

[Home](#) > [Tutorials](#) > [Data Science](#)

# Time Series Forecasting Tutorial

A detailed guide to time series forecasting. Learn to use python and supporting frameworks. Learn about the statistical modelling involved.

Feb 2022 · 19 min read



**Moez Ali**

Data Scientist, Founder & Creator of PyCaret

---

## TOPICS

Data Science

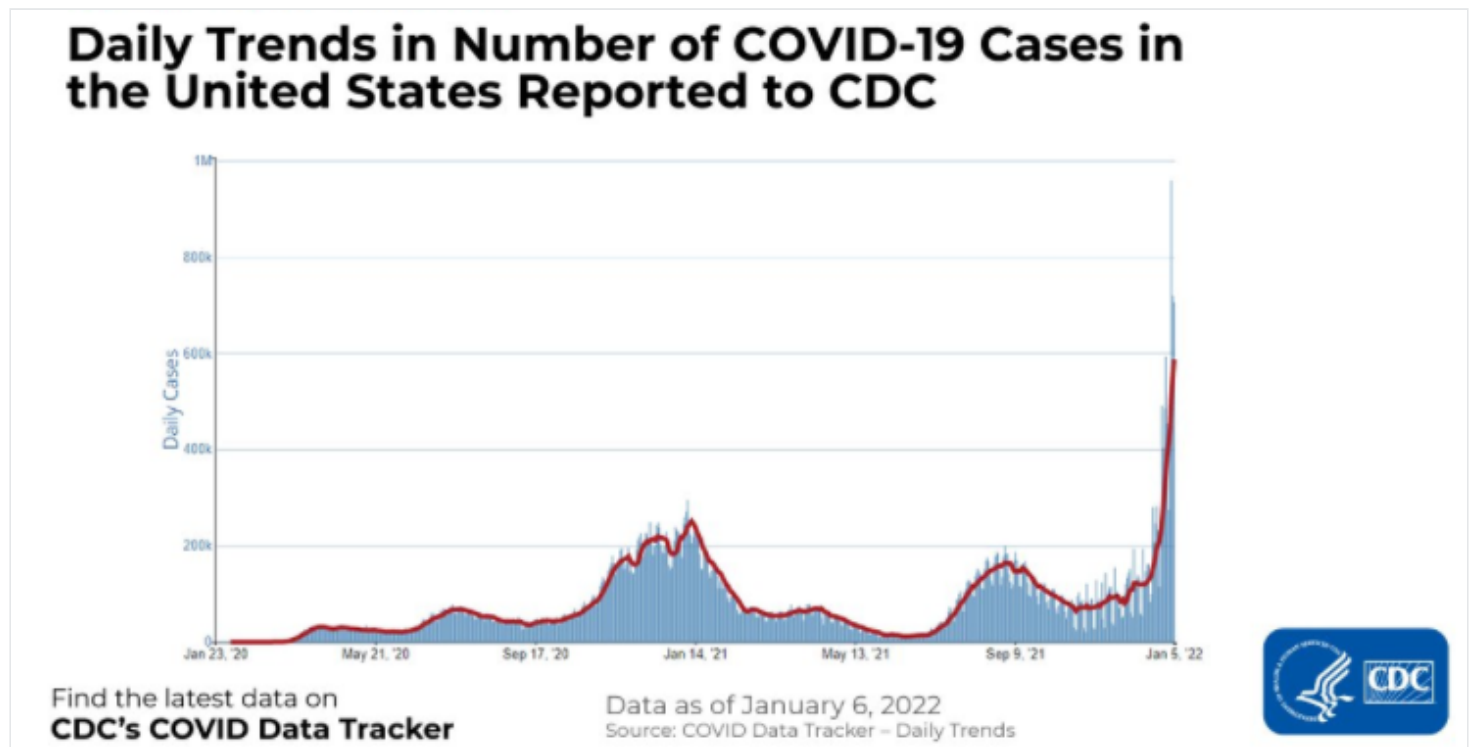
Machine Learning

## Introduction

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.

Below is an example of time series data showing the number of COVID-19 cases in the United States as reported to CDC. The x-axis shows the passing of time and the y-axis

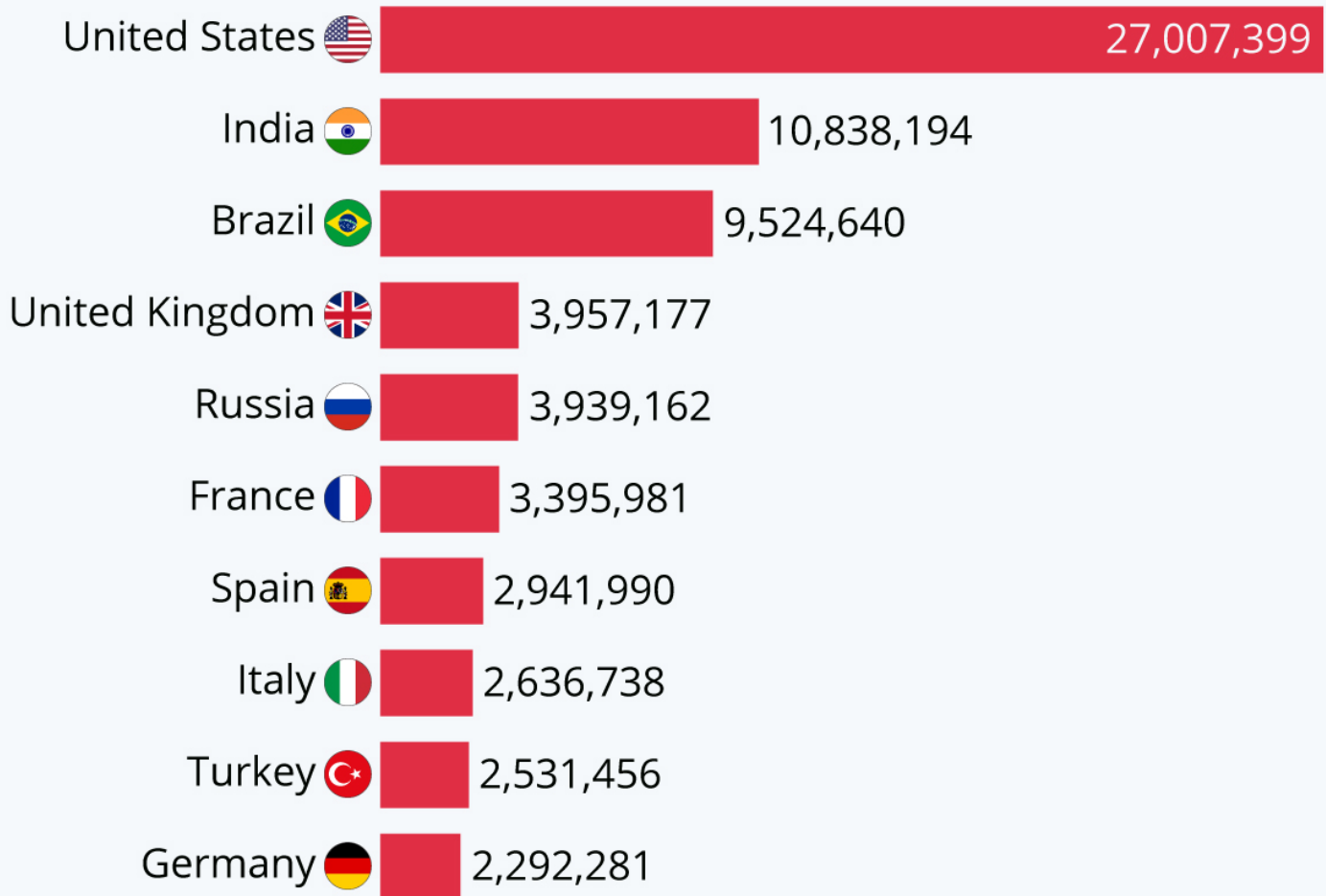
represents the number of COVID-19 cases in thousands.



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. An example of this is shown in the dataset below, tracking countries with the most COVID-19 cases in a fixed and consistent time period for all countries.

# 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



It is not very hard to distinguish the difference between cross-sectional and time-series data as the objectives of analysis for both datasets are widely different. In our examples,

we were first interested in tracking COVID-19 cases over a period of time, before analyzing COVID-19 cases by country in a fixed time period.

A typical real-world dataset is likely to be a hybrid of these formats. For example, we could think of a retailer like Walmart that sells thousands of products every day. If you analyze the sales by product on a particular day, this will be a cross-sectional analysis. You could want to find out what the number 1 selling item on Christmas Eve is for example.

Comparatively, if you wanted to find out the sale of one particular item over a period of time (let's say last 5 years), this would be a time-series analysis.

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.

## What is 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.

There are numerous reasons why companies may be interested in forecasting future values, namely GDP, monthly sales, inventory, unemployment, and global temperatures:

- A retailer may be interested in predicting future sales at an SKU (stock keeping unit) level for planning and budgeting.
- A small merchant may be interested in forecasting sales by store, so it can schedule the right resources (more people during busy periods and vice versa).
- A software giant like Google may be interested in knowing the busiest hour of the day or busiest day of the week so that they can schedule server resources accordingly.
- The health department may be interested in predicting the cumulative COVID vaccinations administered so that they can further predict when herd immunity is expected to kick in.

## Type of Time Series Forecasting

There are three types of time series forecasting. Which one you should use depends on the type of data you are dealing with and the use-case in hand:

## 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.

Consider the above example and suppose that our dataset includes other weather-related attributes over the same time period, such as perspiration percent, dew point, wind speed, etc., along with the temperature values. In this case, there are multiple variables to be considered to optimally predict temperature. A series like this would fall under the category of multivariate time series. Your dataset will look like this now:

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

You are still forecasting temperature values for the future but now you can use other available information in your forecast as we assume temperature values will be dependent on these factors as well.

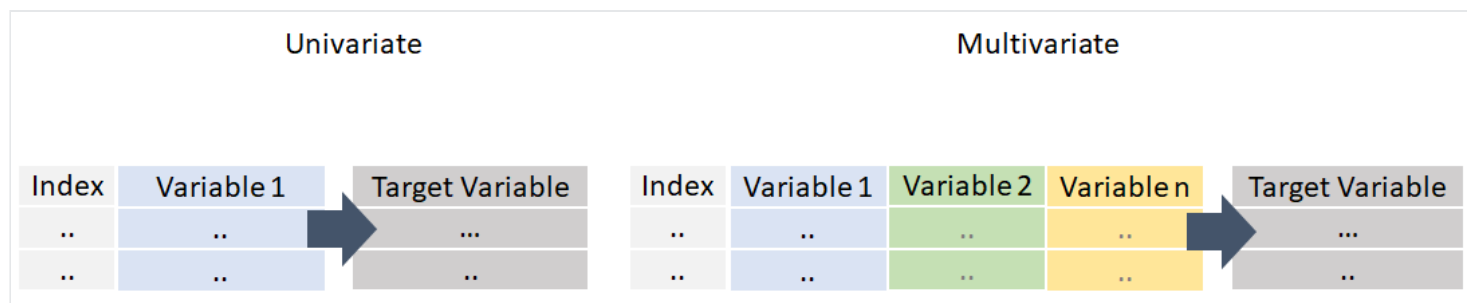


Image Source: Van Nguyen

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.

This tutorial will discuss several classical models but not all of them support multivariate time series forecasting. In situations like these, machine learning models come to the rescue as you can model any time series forecasting problem with regression. We will see an example of this later in this tutorial.

## 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

This tutorial is focused on the first two methods: classical/statistical models and machine learning. **Deep learning** methods are out-of-scope for this tutorial.

## 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. This tutorial will not go in-depth on the mathematical concepts, rather will just give an intuition that you will hopefully find helpful.

### 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

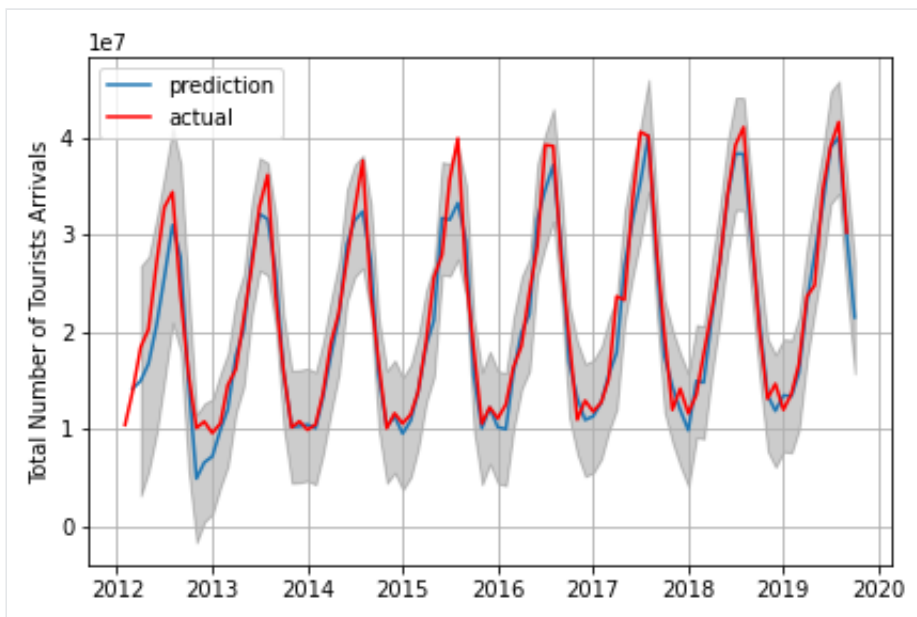
series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).

- 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.





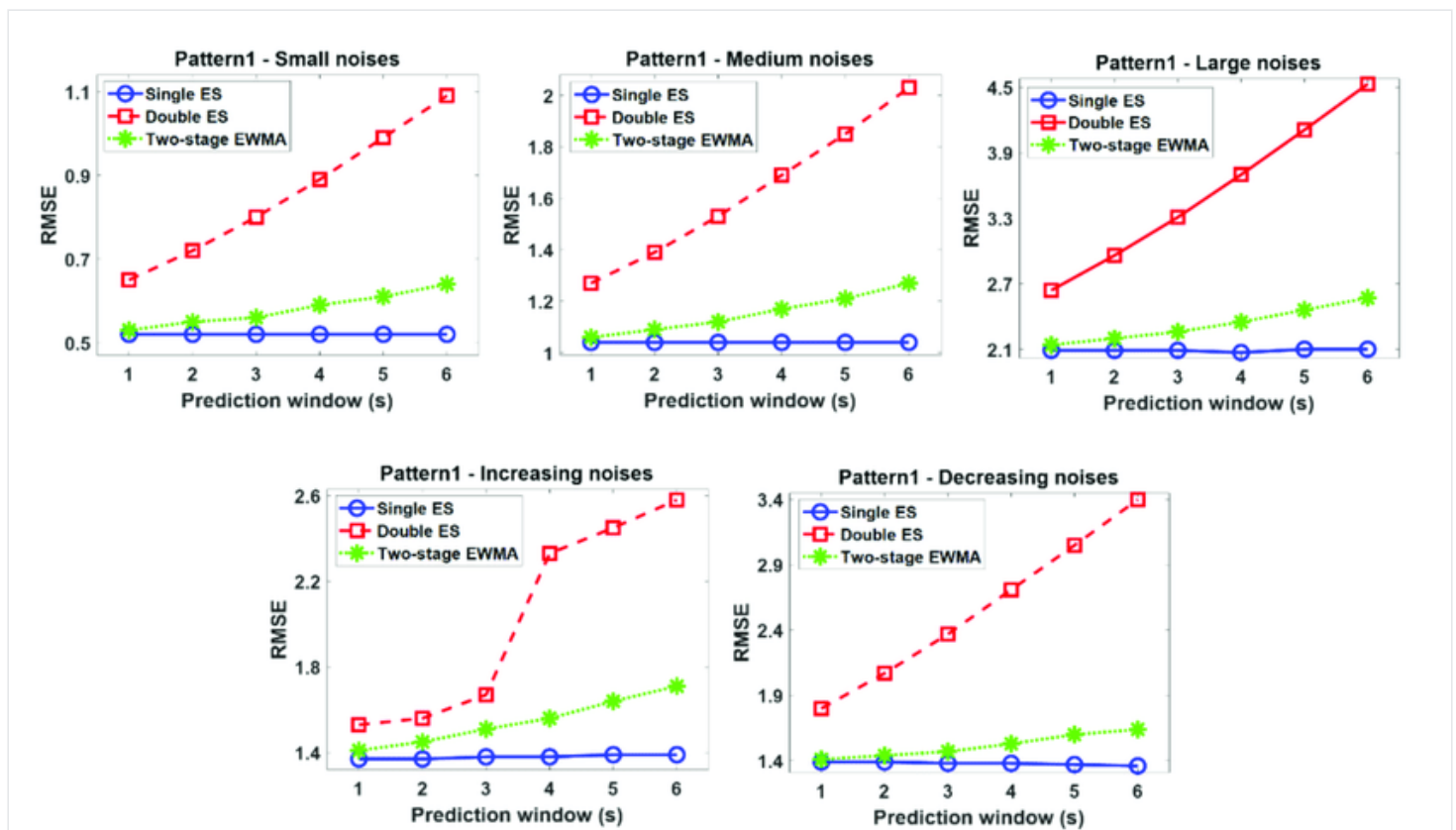
## Forecast using SARIMA model

# Exponential Smoothing

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.

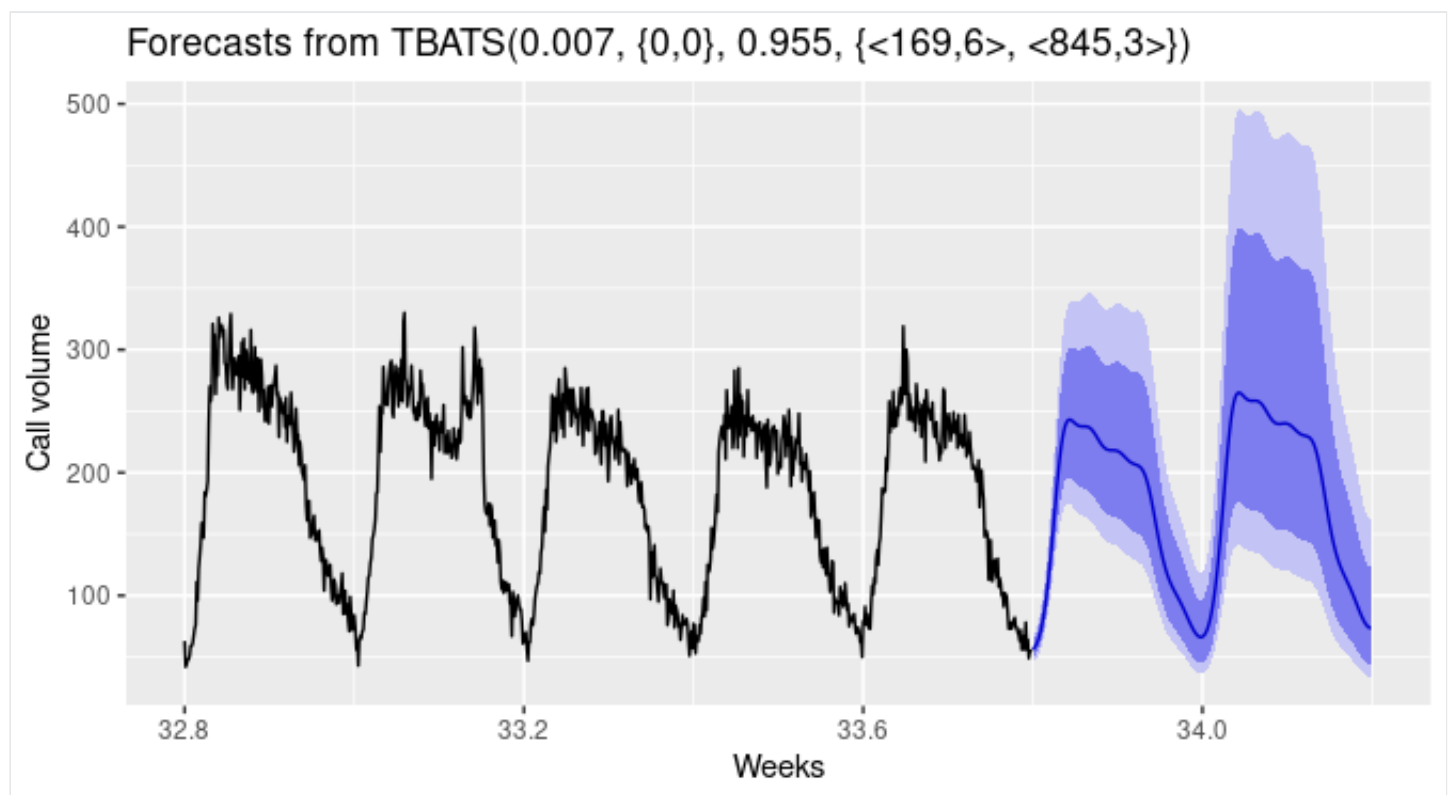


## comparison results between a single exponential smoothing (ES), double ES, and two-stage EWMA

### 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.



Forecast using TBATS

## 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 date to add to our "X variable" and the value of the time-series is "y variable". Let's see an example:

For the purpose of this tutorial, I have used the US airline passengers dataset available to download from [Kaggle](#).

	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

## Dataset Sample

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

```
# 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()
```



 Explain code

 OpenAI

	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

### Sample rows after extracting features

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).

```
# 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))
```

 Explain code



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.

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

[✦ Explain code](#)

If you need any help during installation, please refer to the official documentation.

Assuming you have installed PyCaret successfully:

```
# import the regression module  
from pycaret.regression import *  
  
# initialize setup  
s = setup(data = train, test_data = test, target = 'Passengers', fold_strategy =
```

[✦ Explain code](#)

Now to train machine learning models, you just need to run one line:

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

[✦ Explain code](#)

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

## Output from compare\_models

The best model using 3 fold cross-validation based on Mean Absolute Error (MAE) is Least 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', frequency='M')
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()
```

 Explain code OpenAI

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

Sample rows from future\_df

Now we can use future\_df to make predictions:

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

 Explain code OpenAI

	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

Output from `predictions_future.head()`

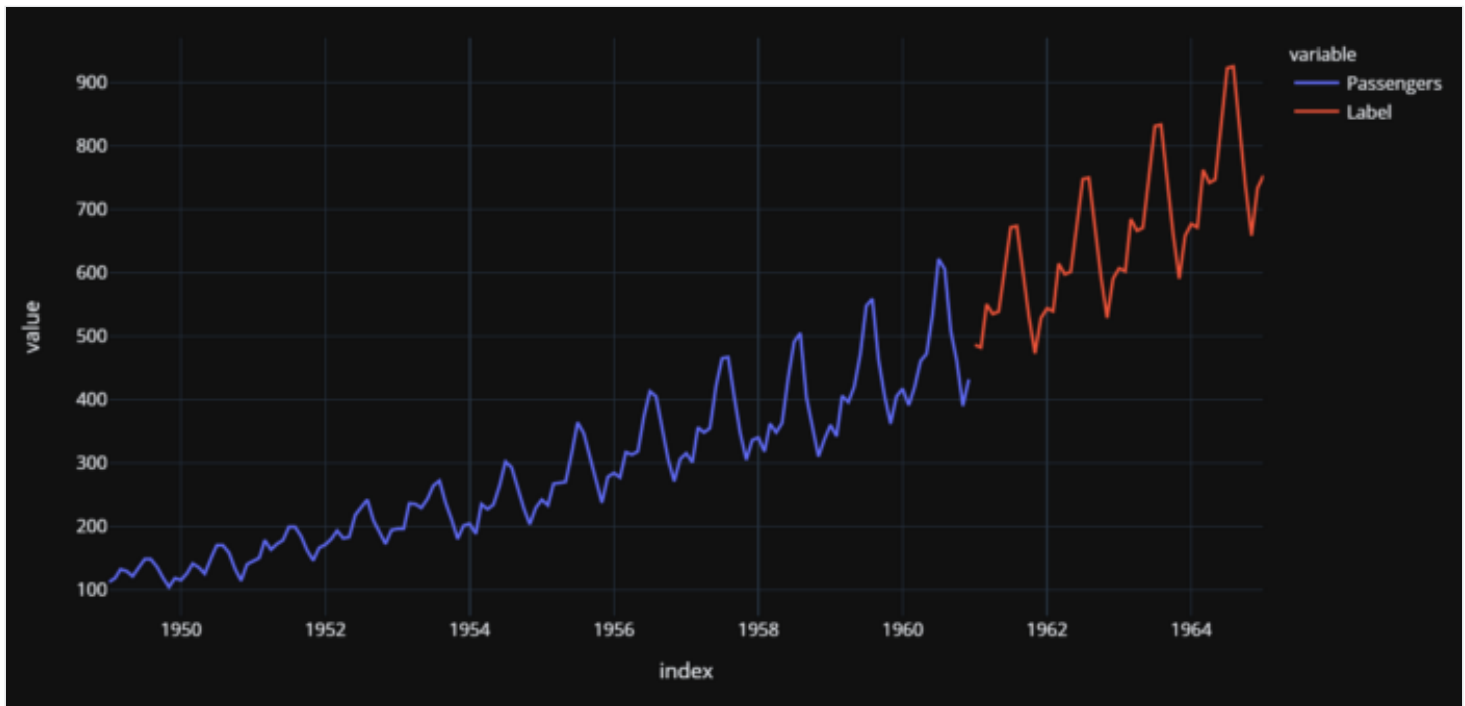
And now we can plot it:

```
concat_df = pd.concat([data, predictions_future], axis=0)
concat_df_i = pd.date_range(start='1949-01-01', end = '1965-01-01', freq = 'MS')
concat_df.set_index(concat_df_i, inplace=True)
fig = px.line(concat_df, x=concat_df.index, y=["Passengers", "Label"], template
fig.show()
```

 Explain code

 OpenAI





Actual (1949–1960) and Predicted (1961–1964) US airline passengers

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. In this example, PyCaret uses [TimeSeriesSplit](#) from the scikit-learn library.

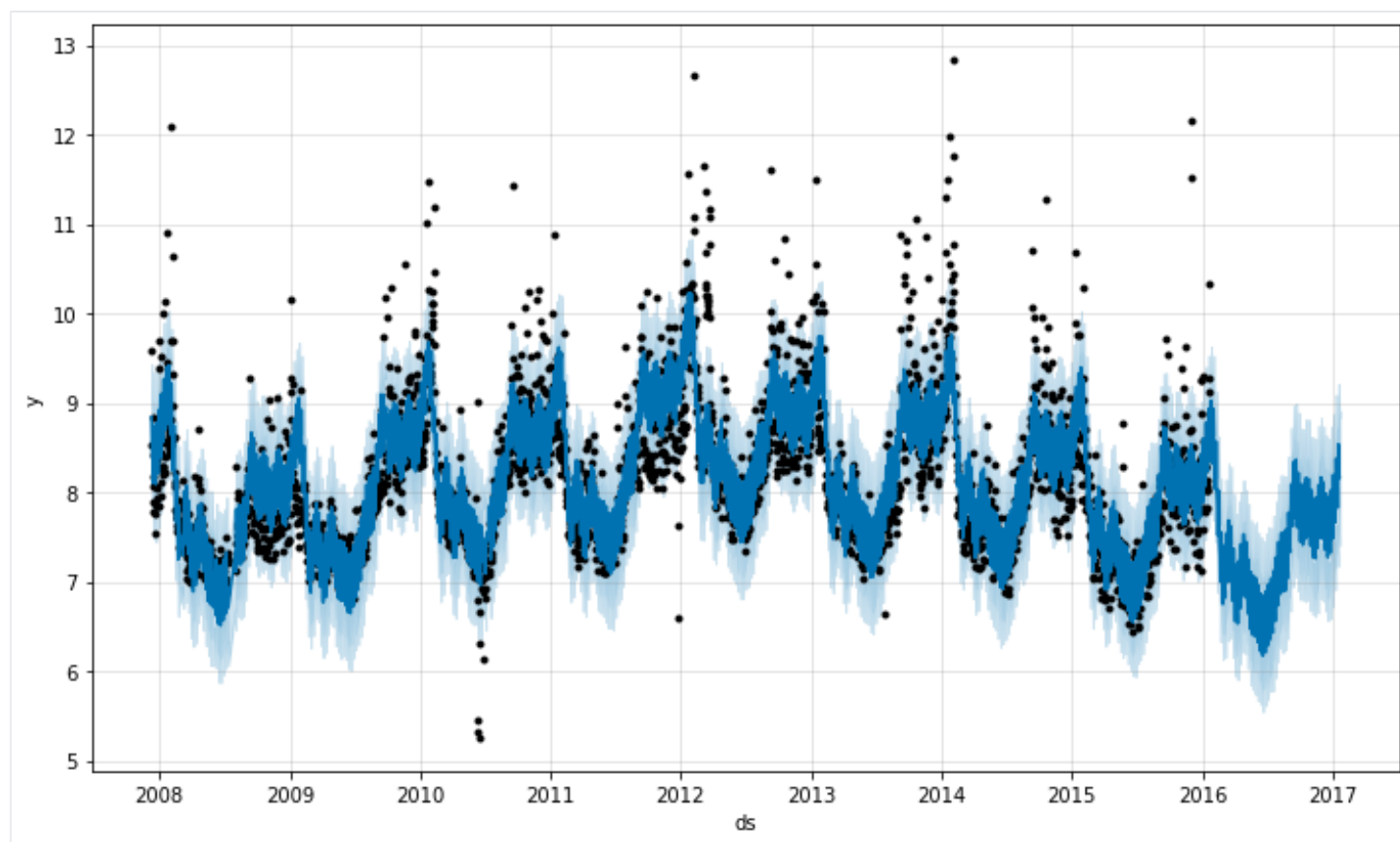
## Python Frameworks for Forecasting

### Facebook Prophet

**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.

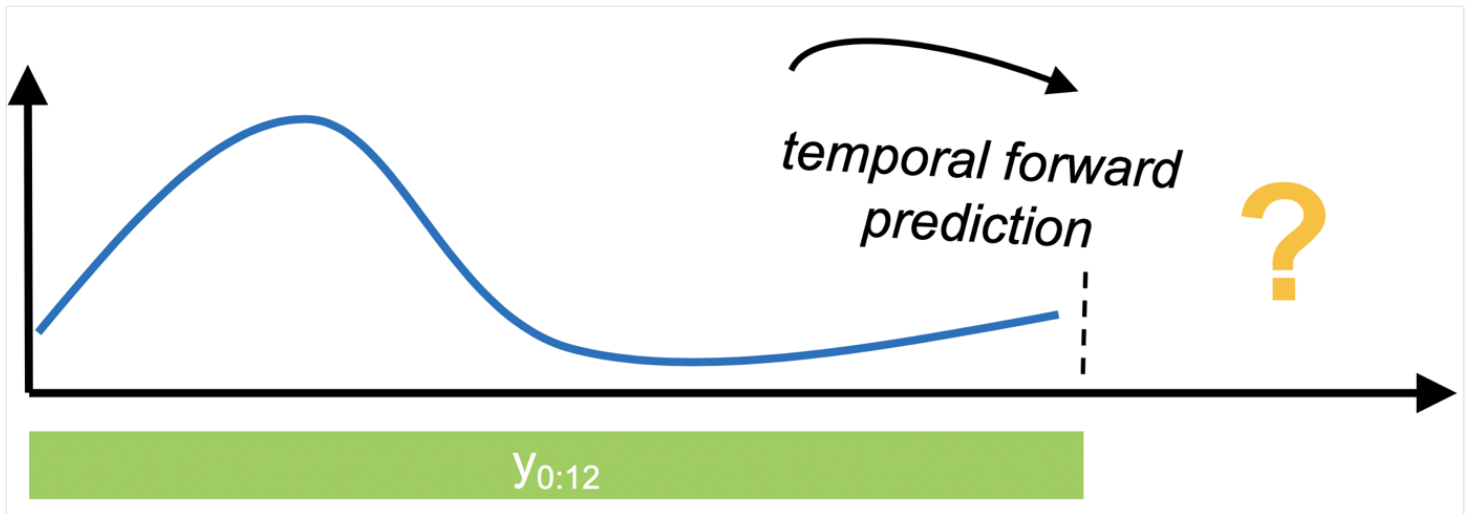
To learn more about it, check out this [link](#).



Forecast using FB Prophet

## 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.



To learn more about sktime, check out this [link](#).

## pmdarima

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

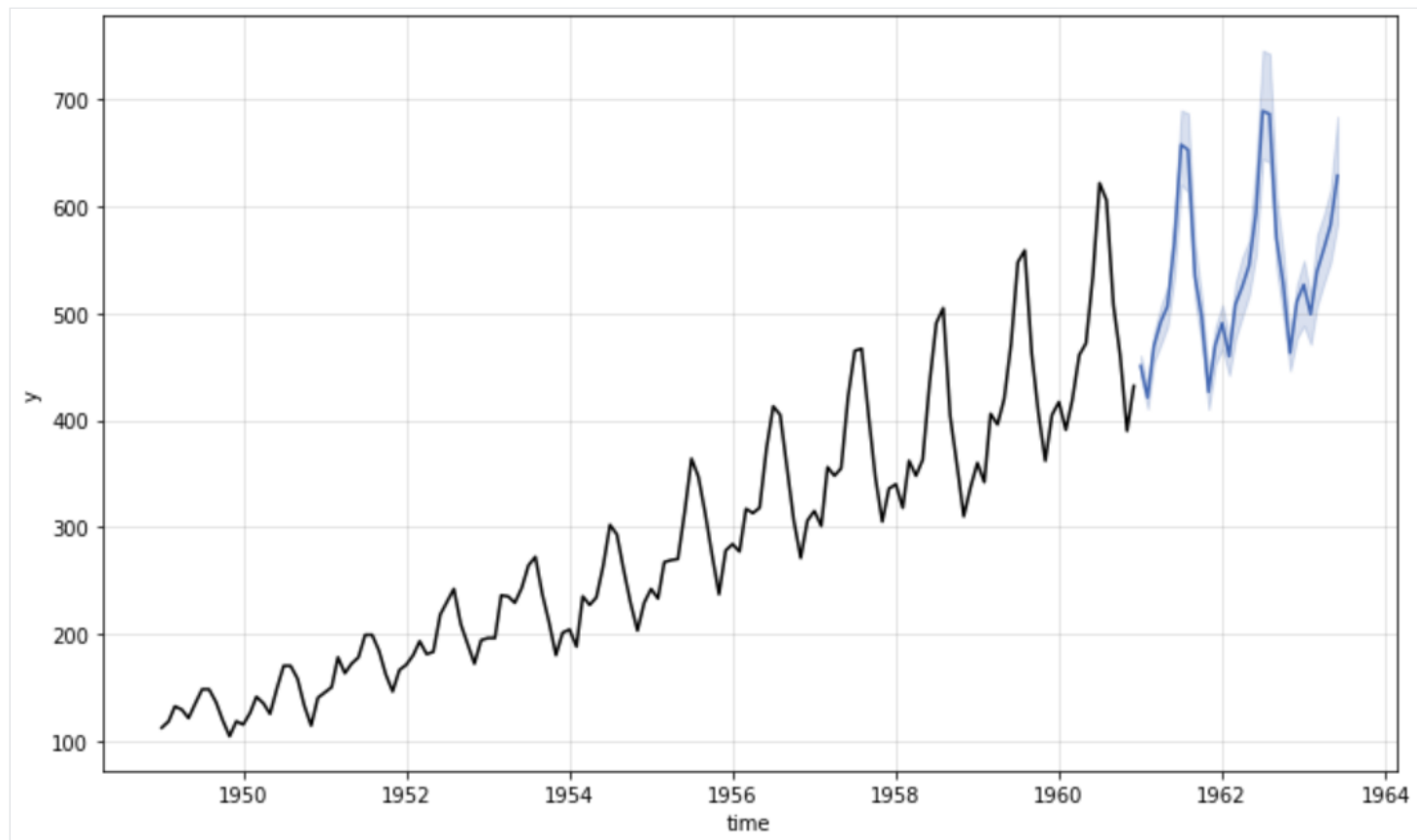
- The equivalent of R's `auto.arima` functionality
- A collection of statistical tests of stationarity and seasonality
- Seasonal time series decompositions
- Cross-validation utilities
- A rich collection of built-in time series datasets for prototyping and examples

To learn more about pmdarima, check out [this page](#).

## 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.

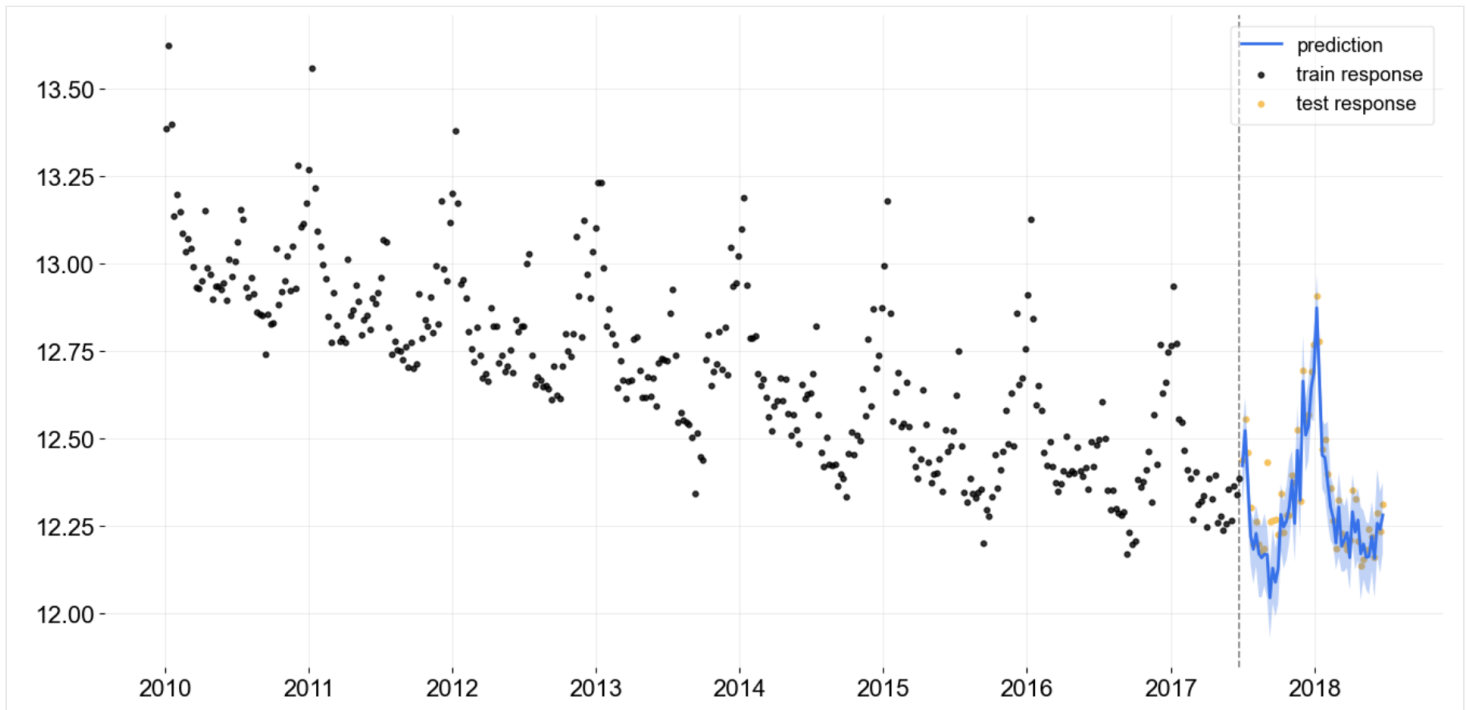


## Forecast using KATS

To learn more about KATS, check out [this link](#).

## 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.

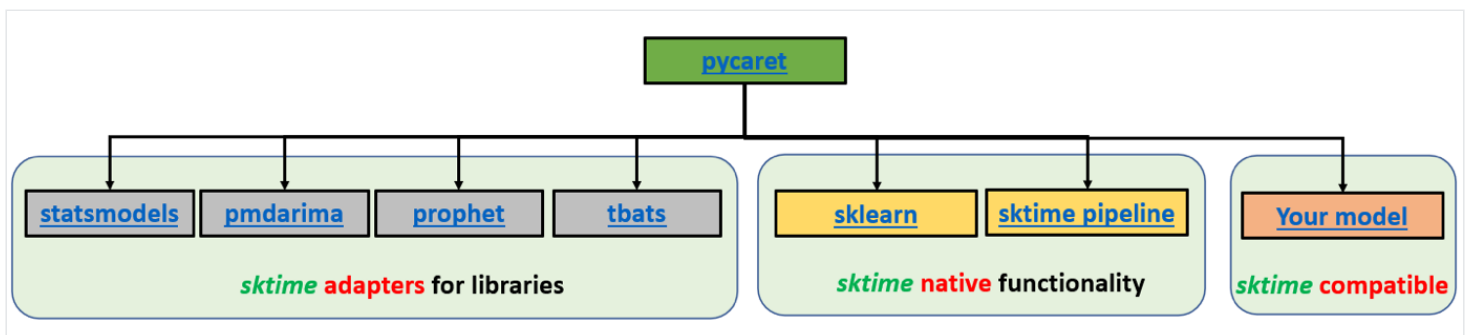


## Forecast using Orbit

To learn more about Orbit, check out this [link](#).

## 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.



PyCaret Time Series Module Architecture. Developed by Lead Developer Nikhil Gupta.

Next, we will see an example of an end-to-end time-series forecasting task using PyCaret library.

## End-to-end Example

PyCaret's time series module is available in beta, you can download it from pip.

```
pip install pycaret-ts-alpha
```



 Explain code

 OpenAI

Link to this tutorial on [Colab](#).

## Dataset

Here we have used the airline dataset which is also available under [PyCaret's data repository](#).

```
from pycaret.datasets import get_data  
data = get_data('airlineer')
```

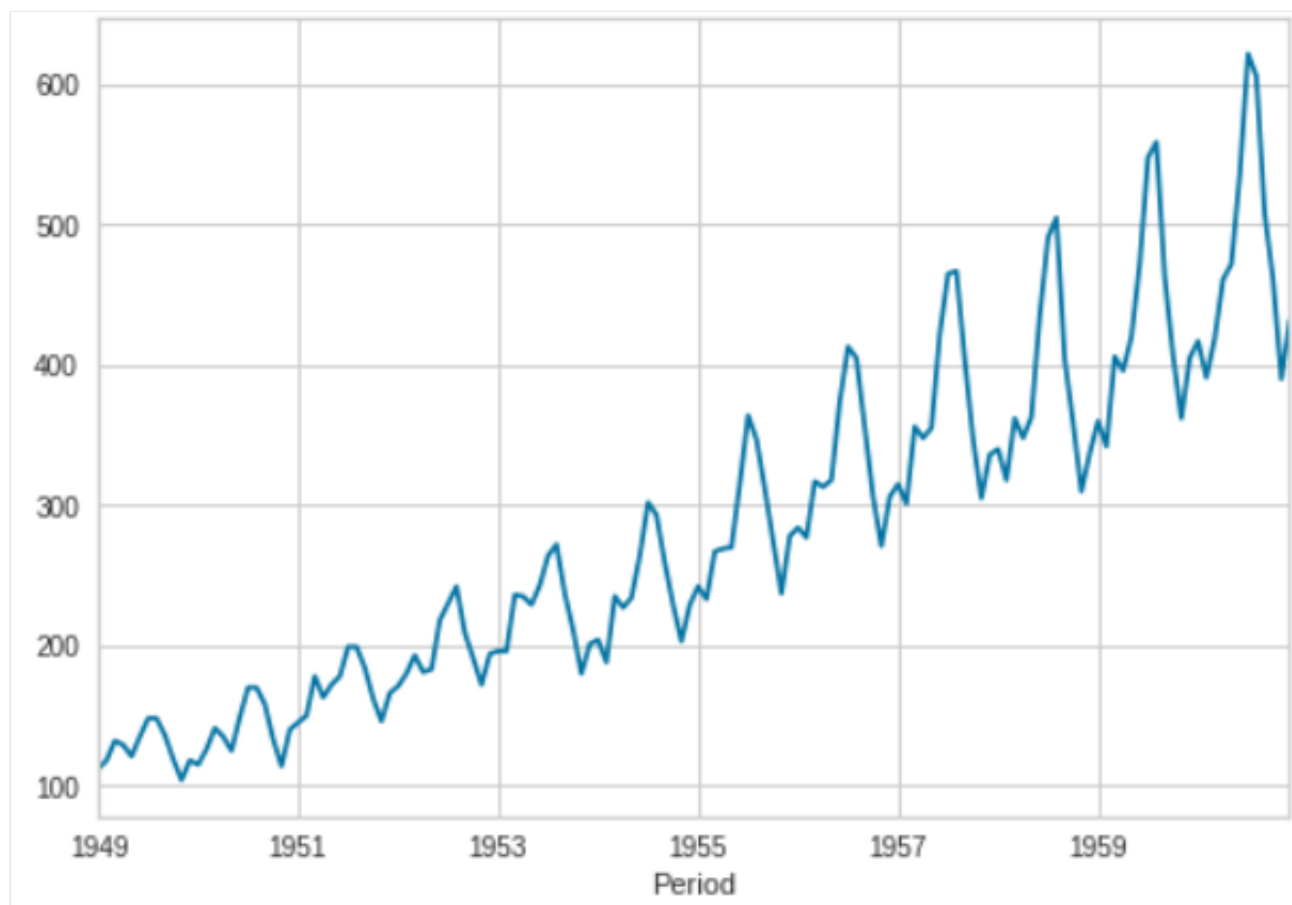


 Explain code

 OpenAI

```
Period  
1949-01    112.0  
1949-02    118.0  
1949-03    132.0  
1949-04    129.0  
1949-05    121.0  
Freq: M, Name: Number of airline passengers, dtype: float64
```

```
data.plot()
```

[✦ Explain code](#)

## Setup Experiment

Experiments in PyCaret are started using the `setup` function. This function takes care of all preprocessing steps, train-test-split, cross-validation strategy, and a few other tasks.

```
from pycaret.time_series import *  
s = setup(data, session_id = 123)
```

[✦ Explain code](#)

	Description	Value
0	session_id	123
1	Original Data	(144, 1)
2	Missing Values	False
3	Transformed Train Set	(143,)
4	Transformed Test Set	(1,)
5	Fold Generator	ExpandingWindowSplitter
6	Fold Number	3
7	Enforce Prediction Interval	False
8	Seasonal Period Tested	12
9	Seasonality Detected	True
10	Seasonality Used in Models	12
11	Target Strictly Positive	True
12	Target White Noise	No
13	Recommended d	1
14	Recommended Seasonal D	1
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	ts-default-name
19	USI	631c
20	Imputation Type	simple

Output from the setup function

## Exploratory Data Analysis (EDA)

When it comes to EDA of univariate time-series data, there are a few things you can and should do.



For the use of classical and statistical models, you can run some statistical tests. There is a function in PyCaret that does this for you.

```
check_stats()
```



Explain code



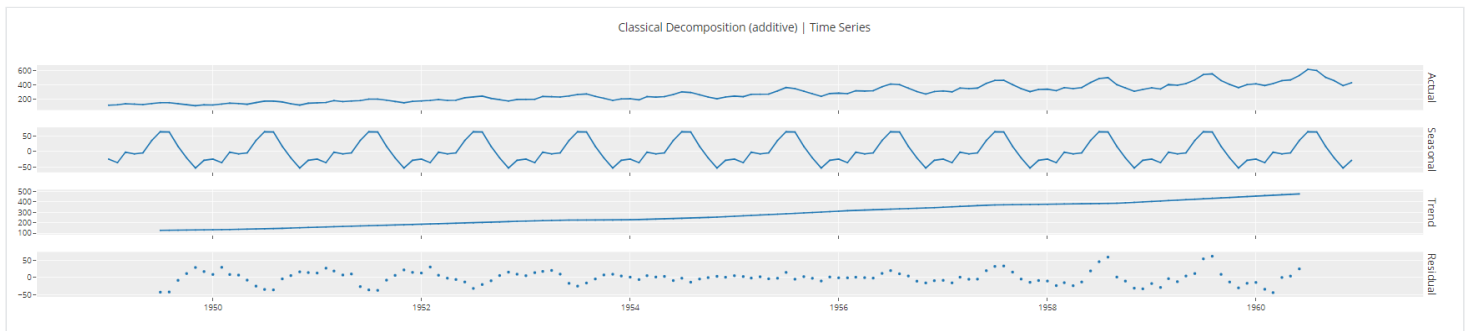
	Test	Test Name	Property	Setting	Value
2	Summary	Statistics	Median		265.5
3	Summary	Statistics	Standard Deviation		119.966
4	Summary	Statistics	Variance		14391.9
5	Summary	Statistics	Kurtosis		-0.364942
6	Summary	Statistics	Skewness		0.58316
7	Summary	Statistics	# Distinct Values		118
8	White Noise	Ljung-Box	Test Statistic	{'alpha': 0.05, 'K': 24}	1606.08
9	White Noise	Ljung-Box	Test Statistic	{'alpha': 0.05, 'K': 48}	1933.16
10	White Noise	Ljung-Box	p-value	{'alpha': 0.05, 'K': 24}	0
11	White Noise	Ljung-Box	p-value	{'alpha': 0.05, 'K': 48}	0
12	White Noise	Ljung-Box	White Noise	{'alpha': 0.05, 'K': 24}	False
13	White Noise	Ljung-Box	White Noise	{'alpha': 0.05, 'K': 48}	False
14	Stationarity	ADF	Stationarity	{'alpha': 0.05}	False
15	Stationarity	ADF	p-value	{'alpha': 0.05}	0.99188
16	Stationarity	ADF	Test Statistic	{'alpha': 0.05}	0.815369
17	Stationarity	ADF	Critical Value 1%	{'alpha': 0.05}	-3.48168

EN

Output from check\_stats function

If you are dealing with univariate time series, you can also decompose them into trend, seasonality, and error components for any further analysis.

```
plot_model(plot = 'decomp_classical')
```

[✦ Explain code](#)

Output from `plot_model` function

### ## Model Training and Selection ##

To train multiple models and select one out of many, we run just one line of code. `best = compare_models()`

	Model	MAE	RMSE	MAPE	SMAPE	MASE	RMSSE	TT (Sec)
exp_smooth	Exponential Smoothing	7.781	7.781	0.0172	0.0175	0.2435	0.2144	1.1600
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	10.9141	10.9141	0.0261	0.0258	0.3408	0.3002	0.1000
arima	ARIMA	11.5951	11.5951	0.0278	0.0274	0.362	0.3189	0.0867
auto_arima	Auto ARIMA	13.0789	13.0789	0.0309	0.0304	0.4086	0.3598	8.6767
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	14.5154	14.5154	0.0344	0.0337	0.4534	0.3993	0.3067
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseaso...	14.5979	14.5979	0.0344	0.0339	0.4561	0.4016	0.0300
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	15.0691	15.0691	0.0345	0.0342	0.4713	0.4149	0.2400
theta	Theta Forecaster	16.0228	16.0228	0.0346	0.0344	0.502	0.4418	1.5733
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Det...	16.3259	16.3259	0.037	0.0368	0.5107	0.4496	0.0833
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize...	16.9467	16.9467	0.0386	0.0383	0.5301	0.4667	0.0400
dt_cds_dt	Decision Tree w/ Cond. Deseasonalize & Detrending	18.1502	18.1502	0.0411	0.0406	0.5678	0.4999	0.0300
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detren...	21.0061	21.0061	0.0466	0.0461	0.6576	0.5788	0.0300
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	20.3369	20.3369	0.0469	0.0462	0.6358	0.5598	0.1033
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	23.2312	23.2312	0.0508	0.05	0.7276	0.6404	0.0500
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	23.9645	23.9645	0.053	0.0525	0.7502	0.6604	0.0300
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	24.2074	24.2074	0.0536	0.0531	0.7578	0.6671	0.0267
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	24.7737	24.7737	0.0549	0.0543	0.7755	0.6827	0.0300
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	24.7764	24.7764	0.0549	0.0543	0.7756	0.6828	0.0300
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasona...	26.7509	26.7509	0.0603	0.0579	0.8371	0.7368	0.0300
par_cds_dt	Passive Aggressive w/ Cond. Deseasonalize & De...	27.7606	27.7606	0.0684	0.0642	0.866	0.7628	0.0300
lar_cds_dt	Least Angular Regressor w/ Cond. Deseasonalize...	29.5795	29.5795	0.0657	0.0648	0.9259	0.8151	0.0333
snaive	Seasonal Naive Forecaster	42.3333	42.3333	0.0925	0.0972	1.3258	1.167	0.0200
polytrend	Polynomial Trend Forecaster	43.9519	43.9519	0.1031	0.0978	1.3735	1.2093	0.0167
croston	Croston	44.1736	44.1736	0.1039	0.0983	1.3802	1.2153	0.0133
naive	Naive Forecaster	72	72	0.159	0.1466	2.2543	1.9842	1.9400
grand_means	Grand Means Forecaster	175.955	175.955	0.3808	0.4751	5.5122	4.8516	0.0167

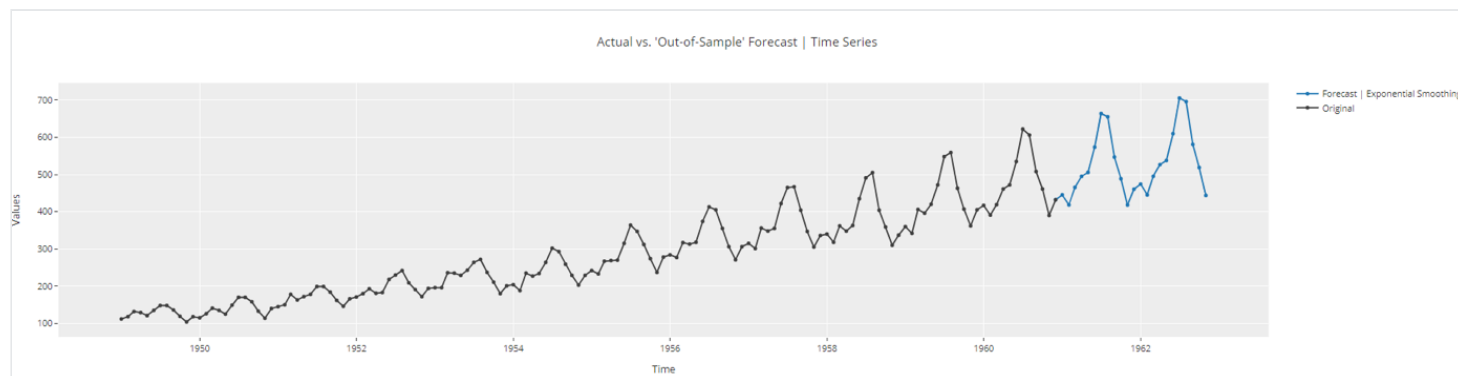
## Output from compare\_models function

PyCaret has trained over 25 models using the time-series appropriate cross-validation and has presented a list of models in order of higher to lower performance. The best model based on this experiment for this dataset happens to be Exponential Smoothing, with a mean absolute error of 7.781.

## Model Analysis

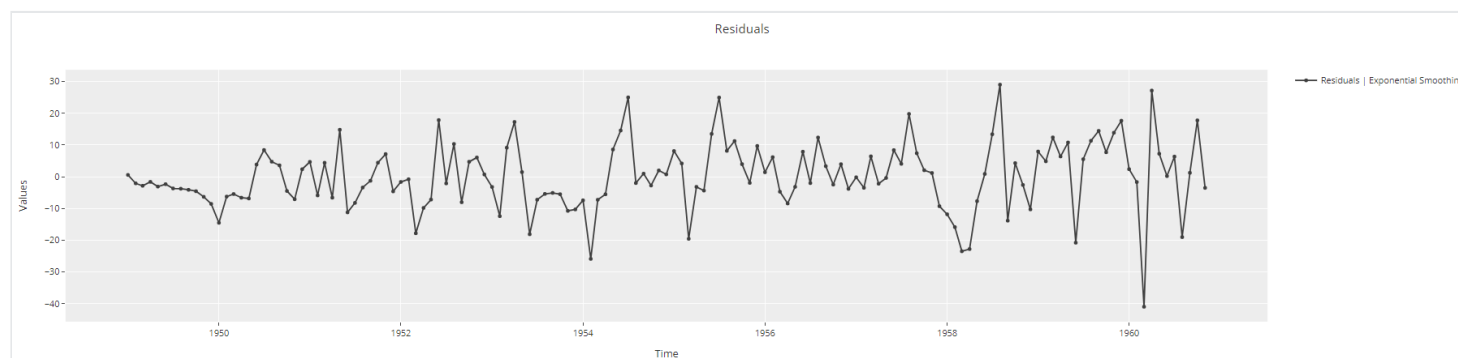
Before we deploy this model to generate predictions for the future, we can see a couple of analysis plots to comment on the quality of our model.

```
plot_model(best, plot = 'forecast', data_kwargs = {'fh': 24})
```

[✦ Explain code](#)

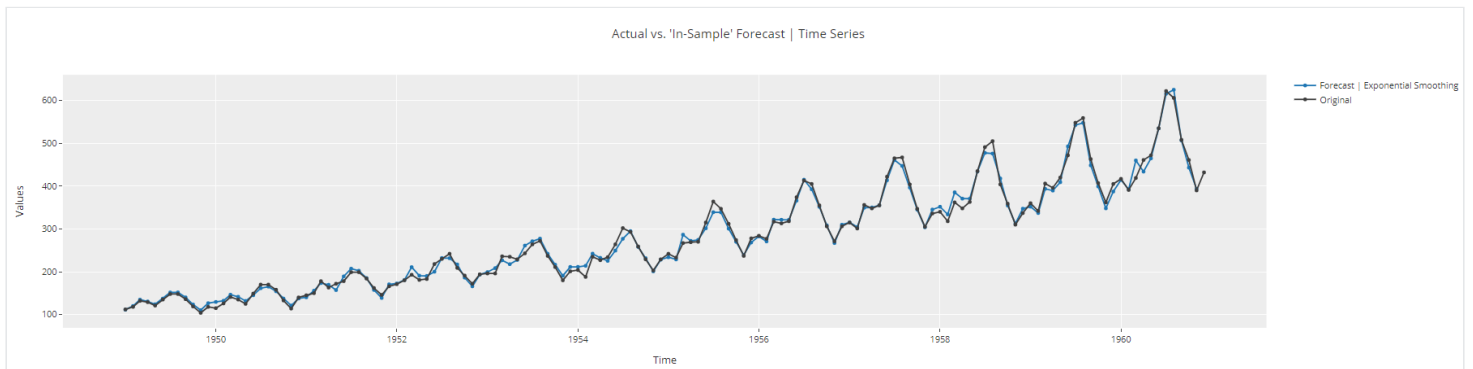
## Time Series Forecast Plot

```
plot_model(best, plot = 'residuals')
```

[✦ Explain code](#)

## Time-Series residuals plot

```
plot_model(best, plot = 'insample')
```

[✦ Explain code](#)

Time-Series insample plot

## Model Deployment

At this point, we are ready to finalize the model and save it for future use.

```
# finalize model
final_best = finalize_model(best)

# save model
final_best = finalize_model(best)
```

[✦ Explain code](#)

To load this file back and generate predictions on future data:






```
# load model
loaded_model = load_model('my_best_model')

# generate predictions for the next 48 months
predict_model(loaded_model, fh = 48)
```

[✦ Explain code](#)

Output truncated

## Learn more about PyCaret

 <a href="#">Tutorials</a>	Check out the official tutorials.
 <a href="#">Notebooks</a>	Example notebooks created by the community.
 <a href="#">Blog</a>	Tutorials and articles by contributors.
 <a href="#">Documentation</a>	The detailed API docs of PyCaret
 <a href="#">Video Tutorials</a>	Video tutorial of PyCaret from various events.

## Conclusion

Time-series forecasting is a very useful skill to learn. Many real-life problems are time-series in nature. Forecasting has a range of applications in various industries, with tons of practical applications including: weather forecasting, economic forecasting, healthcare forecasting, financial forecasting, retail forecasting, business forecasting, environmental studies, social studies, and more.

Basically, any historical data with consistent intervals can be analyzed with time series

analysis methods leading into a forecasting task that learns from the historical data and tries to predict the future. To conclude, there are three broad categories for time series forecasting:

- Statistical Models - Exponential smoothing, ARIMA, SARIMA, TBATS, etc.
- Machine Learning -Linear Regression, XGBoost, Random Forest, etc.
- Deep Learning - RNN, LSTM

If you would like to further build your knowledge and skills in time series forecasting, check out this amazing track of "[Time Series with Python](#)" by Datacamp. In this track, you'll learn how to manipulate time series data using pandas, work with statistical libraries including NumPy and statsmodels to analyze data, and develop your visualization skills using Matplotlib, SciPy, and seaborn. By the end of this track, you'll know how to forecast the future using ARIMA class models and generate predictions and insights using machine learning models.

## TOPICS

Data Science   Machine Learning