A Comparison of Automated Time Series Forecasting Tools for Smart Cities

Pedro José Pereira[1], Nuno Costa[1], Margarida Barros[1], Paulo Cortez[1], Dalila Durães[2], António Silva[2], and José Machado[2]

[1] ALGORITMI Centre, Dep. Information Systems, University of Minho, Guimarães, Portugal
pedro.pereira@dsi.uminho.pt, a89167@alunos.uminho.pt, a89177@alunos.uminho.pt, pcortez@dsi.uminho.pt
[2] ALGORITMI Centre, University of Minho, Braga, Portugal
dalila.duraes@algoritmi.uminho.pt, asilva@algoritmi.uminho.pt, jmac@di.uminho.pt

Abstract. Most smart city sensors generate time series records and forecasting such data can provide valuable insights for citizens and city managers. Within this context, the adoption of Automated Time Series Forecasting (AutoTSF) tools is a key issue, since it facilitates the design and deployment of multiple TSF models. In this work, we adapt and compare eight recent AutoTSF tools (Pmdarima, Prophet, Ludwig, DeepAR, TFT, FEDOT, AutoTs and Sktime) using nine freely available time series that can be related with the smart city concept (e.g., temperature, energy consumption, city traffic). An extensive experimentation was carried out by using a realistic rolling window with several training and testing iterations. Also, the AutoTSF tools were evaluated by considering both the predictive performances and required computational effort. Overall, the FEDOT tool presented the best overall performance.

Keywords: Automated Machine Learning · Time Series Forecasting · Smart cities.

1 Introduction

Smart cities collect a huge variety of data variables by using edge sensors (e.g., traffic cameras, meteorological instruments). Since each sensor often performs a regular collection of digital records over time, the collected data tends to assume a time series format. Under this context, Time Series Forecasting (TSF) is a fundamental component. Indeed, TSF can be used to provide valuable insights for city managers and users, allowing to optimize city resources and to support plans. Moreover, TSF can also help to detect anomalies by comparing the real observations with the values predicted by the forecasting algorithms [5]. In effect, several recent studies have applied TSF to smart cities issues, such as: weather conditions [13], city traffic [14] and energy consumption [7].

There are two main TSF approaches used by the related works: Deep Learning (DL), for instance by adopting the Long Short-Term Memory(LSTM) architecture; and AutoRegressive methodologies, such as assumed by the AutoRegressive Integrated Moving Average (ARIMA) methodology. ARIMA was proposed in the 70s [2]. Due to its success, several extensions have been proposed and evaluated under the smart cities context [14]. Yet, the ARIMA is a rather rigid model that presents limitations when modeling complex nonlinear relationships. More recently, several studies adopted TSF DL approaches for the smart cities domain, including Recurrent Neural Networks (RNNs) for vehicle parking occupancy [3] and LSTMs for modeling vehicle traffic flow [20].

Nowadays, Machine Learning (ML) is widely used by organizations and individuals. Under this context, there is an increasing focus towards the usage of Automated ML (AutoML) and Automated DL (AutoDL) tools[3] [8]. These tools allow non-experts to more easily design and deploy ML algorithms that are capable of providing value in diverse application domains. As described in [8], there is an increasing number of research works that propose and compare AutoML and AutoDL tools for supervised learning tasks (classification or regression). However, less research and empirical studies have been devoted to the Automated TSF (AutoTSF) task. In [17], a systematic review was performed by comparing 40 Python packages for time series analysis. The packages were analyzed in terms of their functionalities, such as performed tasks (e.g., forecasting, anomaly detection). Yet, the review did not perform any kind of empirical comparison. More recently, the FEDOT AutoTSF tool was empirically compared against the Facebook Prophet [19] and AutoTS [21] tools, outperforming both in terms of predictive performances for a set of 12 financial time series [15].

In this paper, we perform a robust benchmark of eight recent AutoTSF tools (a value that is substantially higher than what was executed in [15]), namely: Pmdarima, Prophet, Ludwig (an AutoDL that is adapted here for TSF), DeepAR, TFT, FEDOT, AutoTs and Sktime. To test the tools, nine time series that can be associated with the smart cities context were used. Within our knowledge, this is the first study addressing the AutoTSF topic within the smart city application domain. The comparison includes the adoption of a robust rolling window evaluation, which performs several training and testing iterations over time. For each iteration, the tools are analyzed in terms of two criteria: predictive performances, set in terms of the Normalized Mean Absolute Error (NMAE); and computational effort, set in terms of training and inference times (measured in seconds and milliseconds).

2 Materials and Methods

2.1 Time Series Data

A time series represents a collection of time ordered observations $(y_1, y_2, ..., y_t)$, each recorded at a specific time (t) [6]. This work addresses multi-step ahead

[3] Also known as Neural Architecture Search (NAS).

forecasts, meaning that at time t (the last known value) from $t+1$ to $t+H$ ahead forecasts are performed (H is known as the horizon).

This study considers time series that can be related with the smart cities context, reflecting three city phenomena: meteorology[4], energy consumption[5] and city traffic[6]. For each phenomena, we retrieved three different time series from the Kaggle platform (Table 1). The meteorological data is relative to the maximum daily temperature from three cities (Porto, Lisbon and Madrid), collected from 2008 to 2020. The energy consumption hourly data, measured in Megawatts, was collected from 2004 to 2018. In order to produce a similar time series length (as for the meteorology case), the data was aggregated on a daily basis by summing the hourly values. Each series was recorded by a different North American energy company: American Electric Power (AEP), Commonwealth Edison (COMED); and PJM East Region (PJME). Regarding the traffic data, the series correspond to the hourly number of vehicles passing by three different junctions from a city of the United States of America (USA). The hourly time scale was preserved in order to maintain a series length similar to the meteorology and energy data.

Table 1: Summary of the selected time series (L – series length, K – seasonal period, W – window size, S – step, H – horizon).

Context	**Target**	**Series**	**Location (years)**	L	K	W	S	H
Meteorology	Daily max. temperature (in C)	porto	Porto (2008-2020)	3946	365	1825	105	7
		lisbon	Lisbon (2008-2020)	3946	365	1825	105	7
		madrid	Madrid (2008-2020)	3946	365	1825	105	7
Energy	Daily consumption (in MW)	AEP	USA (2004-2018)	5055	7	1825	161	7
		COMED	USA (2011-2018)	2772	7	1825	47	7
		PJME	USA (2002-2018)	6059	7	1825	211	7
Traffic	Hourly no. of vehicles (in units)	junction1	USA (Jan. to June, 2017)	4344	24	2160	108	24
		junction2	USA (Jan. to June, 2017)	4344	24	2160	108	24
		junction3	USA (Jan. to June, 2017)	4344	24	2160	108	24

For the daily time series (meteorology and energy related) the prediction horizon was set to one week ($H = 7$), while for the hourly vehicle traffic the horizon was set to one day ($H = 24$). To set the seasonal period (K) we followed the methodology adopted in [5], which assumes an inspection of the observed values and its autocorrelations. The visual inspection confirmed seasonal periods of $K = 365$ (one year) for the meteorological data, $K = 7$ (one week) for the energy data and $K = 24$ (one day) for the traffic series. As shown in Figure 1, these K values correspond to higher autocorrelation values within the neighbor-

[4] https://www.kaggle.com/datasets/luisvivas/spain-portugal-weather
[5] https://www.kaggle.com/robikscube/hourly-energy-consumption
[6] https://www.kaggle.com/fedesoriano/traffic-prediction-dataset

hood of a time lag and its multiple values (e.g., {12,24} time lags for the AEP dataset). It should be noted that the value of K is often known apriori by the domain user. Also, the K parameter is only required by the Ludwig tool, to set the number of time lags used by the searched autoregressive models.

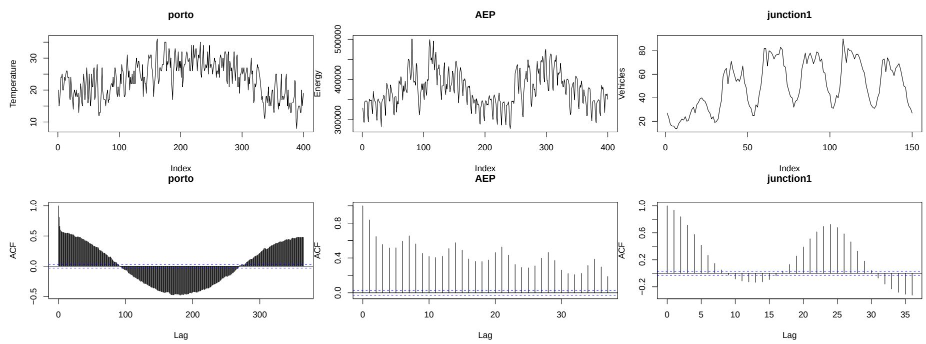

Fig. 1: Examples of time series (top plots) and their autocorrelations (bottom graphs).

2.2 AutoTSF Methods

We compare eight open-source Python AutoTSF tools, summarized in Table 2 in terms of: **Name**, publication **Year**, bibliographic **Ref**erence, automated **Type**, **Opt**imization method used for the TSF model search (when known), domain **License** and **Training Mode**. We selected: three AutoDL tools – Uber Ludwig [12]; Temporal Fusion Transformer (TFT), based on LSTMs [10]; and DeepAR, based on RNNs [16]); three native AutoTSF tools – FEDOT [15]; Auto Time-Series (AutoTS) [21]; and Sktime [11]); and two recent implementations that assume a single TSF model: Auto-Arima [18] and Facebook Prophet [19]). To maintain a fair comparison, whenever possible the tools were executed the default parameters, thus corresponding to a natural choice for an non-expert user. Furthermore, to reduce the computational effort, we limited the models execution time, either by setting a time limitation, selecting a fast execution option or performing the model and hyperparamenter selection only during the first rolling window iteration (Update train mode). This last option assumes fixing the selected model after the first iteration and then only updating it (fit to newer training data) in the remaining iterations. The selected AutoTSF tools are:

1. *Pmdarima*: a recent Python module that implements Auto-ARIMA [18], an extension that automatically chooses the best ARIMA model [1].
2. *Prophet*: Facebook's additive TSF model that is capable to deal with non-linearity [19]. We used the `prophet` Python package.
3. *Ludwig*: Uber's open source AutoDL software that uses a DL architecture called Encoder-Combiner-Decoder [12]. The tool is implemented via the

Table 2: Summary of the analyzed AutoTSF tools.

Name	**Year**	**Ref.**	**Type**	**Task**	**Opt.**	**License**	**Train Mode**
Pmdarima	2017	[18]	Auto-Arima	TSF	-	MIT	Train
Prophet	2017	[19]	AutoProphet	TSF	-	MIT	Train
Ludwig	2019	[12]	AutoDL	Reg.	-	Apache 2.0	Train
DeepAR	2020	[16]	AutoRNN	TSF	-	Apache 2.0	Train
TFT	2021	[10]	AutoLSTM	TSF	-	Apache 2.0	Train
FEDOT	2022	[15]	AutoTSF	TSF	EA	BSD-3-Clause	Update
AutoTs	2022	[21]	AutoTSF	TSF	GA	MIT	Train
Sktime	2022	[11]	AutoTSF	TSF	GS	BSD-3-Clause	Update

`ludwig` Python package. Ludwig was adapted for TSF by converting a time series into a tabular format by using a sliding time window. In particular, autoregressive models are assumed, where $\hat{y}_t = f(y_{t-k_1}, ..., y_{t-k_n})$ is the predicted value, f denotes the learned regression function and $\{k_1, ..., k_n\}$ is the set of time lags used by the sliding window to generate the regression inputs. Similarly to what was proposed in [5], the selected time lags are based on seasonal period (K) heuristics: temperature series – $\{1, 2, 3, 4, 5, 6, 365, 366\}$ (yearly seasonality); energy data – $\{1, 2, ..., 7, 8\}$ (weekly seasonality); and traffic – $\{1, 2, ..., 24\}$ (daily seasonality). To generate multi-step ahead forests, an iterative input feedback of the previous predictions is adopted [5].

4. *DeepAR*: a methodology for probabilistic forecasting that uses autoregressive Recurrent Neural Networks (RNNs) [16]. RNNs do not require sliding windows, since the model is capable of internally memorizing temporal sequences. DeepAR is implemented using the `Gluonts` Python module.
5. *TFT*: similarly to DeepAR, TFT is an AutoDL tool yet with a particular focus in LSTM RNN [10]. This tool was also implemented using `Gluonts` Python module.
6. *FEDOT*: an approach to design ML pipelines based on an Evolutionary Algorithms (EA) and that can be applied to different ML tasks, including TSF [15]. We used the `fedot` Python package with the time series preset setup. Furthermore, the maximum model training time was set to 15 minutes. Since this AutoTSF method is computationally expensive, when compared with the other AutoTSF approaches, the tool was set with the Update training mode.
7. *AutoTS*: an AutoTSF tool based on Genetic Algorithms (GA) [21]. Regarding its implementation, we used the `autots` Python package, assuming the "superfast" model option, which includes a Generalized Least Squares learning and multiple Naive models.

8. *Sktime*: is an unified Grid Search (GS) framework for ML with time series capabilities [11]. We used 4 different models: Theta, Naive, Auto-ARIMA and Auto-ETS. Similarly to FEDOT, the Update mode is assumed.

2.3 Evaluation

In order to perform a robust comparison, we applied a realistic rolling window scheme with a total of $U = 20$ training and testing iterations over time [4]. In each iteration, models are fitted using a training set of a fixed window size W and then performs up to H-ahead predictions. The first iteration assumes that the oldest W data observations are used to fit the TSF model. In the next iteration, the window is rolled by assuming a step size of S, where the S oldest values are discarded from the training set, which is then updated with S newer observations, and so on. In order to set the rolling window parameters, we first adopted a fixed W value for each series type (five years of data for the meteorological and energy series; 90 days of data for the traffic series). Then, the rolling window step was defined as $S = (L - (W + H - 1))/U$.

All selected models were evaluated both in terms of their predictive performances and computational effort. For evaluating the predictive performance, we used the Normalized Mean Absolute Error (NMAE), computed according to $NMAE = \frac{MAE}{y_{max} - y_{min}}$, with $MAE = \frac{\sum_{i=1}^{H} |y_{t+i} - \hat{y}_{t+i}|}{H}$, where y_{t+i} denotes the target values, $\hat{y}_{t+i}$ the predictions (made at time t for the i-th ahead step), and y_{min} and y_{max} the series minimum and maximum values, respectively. The NMAE is a scale independent measure. In terms of computational effort, we measured both the training time, in seconds, and the inference time (when performing one multi-step ahead prediction), in milliseconds. For each time series, we aggregated the results for all 20 iterations by using the median for NMAE (which is less sensitive to outliers) and the mean for the training and inference times (one multi-step prediction). The Wilcoxon non parametric statistic [9] is used to check if paired NMAE differences are statistically significant (p-value below 0.05).

3 Results

All experiments were executed using Python code that was run in an Linux Intel Xeon 2.10GHz server. Table 3 presents obtained results for the meteorological datasets. For the porto series, Ludwig obtained the best predictive performance with 9.56% NMAE, followed by Pmdarima (9.87 %) and FEDOT (10.29 %). Regarding the lisbon series, Pmdarima achieved the best predictive performance, while FEDOT was the second best AutoTS model, followed by Sktime. For the madrid dataset, FEDOT tool achieved the best predictions, followed by Sktime and then Pmdarima. On the other hand, Sktime obtained the worst NMAE value for the porto series and AutoTS produced the worst predictions for the lisbon and madrid data. Regarding the computational effort, Prophet presents the fastest training process but also the slowest prediction times. As for Pmdarima, it corresponds to the fastest TSF model to perform predictions, achieving the lowest inference times for the same datasets.

Table 3: Comparison results for the meteorological data (best values are in **bold**).

Time series	**ML Model**	**NMAE** (in %)	**Train Time (s)**	**Prediction Time (ms)**
porto	Pmdarima	9.87	87.74	**0.47**
	Prophet	13.76	**0.38**	116.33
	Ludwig	***9.56**	5.84	88.74
	DeepAR	10.66	118.13	7.66
	TFT	11.98	291.59	4.74
	FEDOT	10.29	150.80	1.81
	AutoTS	11.34	9.79	7.87
	Sktime	18.67	0.51	4.05
lisbon	Pmdarima	†**5.49**	86.61	**0.39**
	Prophet	7.99	**0.22**	115.95
	Ludwig	9.04	5.37	74.96
	DeepAR	8.55	118.44	7.65
	TFT	5.94	291.89	4.75
	FEDOT	5.68	209.46	1.72
	AutoTS	14.76	10.54	9.85
	Sktime	5.80	0.83	1.84
madrid	Pmdarima	7.26	90.88	**0.37**
	Prophet	13.35	**0.25**	112.84
	Ludwig	7.99	5.88	74.14
	DeepAR	7.70	118.72	7.87
	TFT	8.64	286.85	4.72
	FEDOT	$^{\diamond}$**6.31**	150.81	1.53
	AutoTS	22.56	12.85	9.60
	Sktime	6.96	0.49	1.85

$\star$ – Statistically significant under a paired comparison with Sktime.
$\dagger$ – Statistically significant under a paired comparison with Prophet and Sktime.
$\diamond$ – Statistically significant under a paired comparison with Prophet and AutoTS.

The energy consumption results are presented in Table 4. The DeepAR model achieved the lowest NMAE values for AEP series and COMED while AutoTS selected a TSF model that obtained the lowest NMAE for the PJME data (3.44%). As for Sktime, it presented the worst predictive performances. Similarly to the results obtained with the meteorological data, Prophet is the fastest model in the training stage (less than 1 s), while Pmdarima is the fastest to perform predictions (around 0.4 ms).

Table 4: Comparison results for the energy data (best values in **bold**).

Time series	ML Model	**NMAE** (in %)	**Train Time (s)**	**Prediction Time (ms)**
AEP	Pmdarima	4.52	128.86	**0.40**
	Prophet	3.93	**0.47**	113.86
	Ludwig	3.75	5.55	77.42
	DeepAR	⋆**3.47**	120.58	7.83
	TFT	3.96	292.24	4.65
	FEDOT	3.83	208.16	1.83
	AutoTS	4.83	8.28	6.25
	Sktime	5.30	3.05	3.52
COMED	Pmdarima	4.75	99.86	**0.39**
	Prophet	4.36	**0.39**	115.82
	Ludwig	3.75	6.28	78.23
	DeepAR	**2.45**	119.80	7.74
	TFT	3.96	292.24	4.72
	FEDOT	3.77	150.86	1.47
	AutoTS	4.40	9.41	8.08
	Sktime	6.91	1.29	4.11
PJME	Pmdarima	3.75	97.65	**0.40**
	Prophet	3.62	**0.45**	115.33
	Ludwig	4.63	5.84	78.49
	DeepAR	4.48	119.48	8.45
	TFT	4.06	292.19	4.62
	FEDOT	4.12	207.77	1.40
	AutoTS	⋆**3.44**	8.19	6.64
	Sktime	7.31	1.34	4.15

⋆ – Statistically significant under a paired comparison with Sktime.

Table 5 shows the city traffic results. FEDOT obtained the best predictive performance for two of the three series (junction1 and junction2), while DeepAR achieved the lowest NMAE value for the junction3 data. Similarly to the previous obtained results, the Sktime TSF model presented the highest prediction errors. In terms of the computation effort, the previously detected behavior is repeated (e.g., Prophet is the fastest training method).

The overall results are shown in Table 6 in terms of the tool NMAE median and computational effort average results when considering all 9 time series. The best median predictive performance is obtained by the FEDOT (4.58%), followed by Prophet (5.31%) and DeepAR (5.91 %). The obtained results are consistent with the study performed in [15], since in this work FEDOT also outperforms the Prophet and AutoTS tools in terms of the obtained median NMAE value. Moreover, FEDOT produces the lowest NMAE values for 3 out of 9 analyzed

Table 5: Comparison results for the traffic data (best values in **bold**).

Time series	ML Model	NMAE (in %)	Train Time (s)	Prediction Time (ms)
junction1	Pmdarima	12.91	98.10	**0.21**
	Prophet	5.52	**0.39**	34.13
	Ludwig	8.17	8.35	28.28
	DeepAR	6.26	286.20	8.86
	TFT	6.29	711.38	2.25
	FEDOT	***3.44**	208.22	0.40
	AutoTS	6.54	13.51	4.49
	Sktime	14.36	1.28	2.59
junction2	Pmdarima	11.00	145.47	**0.22**
	Prophet	7.13	**0.36**	32.87
	Ludwig	11.34	6.21	28.30
	DeepAR	10.29	286.90	8.13
	TFT	10.70	708.42	2.17
	FEDOT	***5.60**	208.13	0.39
	AutoTS	8.95	13.91	5.22
	Sktime	17.19	1.33	3.99
junction3	Pmdarima	3.82	121.28	**0.18**
	Prophet	2.56	**0.63**	39.20
	Ludwig	2.99	6.98	27.67
	DeepAR	†**1.82**	279.04	7.65
	TFT	2.58	703.93	2.17
	FEDOT	2.46	208.38	0.43
	AutoTS	2.15	13.70	5.66
	Sktime	4.26	0.59	1.74

$\star$ – Statistically significant under a paired comparison with all other methods.
† – Statistically significant under a paired comparison with Pmdarima, Prophet, Ludwig, DeepAR, TFT, FEDOT and Sktime.

time series (madrid, junction1 and junction2). For demonstration purposes, some examples of FEDOT forecasts are shown in Figure 2. While DeepAR also obtains three best results (AEP, COMED and junction3), in terms of the median NMAE, this AutoTSF method is ranked at third place. The other best forecasting results were obtained by: Ludwig – porto series; Pmdarima – lisbon series; and AutoTS – PJME series. On the other extreme, Sktime achieved the worst predictive performances in 7 out of 9 time series. As for the computational cost, Prophet is the lighter model in terms of training time, while Pmdarima produces TSF models that result in very fast inference times (around 0.34 ms). As for FEDOT, it requires a reasonable computational effort, around 3 minutes for model selection and training with thousands of observations and around 1.22 ms to generate a prediction, which is affordable for the analyzed hourly and daily time scales.

Table 6: Overall comparison results (median NMAE values and average training and prediction times; best values in **bold**).

ML Model	**NMAE** (in %)	**Train Time (s)**	**Prediction Time (ms)**
Pmdarima	6.54	106.27	**0.34**
Prophet	5.31	**0.39**	88.48
Ludwig	7.99	6.26	61.80
DeepAR	5.91	174.14	7.98
TFT	5.99	430.08	3.87
FEDOT	**4.58**	189.18	1.22
AutoTS	6.74	11.13	7.07
Sktime	8.99	1.19	3.09

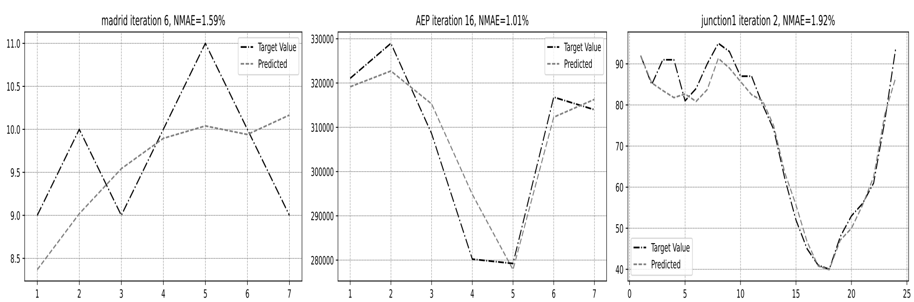

Fig. 2: Examples of multi-step ahead predictions using the FEDOT tool.

4 Conclusions

This paper compares eight recent AutoTSF tools (Pmdarima, Prophet, Ludwig, DeepAR, TFT, FEDOT, AutoTs and Sktime) using nine freely available time series that can be associated with smart city contexts. Using a realistic rolling window scheme, the AutoTSF tools were compared in terms of their predictive performances and computational effort (training and prediction times). Overall, the interesting results were obtained by the FEDOT AutoTSF tool. FEDOT obtained a low average forecasting error (around 4.58%), while requiring a reasonable computational effort, around 3 minutes to generate a new TSF model and 1.22 ms to produce a single prediction. In terms of future work, we intend to enlarge the comparison study by considering more time series (e.g., public car parking occupation, water consumption levels per district). Furthermore, we also plan to explore more AutoTSF Python modules (e.g., `hcrystalball`, `pyaf`).

Acknowledgments

This work has been supported by FCT - Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020 and the project "Integrated and Innovative Solutions for the well-being of people in complex urban centers" within the Project Scope NORTE-01-0145-FEDER-000086.

References

1. Alghamdi, T., Elgazzar, K., Bayoumi, M., Sharaf, T., Shah, S.: Forecasting traffic congestion using ARIMA modeling. In: 15th International Wireless Communications & Mobile Computing Conference, IWCMC 2019, Tangier, Morocco, June 24-28, 2019. pp. 1227–1232. IEEE (2019). https://doi.org/10.1109/IWCMC.2019.8766698
2. Box, G.E.: Gm jenkins time series analysis: Forecasting and control. San Francisco, Holdan-Day (1970)
3. Camero, A., Toutouh, J., Stolfi, D.H., Alba, E.: Evolutionary deep learning for car park occupancy prediction in smart cities. In: Battiti, R., Brunato, M., Kotsireas, I.S., Pardalos, P.M. (eds.) Learning and Intelligent Optimization - 12th International Conference, LION 12, Kalamata, Greece, June 10-15, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11353, pp. 386–401. Springer (2018). https://doi.org/10.1007/978-3-030-05348-2_32
4. Cortez, P., Matos, L.M., Pereira, P.J., Santos, N., Duque, D.: Forecasting store foot traffic using facial recognition, time series and support vector machines. In: Graña, M., López-Guede, J.M., Etxaniz, O., Herrero, Á., Quintián, H., Corchado, E. (eds.) International Joint Conference SOCO'16-CISIS'16-ICEUTE'16 - San Sebastián, Spain, October 19th-21st, 2016, Proceedings. Advances in Intelligent Systems and Computing, vol. 527, pp. 267–276 (2016). https://doi.org/10.1007/978-3-319-47364-2_26
5. Cortez, P., Rio, M., Rocha, M., Sousa, P.: Multi-scale internet traffic forecasting using neural networks and time series methods. Expert Systems **29**(2), 143–155 (2012). https://doi.org/10.1111/j.1468-0394.2010.00568.x, `https://doi.org/10.1111/j.1468-0394.2010.00568.x`
6. Cortez, P., Rio, M., Rocha, M., Sousa, P.: Multi-scale internet traffic forecasting using neural networks and time series methods. Expert Syst. J. Knowl. Eng. **29**(2), 143–155 (2012), `https://doi.org/10.1111/j.1468-0394.2010.00568.x`
7. Elattar, E.E., Sabiha, N.A., Alsharef, M., Metwaly, M.K., Abd-Elhady, A.M., Taha, I.B.M.: Short term electric load forecasting using hybrid algorithm for smart cities. Appl. Intell. **50**(10), 3379–3399 (2020). https://doi.org/10.1007/s10489-020-01728-x

8. Ferreira, L., Pilastri, A.L., Martins, C.M., Pires, P.M., Cortez, P.: A comparison of automl tools for machine learning, deep learning and xgboost. In: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021. pp. 1–8. IEEE (2021). https://doi.org/10.1109/IJCNN52387.2021.9534091
9. Hollander, M., Wolfe, D.A., Chicken, E.: Nonparametric statistical methods. John Wiley & Sons (2013)
10. Lim, B., Ark, S.O., Loeff, N., Pfister, T.: Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting **37**(4), 1748–1764 (2021). https://doi.org/https://doi.org/10.1016/j.ijforecast.2021.03.012
11. Lning, M., Bagnall, T., Kirly, F., Middlehurst, M., Ganesh, S., Oastler, G., Lines, J., ViktorKaz, Walter, M., RNKuhns, Mentel, L., Owoseni, T., Rockenschaub, P., jesellier, Tsaprounis, L., Bulatova, G., chrisholder, Lovkush, Take, K., Meyer, S.M., AidenRushbrooke, Schfer, P., oleskiewicz, danbartl, eenticott shell, Xu, Y.X., Ansari, A., Sakshi, A., Arelo, Hongyi: alan-turing-institute/sktime: v0.11.0 (mar 2022). https://doi.org/10.5281/zenodo.6386934, `https://doi.org/10.5281/zenodo.6386934`, [Online; accessed 2022-04-06]
12. Molino, P., Dudin, Y., Miryala, S.S.: Ludwig: a type-based declarative deep learning toolbox. CoRR **abs/1909.07930** (2019)
13. Murat, M., Malinowska, I., Gos, M., Krzyszczak, J.: Forecasting daily meteorological time series using arima and regression models. International Agrophysics **32**(2), 253–264 (2018). https://doi.org/10.1515/intag-2017-0007
14. Nagy, A.M., Simon, V.: Survey on traffic prediction in smart cities. Pervasive Mob. Comput. **50**, 148–163 (2018). https://doi.org/10.1016/j.pmcj.2018.07.004
15. Nikitin, N.O., Vychuzhanin, P., Sarafanov, M., Polonskaia, I.S., Revin, I., Barabanova, I.V., Maximov, G., Kalyuzhnaya, A.V., Boukhanovsky, A.: Automated evolutionary approach for the design of composite machine learning pipelines. Future Gener. Comput. Syst. **127**, 109–125 (2022). https://doi.org/10.1016/j.future.2021.08.022
16. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**(3), 1181–1191 (2020). https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.07.001
17. Siebert, J., Groß, J., Schroth, C.: A systematic review of python packages for time series analysis. CoRR **abs/2104.07406** (2021), `https://arxiv.org/abs/2104.07406`
18. Smith, T.G., et al.: pmdarima: Arima estimators for Python (2017–), `http://www.alkaline-ml.com/pmdarima`, [Online; accessed 2022-04-06]
19. Taylor, S.J., Letham, B.: Forecasting at scale. PeerJ Prepr. **5**, e3190 (2017). https://doi.org/10.7287/peerj.preprints.3190v1
20. Vijayalakshmi, B., Ramar, K., Jhanjhi, N.Z., Verma, S., Kaliappan, M., Vijayalakshmi, K., Vimal, S., Kavita, Ghosh, U.: An attention-based deep learning model for traffic flow prediction using spatiotemporal features towards sustainable smart city. Int. J. Commun. Syst. **34**(3) (2021). https://doi.org/10.1002/dac.4609
21. Wang, C., Chen, X., Wu, C., Wang, H.: Autots: Automatic time series forecasting model design based on two-stage pruning. CoRR **abs/2203.14169** (2022). https://doi.org/10.48550/arXiv.2203.14169