Time Series Forecasting with Supervised Machine Learning

Time series analysis and forecasting using supervised machine learning models

Unai López Ansoleaga · Following

Published in Towards Data Science · 6 min read · Jan 5, 2022

Photo by Aron Visuals on Unsplash

When I first saw a time series forecasting problem I was very confused. Until that moment, I just did some supervised learning predictions on tabular data so I didn't know how to do the forecastings if I didn't have the target values. Many of you may have face that problem so in this post I want to introduce a very powerful way of solving time series forecasting problems using supervised machine learning models instead of statistical models such as ARIMA, ARMA, MA, AR...

I decided to write about the machine learning approach of solving time series problems because I believe that these models are very versatile and

powerful and they're much more beginner friendly than other statistical approaches.

The code is available in the link below:

towards-data-science-posts-notebooks/Bike Sharing Demand Prediction.ipynb at master ·...

Contribute to unaiLopez/towards-data-science-posts-notebooks development by creating an account on GitHub.

github.com

Dataset

We are going to use Kaggle's Bike Sharing Demand competition dataset because it suites perfectly for this tutorial. You can download and read about the data in the link below:

Bike Sharing Demand

Forecast use of a city bikeshare system

www.kaggle.com

Time Series Analysis

Before using any model, it's important to do some time series analysis to understand the data. In this step we will check all variable types,

seasonalities, if the series is autoregressive or not, etc.

First of all, let's visualize the data:

```
df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

10886 rows × 12 columns

Bike sharing demand pandas dataframe

If we look to the screenshot above, we can see that the dataframe is 10886 rows long and 12 columns wide. The time series has an hourly period and our target variable will be the count column. This column is the sum of casual and registered columns but for the simplicity of the tutorial we'll remove casual and registered columns later and we'll just predict the count column. If you want to understand better the different variables of the data, you can check kaggle's link above and read some information about the bike sharing demand competition's dataset.

Now let's check dataframe's variable types:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

Variable types of the bike sharing demand pandas dataframe

All the dataframe's variables are correct except from the datetime column. This variable's type should be pandas' datetime instead of object. We'll change it later.

Let's see some statistical data about our dataframe's columns:

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
season	10886.0	2.506614	1.116174	1.00	2.0000	3.000	4.0000	4.0000
holiday	10886.0	0.028569	0.166599	0.00	0.0000	0.000	0.0000	1.0000
workingday	10886.0	0.680875	0.466159	0.00	0.0000	1.000	1.0000	1.0000
weather	10886.0	1.418427	0.633839	1.00	1.0000	1.000	2.0000	4.0000
temp	10886.0	20.230860	7.791590	0.82	13.9400	20.500	26.2400	41.0000
atemp	10886.0	23.655084	8.474601	0.76	16.6650	24.240	31.0600	45.4550
humidity	10886.0	61.886460	19.245033	0.00	47.0000	62.000	77.0000	100.0000
windspeed	10886.0	12.799395	8.164537	0.00	7.0015	12.998	16.9979	56.9969
casual	10886.0	36.021955	49.960477	0.00	4.0000	17.000	49.0000	367.0000
registered	10886.0	155.552177	151.039033	0.00	36.0000	118.000	222.0000	886.0000
count	10886.0	191.574132	181.144454	1.00	42.0000	145.000	284.0000	977.0000

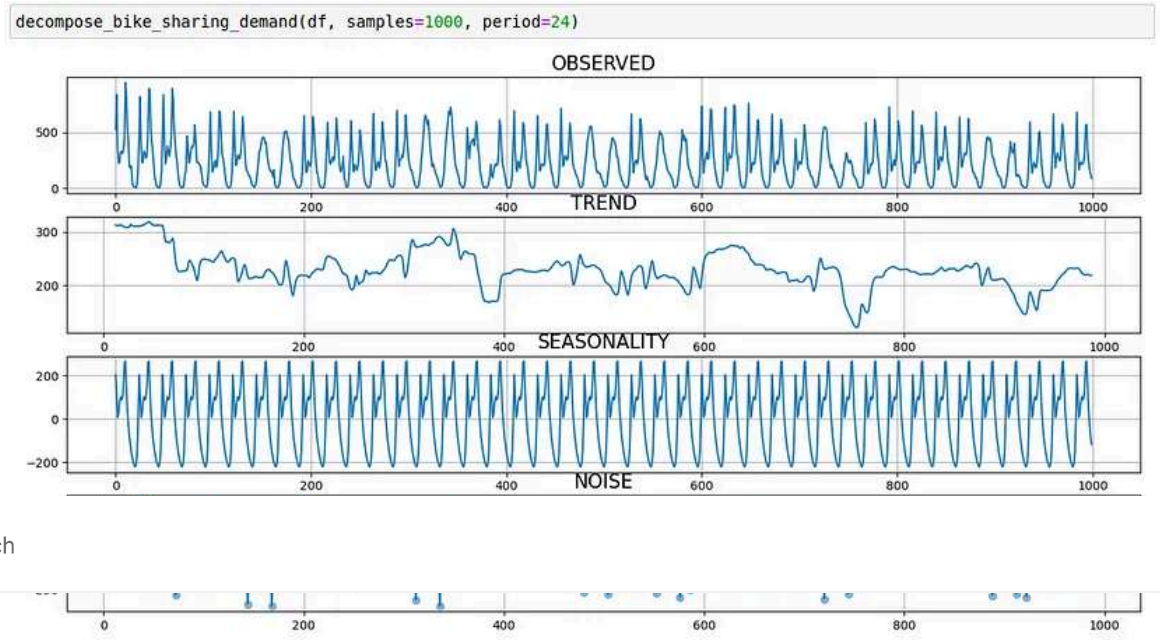
Statistical data about bike sharing demand pandas dataframe's columns

The data shown in the screenshot can be interesting for extracting some insights from our data. Let's continue with our data analysis looking for seasonalities and trends.

We can easily look for seasonalities using statsmodels' seasonal_decompose function. This function will decompose our time series into trend, seasonality and noise:

```python
def decompose_bike_sharing_demand(df, share_type='count', samples=250, period=24):
    if samples == 'all':
        #decomposing all time series timestamps
        res = seasonal_decompose(df[share_type].values, period=period)
    else:
        #decomposing a sample of the time series
        res = seasonal_decompose(df[share_type].values[-samples:], period=period)

    observed = res.observed
    trend = res.trend
    seasonal = res.seasonal
    residual = res.resid

    #plot the complete time series
    fig, axs = plt.subplots(4, figsize=(16,8))
    axs[0].set_title('OBSERVED', fontsize=16)
    axs[0].plot(observed)
    axs[0].grid()

    #plot the trend of the time series
    axs[1].set_title('TREND', fontsize=16)
    axs[1].plot(trend)
    axs[1].grid()

    #plot the seasonality of the time series. Period=24 daily seasonality | Period=24*7 weekly se
    axs[2].set_title('SEASONALITY', fontsize=16)
    axs[2].plot(seasonal)
    axs[2].grid()

    #plot the noise of the time series
    axs[3].set_title('NOISE', fontsize=16)
    axs[3].plot(residual)
    axs[3].scatter(y=residual, x=range(len(residual)), alpha=0.5)
    axs[3].grid()

    plt.show()
```

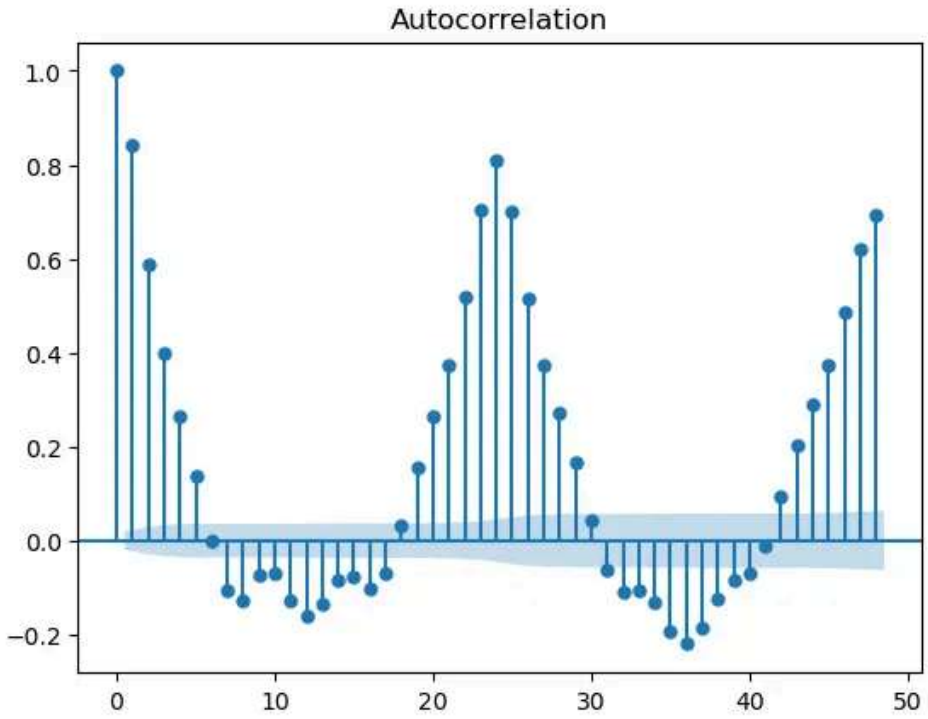
Now, we are going to use the custom function above to decompose 1000 hours of our time series with a daily seasonality (period=24 hours):



Time series decomposition using our custom function and statsmodels package

There we go! We can extract a lot of insights from the graphs above. If we look closely, we can see a clear daily seasonal pattern with 2 peeks and a valley between them. Despite this pattern, there is still a lot of noise that is not explained by our daily seasonality so we will try to model this noise using other variables in the dataset and some feature engineering. But before that, let's see if our data is autoregressive:

```
plot_acf(df['count'].values, lags=48)
plt.show()
```



Plot of the autocorrelation of our time series with a 48 hours lag

After plotting this autocorrelation graph, we can say with a high confidence that our data is autoregressive and that we can improve our model's performance using lags. In other words, the bike sharing demand can be explained using previous hour's and day's values.

Time Series Forecasting

After understanding the data and getting some insights, we're ready to start modelling and forecasting the bike sharing demand per hour. In this post,

we are going to forecast 1 week bike sharing demand. This means that if a week has 7 days and every day has 24 hours, we are going to predict the bike sharing demand for the next 168 hours.

We're going to use Microsoft's Light Gradient Boosting Machine model. This model was developed by Microsoft and it beats the standard Extreme Gradient Boosting (XGBoost) in training speed and sometimes in accuracy. Even though I use this machine learning model, you can use whatever model you want within scikit-learn regressors or beyond.

In this approach, we will extract new features from our timestamp and we will use this new features to perform a multi-output regression:

Let’s see the result of this feature engineering process:

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	hour	day	month
2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	16	0	1	1
2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	40	1	1	1
2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	32	2	1	1
2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	13	3	1	1
2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	1	4	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	336	19	19	12
2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	241	20	19	12
2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	168	21	19	12
2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	129	22	19	12
2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	88	23	19	12

10886 rows × 12 columns

Dataset with new features extracted from the date

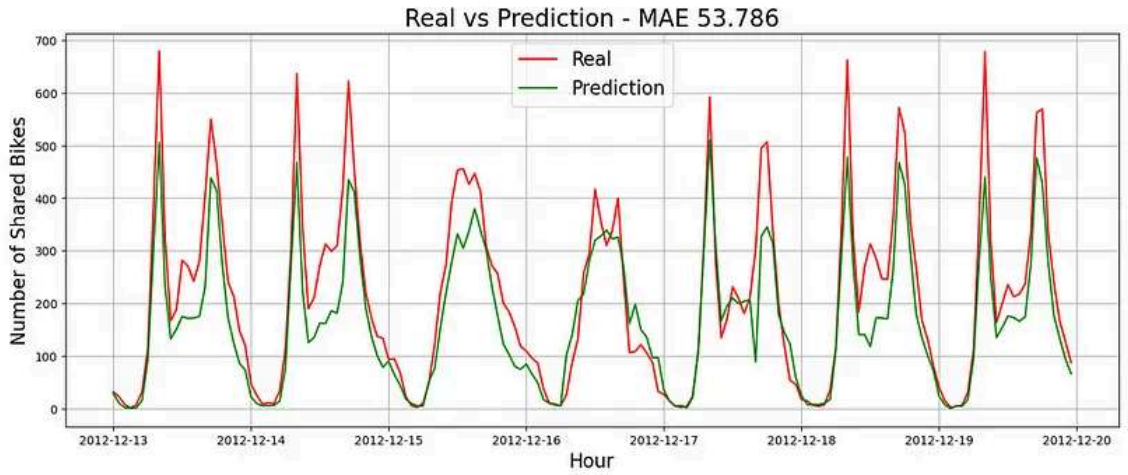
Once we have our dataset with the regressors we are going to use, let's build a custom function for predicting our horizon:

Custom training function and prediction plot

Without Lags

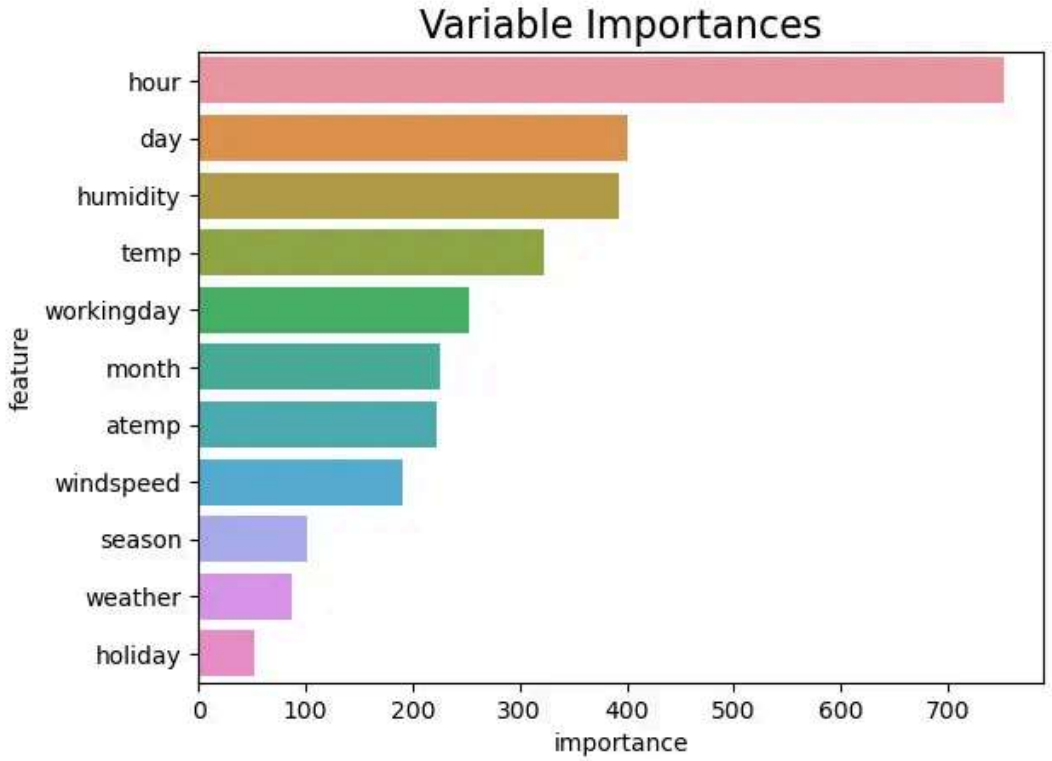
Once we have defined our custom function, we are going to use it and check the results of the model:

```
train_time_series_with_folds(df)
```



Model predictions without lags

Without using any lagged variables we got a MAE of 53. That's not bad at all!



Model variable importances without lags

If we check the importance of the variables according to our model, the hour and the day seem to be quite important so we can say that our features created by our feature engineering process are very helpful.

But, can we create more features and improve our performance even more?

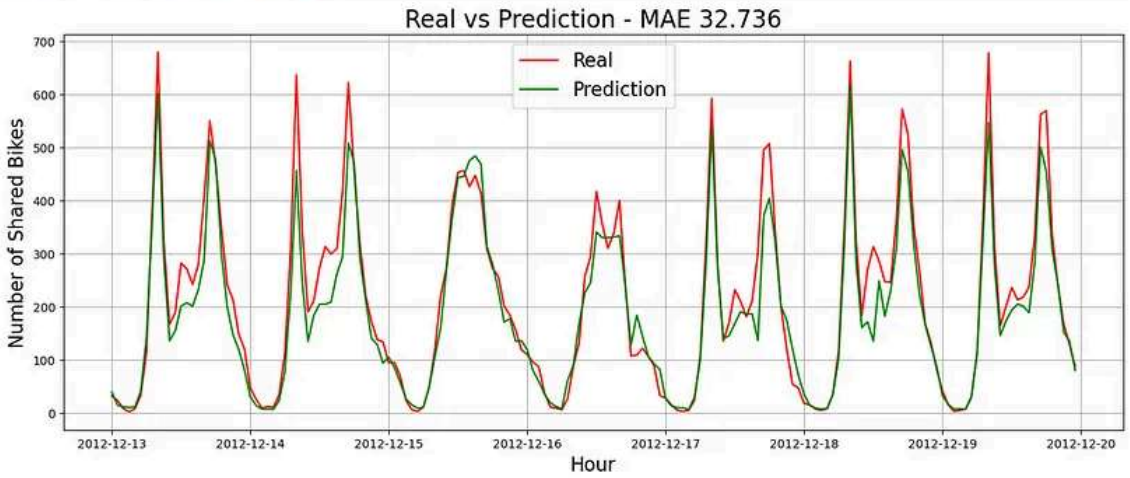
With Lags

As we said before, the data seems to be very autocorrelated so let's try adding lags and let's see if this new feature improves the model's performance:

Creating a new lag feature

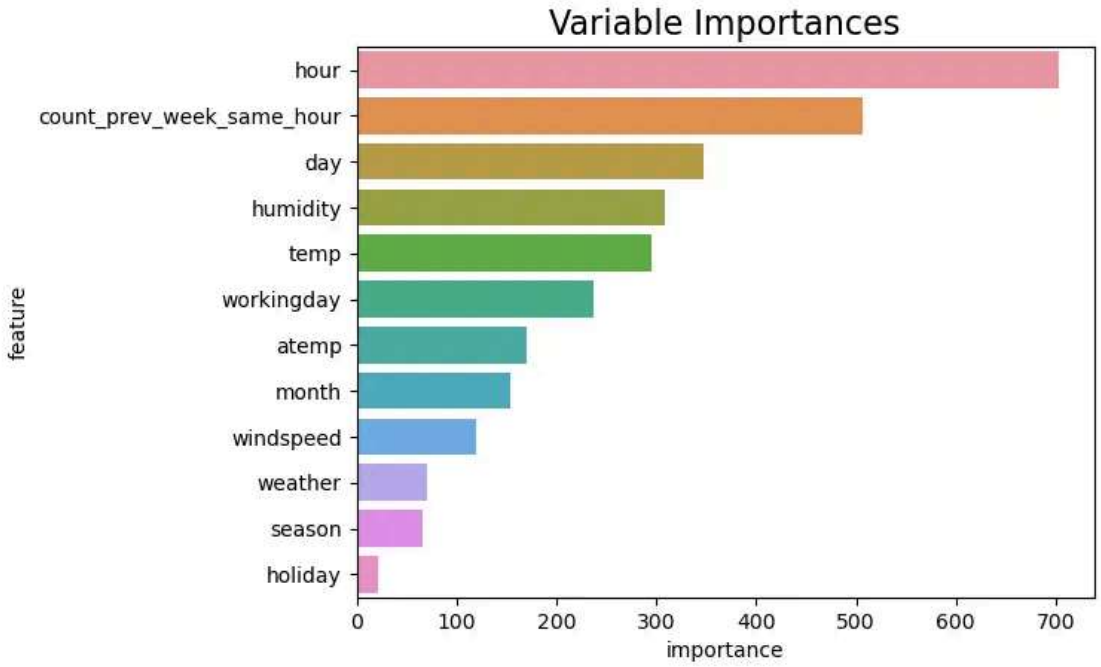
If we look to the code above, we can see how panda's shift function can help a lot on creating lagged features. Let's see if this new feature can help us improving our model's performance:

```
train_time_series_with_folds(df)
```



Model predictions with lags for 1 week (168 hours)

Wow! The model accuracy has improved a lot just adding a new lag feature. It went from a mean absolute error of 53 to 32. That's about a 40% improvement comparing with the model without lagged features!



Model variable importances with lags

If we look closely to the variable importances, the lagged feature (count_prev_week_same_hour) seems to be very useful for predicting our target. Feature engineering is great!

Conclusion

As we saw in this post, supervised machine learning models can be very versatile and even better than other statistical approaches for time series forecasting in some cases. That said, we can conclude that these models are very powerful for time series forecasting. In spite of their power, they require some feature engineering to make them work, otherwise, their performance will be poor.

If you like my content you can check my other posts too:

Unleash the Power of Scikit-learn's Pipelines

An intermediate guide to scikit-learn's pipelines

towardsdatascience.com

Introduction to Scikit-learn's Pipelines

Building an end-to-end machine learning pipeline using scikit-learn

towardsdatascience.com

How to Detect, Handle and Visualize Outliers

When I first started developing data science projects, I didn't care about data visualization nor outlier detection, I...

towardsdatascience.com

References

Welcome to LightGBM's documentation! - LightGBM 3.3.1.99 documentation

Edit description

lightgbm.readthedocs.io

Python | Pandas dataframe.shift() - GeeksforGeeks

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python...

www.geeksforgeeks.org

Kaggle: Your Machine Learning and Data Science Community

Kaggle is the world's largest data science community with powerful tools and resources to help you achieve your data...

www.kaggle.com

Time Series Analysis | Time Series Forecasting | Machine Learning | Feature Engineering | Data Visualization

tds

Published in Towards Data Science

772K Followers · Last published just now

Following

Your home for data science and AI. The world's leading publication for data science, data analytics, data engineering, machine learning, and artificial intelligence professionals.

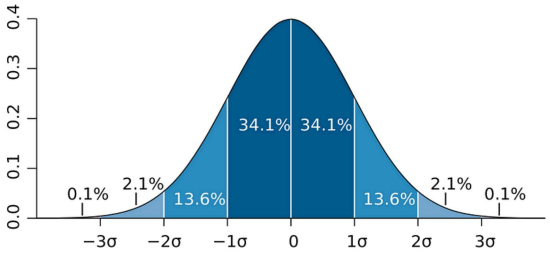
Written by Unai López Ansoleaga

151 Followers · 8 Following

Following

Data Scientist

More from Unai López Ansoleaga and Towards Data Science



tds In Towards Data Science by Unai López Ansoleaga

How to Detect, Handle and Visualize Outliers

When I first started developing data science projects, I didn't care about data visualizatio...

Jun 6, 2021 23 1

tds In Towards Data Science by Cassie Kozyrkov

The Name That Broke ChatGPT: Who is David Mayer?

AI, privacy, human bias, prompting, the future of content, and how to hack a chatbot

1d ago 452 8

In Towards Data Science by Stephanie Kirmer

The Cultural Impact of AI Generated Content: Part 1

What happens when AI generated media becomes ubiquitous in our lives? How does...

1d ago 233 5

In Towards Data Science by Unai López Ansoleaga

Unleash the Power of Scikit-learn's Pipelines

An intermediate guide to scikit-learn's pipelines

Dec 19, 2021 51 2

See all from Unai López Ansoleaga

See all from Towards Data Science

Recommended from Medium

In Towards Data Science by Bradley Stephen Shaw

False Prophet: Feature Engineering for a Homemade Time Series...

Building on ideas from Meta's Prophet package to create powerful features for time...

Oct 14, 2023 230 2

George Kamtziridis

Enhancing Time Series Forecasting with XGBoost: Incorporating...

Welcome to the third article on predicting energy consumption demands for the city of...

Sep 25 30 1

Lists

Predictive Modeling w/ Python

20 stories · 1700 saves

Practical Guides to Machine Learning

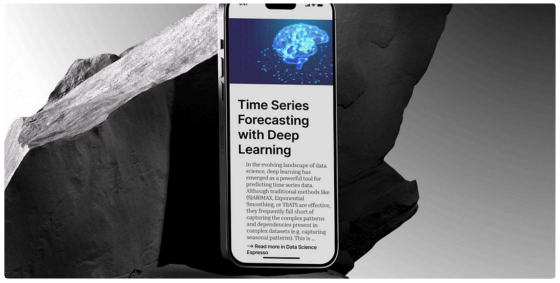
10 stories · 2068 saves

Natural Language Processing

1842 stories · 1466 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 518 saves

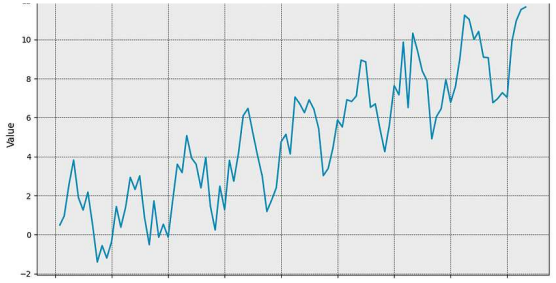


In Python in Plain English by Sarah Lea

Beginner's Guide to Advanced Techniques for Time Series...

In the evolving landscape of data science, deep learning has emerged as a powerful to...

Jul 22 44



In Stackademic by Ganesh Bajaj

Time Series Analysis: Interpretation of ACF and PACF...

Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots are powerful...

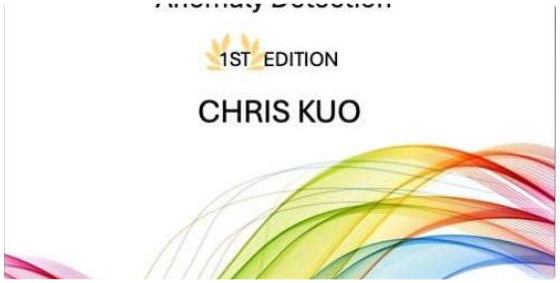
Sep 24 50

Irina (Xinli) Yu, Ph.D.

Mastering Time Series Forecasting with ARIMA Models

Time series forecasting is a critical component in many domains, including...

Jun 7 2



In Dataman in AI by Chris Kuo/Dr. Dataman

Temporal Fusion Transformer for Interpretable Time Series...

Sample eBook chapters (free): https://github.com/dataman-git/modern-...

Apr 18 381 3

See more recommendations