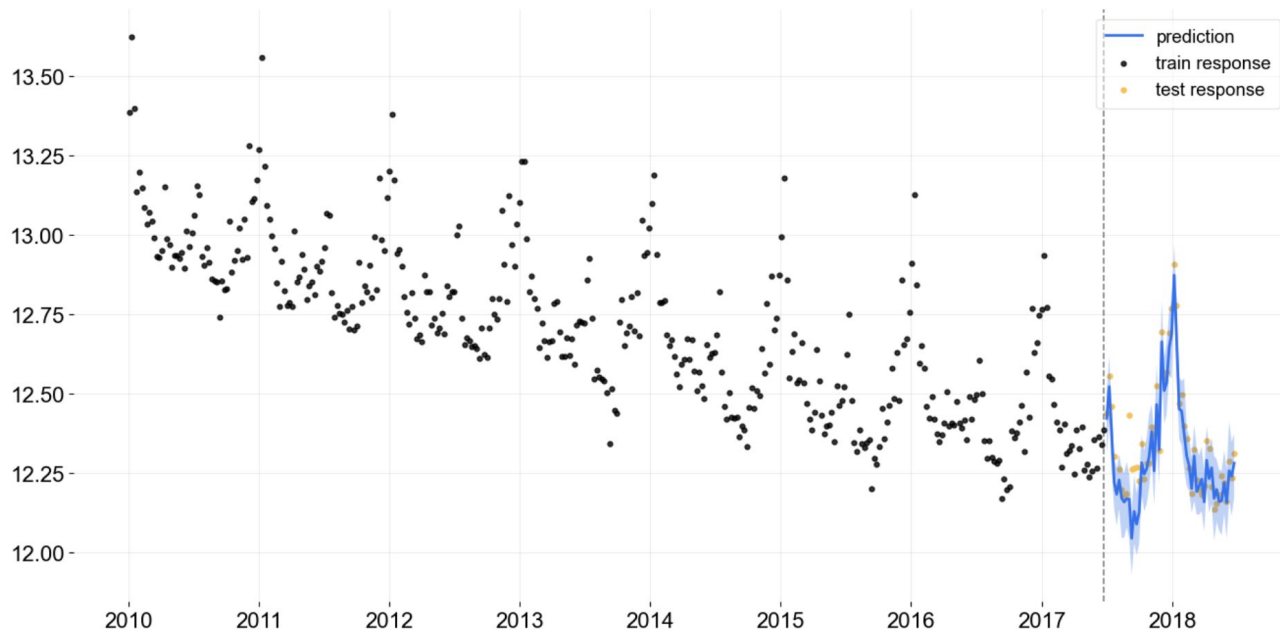


Kats is another amazing open-source project by Facebook, released by their Infrastructure Data Science team. It is available for download on PyPI.

Kats is a toolkit to analyze time-series data; a lightweight, easy-to-use, and generalizable framework to perform time series analysis. Kats aims to provide a one-stop shop for time series analysis, including detection, forecasting, feature extraction/embedding, and multivariate analysis, etc.



Orbit



Orbit is an amazing open-source project by Uber. It is a Python library for Bayesian time series forecasting. It provides a familiar and intuitive initialize-fit-predict interface for time series tasks, while utilizing probabilistic programming languages under the hood.



PyCaret

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. With PyCaret, you spend less time coding and more time on analysis. You can train your model, analyze it, iterate faster than ever before, and deploy it instantaneously as a REST API or even build a simple front-end ML app, all from your favorite Notebook

