Seoul Bike Demand Analysis

William K Davis III Max Kutschinski Pei-Yin Yang

2022-07-11

Introduction

While the concept of renting transportation has been around for decades in the form of rental car companies, such as those often found at airports, it has also been adapted to the modern "sharing economy" in the form of bikeshare programs. These programs, often offered by local governments, allow users to rent bicycles for individual (point-to-point) trips, such as commuting to and from work. Cycling can be a faster mode of transportation than walking and might even be faster than driving or taking a taxi in the most congested of cities. Finally, bicycles offer a lower-pollution alternative to driving, which can be appealing to cities struggling to contain emissions.

A key challenge faced by administrators of bikesharing programs is the efficient allocation of available bicycles. Bikes must be available in the places that people need them and at the time they are needed in order for the program to be effective. In order to efficiently allocate bikes, administrators can periodically transport bikes from areas of low demand to areas of high demand (Schuijbroek et al. [2017]). Demand in a bikeshare system is a function of both time and location. In this paper we focus on the time component of the demand for bicycles at each hour of the day, ignoring the spatial component of demand. We will use the Seoul Bike Sharing Demand data from UCI (UCI [2020]). Our focus will be limited to predicting demand at each hour of the day, ignoring inferential aspects of the analysis. While much of the recent research using this dataset has focused on machine learning methodologies, we will take a multifaceted approach that incorporates advances in time series modeling in addition to the popular machine learning methodologies.

This paper begins with a review of the relevant literature, including studies of the Seoul data specifically. Next, we present the results of the exploratory data analysis, including a description of the dataset and relevant profiles of the features. The fourth section describes the modeling techniques to be applied to the data. The fifth section describes the evaluation techniques to be used for measuring model performance and selecting the best model. The penultimate section presents the results of the analysis and a discussion in the context of the problem to be solved. Finally, the conclusion provides suggestions for action and application of the results, while also highlighting potential areas of future research.

Literature

The popularity and accessibility of the Seoul bike dataset has resulted in its use for numerous studies. A majority of these studies have focused on the use of various machine learning algorithms. Sathishkumar and Yongyun found that a CUBIST model, which combines tree- and regression-based methods into a series of rules, performed best on the Seoul data when measured by R^2 and RMSE on the testing dataset (E and Cho [2020]). Gao and Chen found that another tree-based method, random forest (RF), performed best on a similar bikeshare dataset when measured by R^2 and RMSE (Gao and Chen [2022]). Both studies further showed that weather-related variables, such as temperatures and precipitation, where among the most important for predicting demand. Gao and Chen's results highlight the importance of selecting a relevant evaluation metric and explanatory variables. When socioeconomic variables were included in the model, the RF outperformed the support vector machine (SVM) when measured by both RMSE and MAE. However, when the socioeconomic variables were excluded from the model, the RF outperformed the SVM when measured by RMSE, but the SVM performed better when measured by MAE This indicates that without the socioeconomic variables included, the SVM was prone to a few errors that were quite large in magnitude, while the RF was more prone to smaller but more frequent errors. This reinforces the importance of using multiple metrics when evaluating predictions.

Considerably fewer researchers have made use of traditional time series methodologies when predicting demand of a similar nature to the bikeshare data. Both (E and Cho [2020]) and (Gao and Chen [2022]) make use of temporal variables such as hour, day of the week, and holidays. In each case they were found to be of moderate or high importance. (Gao and Chen [2022]) applied linear regression using temporal variables such as a weekend indicator as a predictor, but this model greatly underperformed the machine learning methods. Further, there is no discussion of any attention paid to stationarity and autoregression, which are common in time series data but may also result in violations of the standard assumptions of linear regression.

Our analysis looks to build on this work by using newer machine learning methods such as long short-term memory and recurrent neural networks, as well as modern time series techniques that leverage the underlying structure of the data.

Exploratory Analysis

The dataset consists of 8,760 hourly observations of 12 variables from 2017-12-01 00:00:00 to 2018-11-30 23:00:00. There are 295 observations where *BikeCount*=0 due to the bikeshare system not functioning. There are no other periods where *BikeCount*=0. Table **??** contains information on the variables in the dataset, including the

Table 1: Variable definitions

Variable	name	Type	Definition
Hour	Hour	datetime	year-month-day hour:minute:second
Rented Bike count	BikeCount	numeric	Count of bikes rented at each hour
Temperature	Temperature	numeric	Temperature in Celsius
Humidity	Humidity	numeric	% humidity
Windspeed	WindSpeed	numeric	meters/second
Visibility	Visibility	numeric	in 10m
Dew point temperature	Dewpoint	numeric	Celsius
Solar radiation	SolarRadtion	numeric	MJ/m2
Rainfall	Rainfall	numeric	mm
Snowfall	Snowfall	numeric	cm
Seasons	Seasons	categorical	Winter, Spring, Summer, Autumn
Holiday	Holiday	categorical	Holiday/No holiday
Functional Day	FunctionalDay	categorical	NoFunc(Non Functional Hours), Fun(Functional hours)
Workday	Workday	categorical	Workday/Not Workday. A workday is a weekday that is not a holiday.

Bike Count

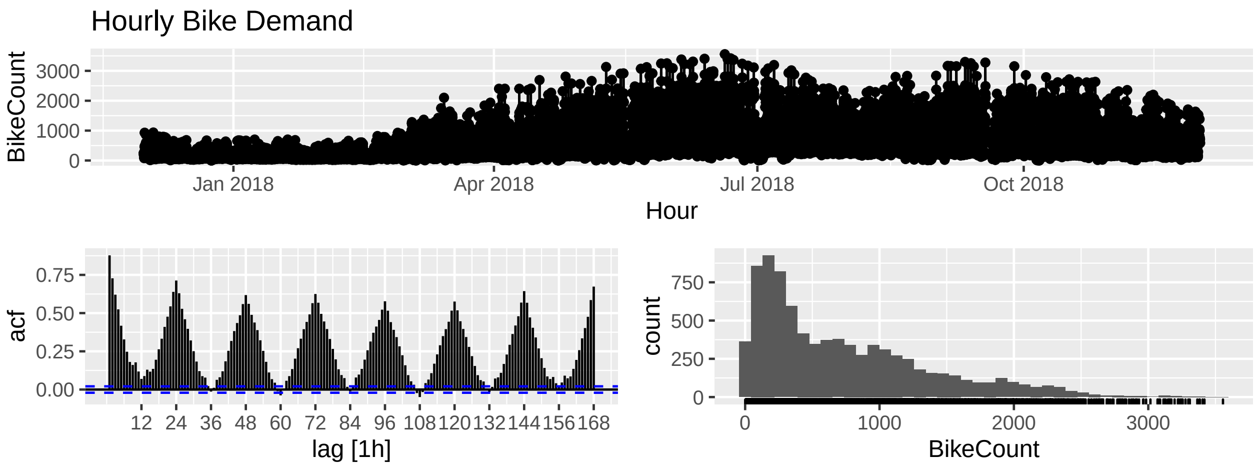

Figure 1: Hourly bike demand

The data shows increasing demand and variability during the summer months. The bike demand data are counts, meaning they are technically discrete. Based on the histogram and the count nature of the data, it appears that a poisson distribution would be most appropriate for the data. If we were to model the bike demand on a continuous scale, a log-normal distribution might be appropriate.

Figure 1 highlights the incredible variation in demand over time. The variance in demand appears to increase during the summer months and then decrease again in the autumn. This heteroskedasticity violates the constant variance assumption of ordinary least squares regression and will have to be corrected if regression-based methods are to be used. We can also see that the mean of the series appears to increase during the summer months before decreasing again in the autumn.

A KPSS test yields $p = 0.01$, meaning there is strong evidence in favor of the presence of a unit root and the data is likely non-stationary (Kwiatkowski et al. [1992]). A Ljung-Box test was conducted up to 24 lags which resulted in a test statistics of 39743 with $p = 0$, indicating strong evidence that there is serial correlation in the hourly bike count (LJUNG and BOX [1978]). The acf plot in Figure 1 shows that autocorrelation is significant at most lags out to 168 hours, which represents the same hour of the same day in the previous week. The strong serial correlation makes this dataset a good candidate for time series techniques.

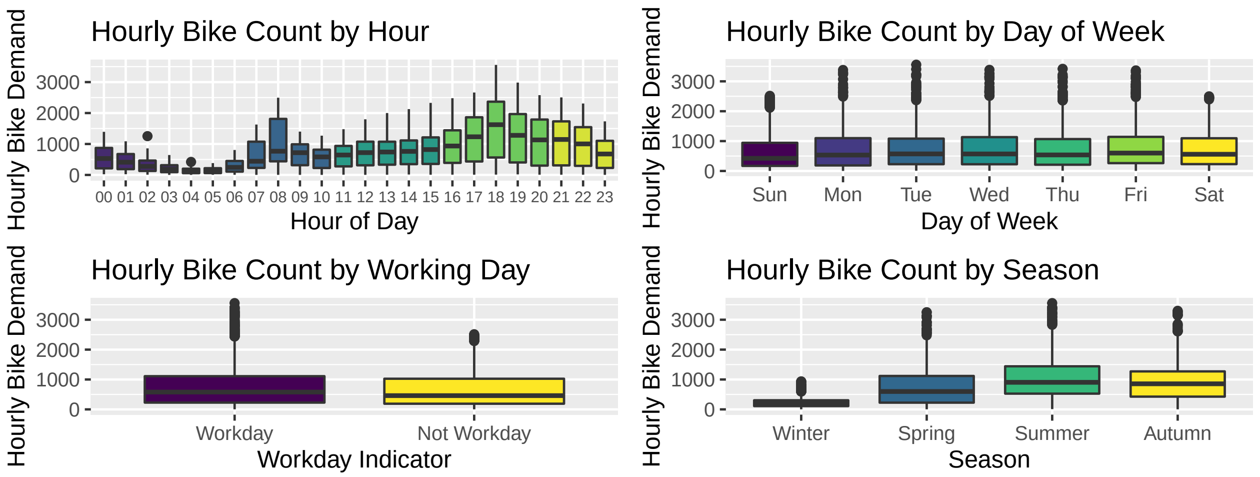

Figure 2: Hourly bike demand by time period

Figure 2 highlights various seasonal and temporal patterns in the data. There is certainly an effect based on the hour of day, with demand increasing sharply in the morning, presumably during the morning commute, before decreasing into the lunchtime hour. From there, demand rises steadily through the evening commute before peaking around dinner time and then falling through the nighttime hours. Variability also appears greater during the evening hours, which is consistent with the idea heteroscedasticity; the variance increases with the level of the series. Demand appears slightly higher during workdays (where workdays are weekdays that are not holidays) and weekdays, though the difference does not appear particularly large. There are a number of large positive outliers during the weekdays. Finally, demand appears largest during the summer months and smallest during the winter, as biking would be a less desirable option in the cold.

Figure **??** provides insight into the seasonality of the bike demand. Each panel is a 4-week sample of hourly bike demand, with non-workdays (weekends and holidays) highlighted in red. While these days appear to follow a consistent hourly pattern much like the workdays, we can see that their is a difference between the seasonality of workdays and the seasonality of non-workdays. Most notably, the troughs in demand appear consistent between the types of days while the peaks are consistently higher for workdays as compared to non-workdays. This changing seasonality based on type of day will need to be captured in our model.

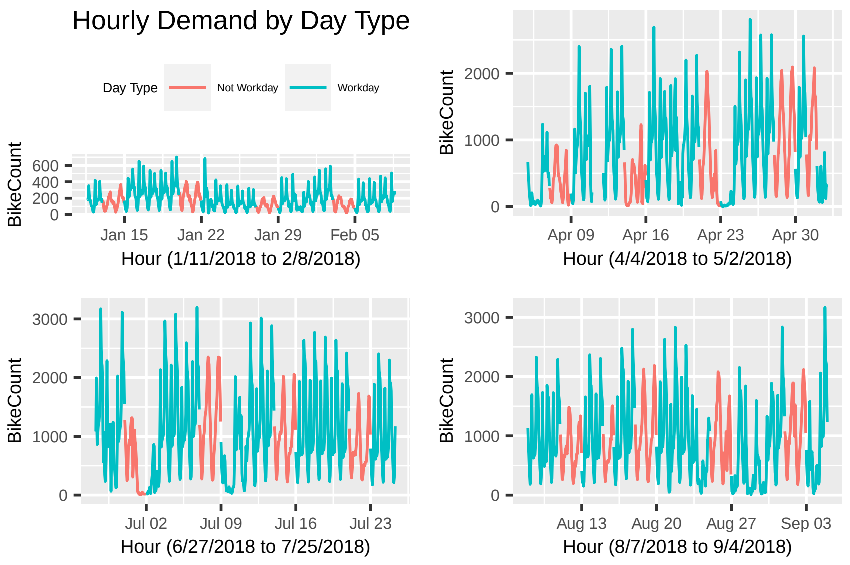

Figure 3: Hourly bike demand by type of day, highlighting differing seasonality between workdays and non-workdays

Continuing with the impact of seasonality and temperature on demand, we will next explore covariates included with the dataset.

Covariates

This dataset includes a number of covariates to aid in modeling bike demand. All of these covariates are listed in Table 1, along with their definitions. The covariates fall into one of two broad categories: weather and social. Weather covariates include variables such as temperature, humidity, precipitation, and others. Social variables capture the impact of calendar-based human behavior, such as holidays and weekends.

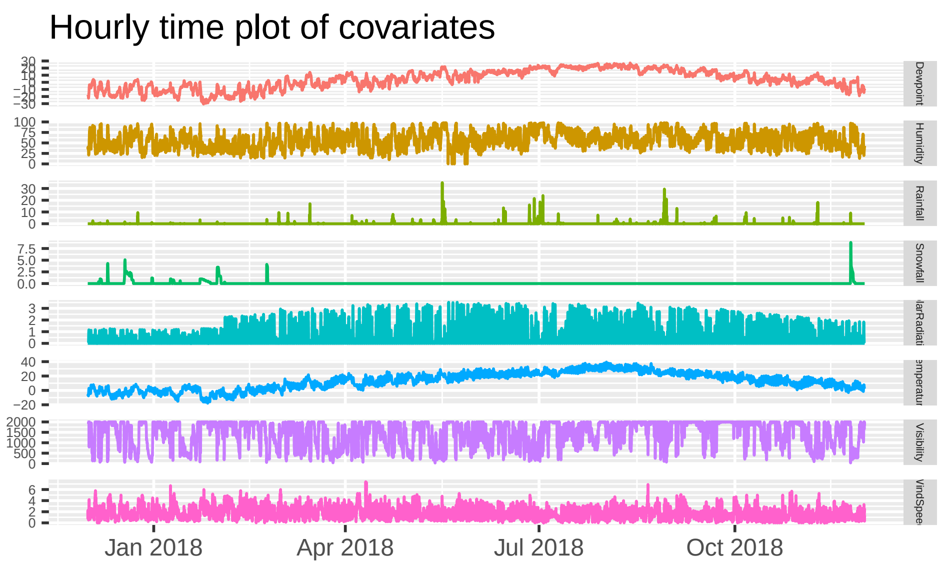

Figure 4: Hourly time plots of covariates

Time plots of the covariates (Figure 1) show a dynamic set of variables. Most of the covariates appear to be typical time series data with varying degrees of trend, seasonality, cyclicality, autocorrelation, and heteroskedasticity. Dewpoint and Solar Radiation follow a predictable pattern that mirrors temperature throughout the year, with an upward trend peaking in the summer months and downward trend that hits a trough in the winter months. Humidity, visibility, and Wind Speed appear to have less of a trend throughout the year. Precipitation (rainfall and snowfall) appears to have a much more random pattern throughout the year, with a large number of periods having no precipitation. Depending on the predictive performance of the raw continuous precipitation features, it may prove more performent to convert them to binary variables that simply indicate if precipitation occurred during that period.

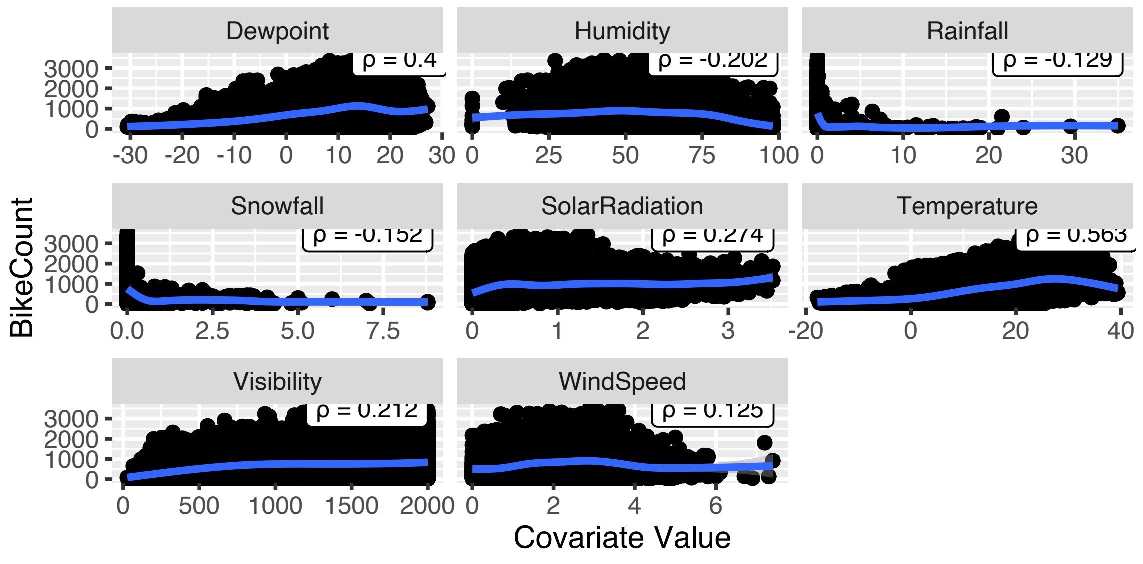

Figure 5: Scatterplots and smoothed GAM fits of Bike Count vs. each covariate

Figure 5 highlights a number of important relationships in the data. Bike count appears moderately linearly correlated with both temperature ($\rho = 0.563$) and dewpoint ($\rho = 0.400$), though a higher-order interaction might better represent this relationship. However, there is significant multicollinearity between temperature and dewpoint ($\rho = 0.913$), so additional analysis will be required to isolate the effect of each variable. Finally, we can see that a number of the covariates, such as wind speed, visibility, solar radiation, rainfall, and snowfall all appear to follow non-normal distributions. As such, their relationship to bike count might be best represented with a higher-order polynomial or other non-linear relationship.

Modeling

The modeling methods we have chosen for predicting bike demand fall into roughly two categories: parametric time series modeling and non-parametric machine learning. The parametric time series models include traditional time series methods like ARIMA and modern methods including the regression-based Prophet and dynamic linear model FASSTER. The non-parametric machine learning methods include random forests, boosted trees, and Long Short-Term Memory neural networks.

The stated goal of this analysis is to predict bike demand, not to conduct statistical inference (on the factors that impact bike demand). Therefore, the non-parametric methods can be used freely despite their typical lack of inferential ability. Further, we do not have to constrain ourselves to the typical assumptions of parametric methods that are required for inference, namely independent observations and normally distributed residuals with mean zero and constant variance. The assumption of independent observations is obviously violated in time series data and most often appears in the form of autocorrelated errors. Therefore, the ability to disregard the assumptions required for inference in favor of a focus on prediction will greatly expand the number of viable methods for modeling the data.

Time Series Modeling

Figure 1 highlights the heteroskedasticity in the bike demand. This can be corrected using a simple Box-Cox transformation (Box and Cox [1964]). If we let x_t represent the value of raw Bike Count data and y_t represent the transformed data, we have

$$y_t = \frac{x_t^\lambda - 1}{\lambda} \tag{1}$$

where $\lambda = 0.1478452$ was selected using Guerrero's method (Guerrero [1993]).

SARIMAX

Traditional linear regression models can be adapted to handle the autocovariance structure of time series by assuming that the errors follow a seasonal ARIMA (SARIMA) process instead of the traditional $\epsilon \sim$ iid $N(0, \sigma^2)$. This results in the modified regression equation

$$y_t = \beta_0 + \sum_{j=1}^{k} \beta_j z_{jt} + x_t \tag{2}$$

where:

- y_t is the response variable at time t
- β_0 is the traditional intercept
- $z_{1t}, \ldots, z_{kt}$ are the k exogenous regressors observed at time t.
- $\beta_1, \ldots, \beta_k$ are the regression coefficients.
- x_t are the regression errors, which are assumed to follow an SARIMA process as in (3).

(2) is often referred to as SARIMAX for Seasonal ARIMA with eXogenous regressors.

$$\Phi_P(B^S)\phi(B)\nabla_S^D\nabla^d x_t = \Theta_Q(B^S)\theta(B)w_t \tag{3}$$

where:

- $\Phi_P(B^S)$ are the P seasonal autoregressive components
- $\phi(B)$ are the p autoregressive components
- ∇_S^D are the D seasonal differences
- ∇^d are the d differences
- $\Theta_Q(B^S)$ are the Q seasonal moving average components
- $\theta(B)$ are the q moving average components
- $w_t \sim$ iid $N(0, \sigma_w^2)$ is the traditional Gaussian white noise

(Shumway and Stoffer [2019]). SARIMA models of this form are often written $ARIMA(p, d, q) \times (P, D, Q)_S$. A KPSS test indicates that the data is non-stationary and requires $D = 1$ seasonal difference (Kwiatkowski et al. [1992]). Autocorrelation and partial autocorrelation plots can be used to determine the other ARIMA parameters (AR and MA).

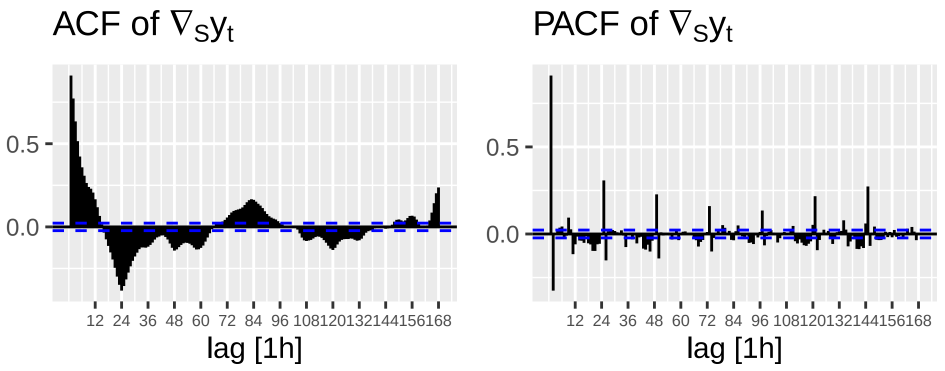

Figure 6: Correlograms for $\nabla_S y_t$

Figure 6 indicates a seasonal autoregressive model with $P = 6$ and $p \geq 12$. This is a large number of seasonal autoregressive terms and will almost certainly result in characteristic roots inside the unit circle, causing the model to be unstable. A quick search of the parameter space using the `R` function `fable::ARIMA` (O'Hara-Wild et al. [2021]) to automatically select values for p, d, q, Q results in no stable $ARIMA(p, d, q) \times (6, 1, Q)_{24}$ models being found.

When the data exhibit higher frequency or multiple types of seasonality (such as daily and weekly, in our case), an alternative solution for modeling seasonality can be to use fourier terms (Hyndman and Athanasopoulos [2021]). Introducing the fourier terms will result in a model of the form

$$y_t = \beta_0 + \sum_{j=1}^{k} \beta_j z_{jt} + s_d(t, m) + s_w(t, n) + x_t \tag{4}$$

$$s_d(t, m) = \sum_{i=1}^{m} \left[\alpha_i \sin\left(\frac{2\pi i t}{24}\right) + \beta_i \cos\left(\frac{2\pi i t}{24}\right) \right] \tag{5}$$

$$s_w(t, n) = \sum_{l=1}^{n} \left[\gamma_l \sin\left(\frac{2\pi l t}{168}\right) + \delta_l \cos\left(\frac{2\pi l t}{168}\right) \right] \tag{6}$$

where:

- $\beta_0 + \sum_{j=1}^{k} \beta_j z_{jt}$ are the intercept, independent variables and their coefficients, as defined in (2)
- $s_d(t, m)$ are the m daily seasonality fourier terms
- $s_w(t, n)$ are the n weekly seasonality fourier terms

- x_t are the model errors, which are assumed to follow a $ARIMA(p, d, q)$ process (note that in contrast to (2) this is a non-seasonal ARIMA model; the seasonality is omitted from the ARIMA process because it is captured by the fourier terms).

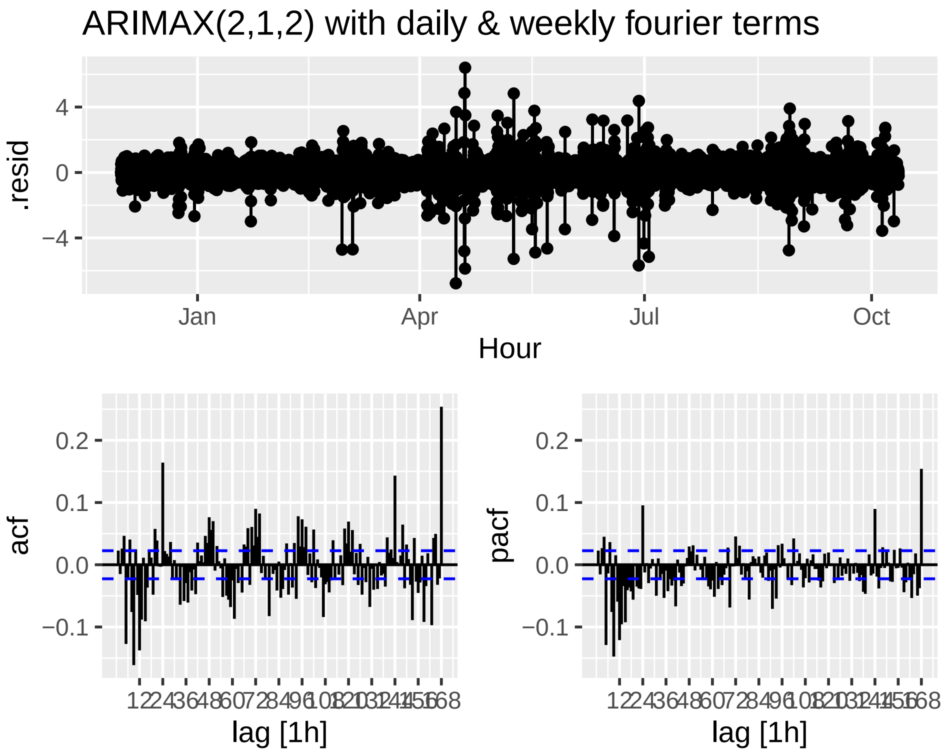

Figure 7: Residuals for an ARIMAX(2,1,2) model with daily and weekly fourier terms

The fourier model resulted in the selection of an $ARIMA(2, 1, 2)$ process for the regression residuals. The residual ACF and PACF plots from the fourier model (Figure **??**) still show significant autocorrelation. Autocorrelation in the residuals violates the Gauss-Markov assumptions, meaning the model should not be used for inference. The goal of this analysis is to generate accurate (point) forecasts, which are still valid even when the errors exhibit autocorrelation.

Finally, we will fit a linear regression with SARIMA errors to the data where all parameters p, d, q, P, D, Q are chosen automatically using `fable::ARIMA`, which relies on the algorithm developed by (Hyndman and Khandakar [2008]). This resulted in the selection of an $SARIMA(1, 0, 1) \times (5, 1, 0)_{24}$ for x_t.

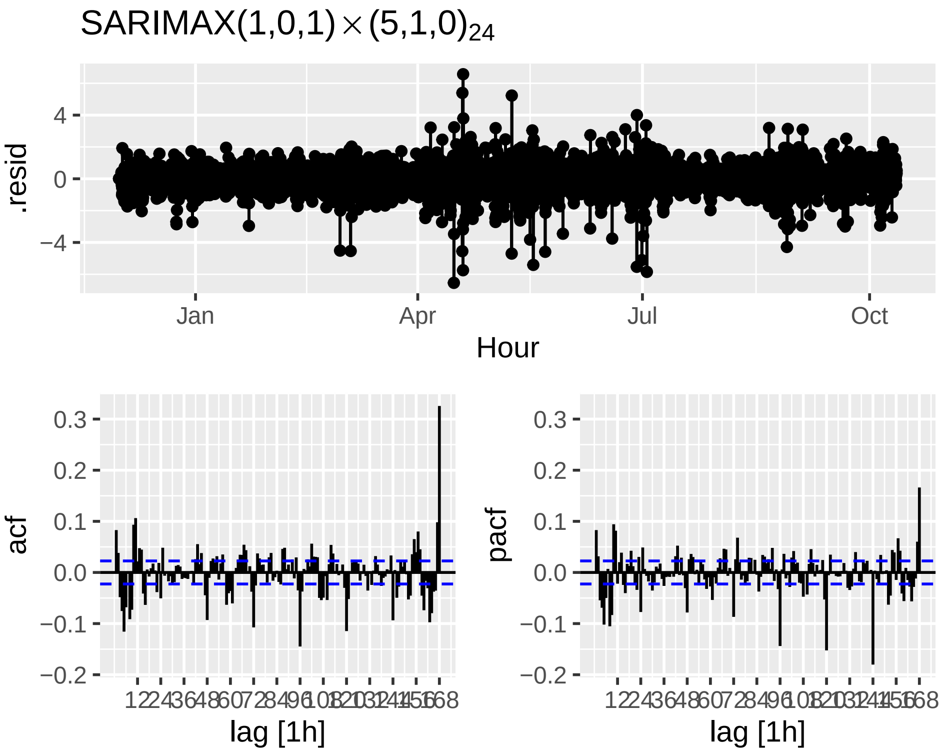

Figure 8: Residuals for an $SARIMAX(1,0,1) \times (5,1,0)_{24}$

Figure **??** shows that, while the autocorrelation (and partial autocorrelation) for a $SARIMA(1,0,1)\times(5,1,0)_{24}$ model has been reduced over the $ARIMAX(2,1,2)$ with fourier terms, there is still repeating statistically significant autocorrelation. It is possible that the seasonal patterns in the data are too complex or too high frequency to be captured by an ARIMA, ARIMAX, or SARIMAX model. Therefore, we will explore more modern time series techniques.

Prophet

Prophet uses a generalized additive model (GAM) to capture the different features of the time series (Taylor and Letham [2018]). Our model will take the form

$$y(t) = g(t) + s_d(t,m) + s_w(t,n) + \sum_{j=1}^{k} \beta_j z_{jt} + \epsilon_t \tag{7}$$

where:

- $g(t)$ is a piecewise constant function to represent the trend in the series
- $s_d(t,m)$ as in (5)
- $s_w(t,n)$ as in (6)
- $\sum_{j=1}^{k} \beta_j z_{jt}$ is as in 2
- ϵ_t is the model error

One major benefit of (7) is that it is much faster to fit than a model with ARIMA terms. The downside is that it does not explicitly capture (non-seasonal) $AR(p)$ and $MA(q)$ terms. We will use `fable.prophet::prophet` to train the model in `R` (O'Hara-Wild [2020]). The `R` function sets a number of desirable default parameter values as recommended by (Taylor and Letham [2018]). In order to properly tune the model to our dataset we will use TSCV (see Model Selection and Test Error) to select m and n. The TSCV will consist of 63 folds/time-slices. We will search over the parameter space $m \in \{1, 2, 5, 10, 20, 50\}$ and $n \in \{3, 5, 10, 20, 50\}$, giving us $6 \times 5 = 30$ possible parameter combinations.

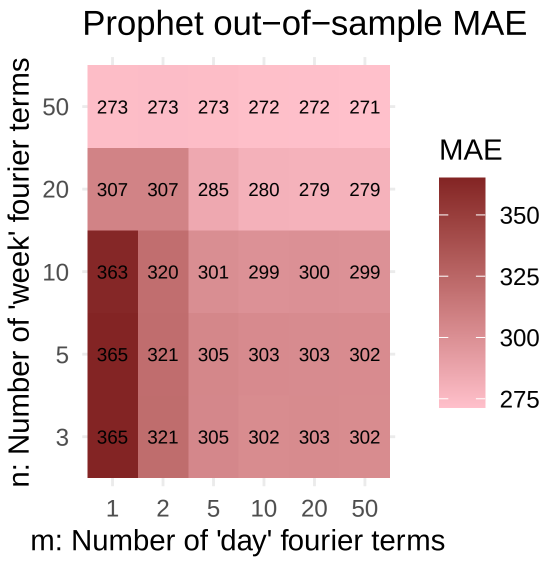

Figure 9: Cross-validated MAE for Prophet models with various numbers of (daily and weekly) fourier terms

Figure 9 identifies $m = 50$ and $n = 50$ as the optimal number of fourier terms (they minimize MAE). We fit this model to the entire training dataset and analyze the residuals.

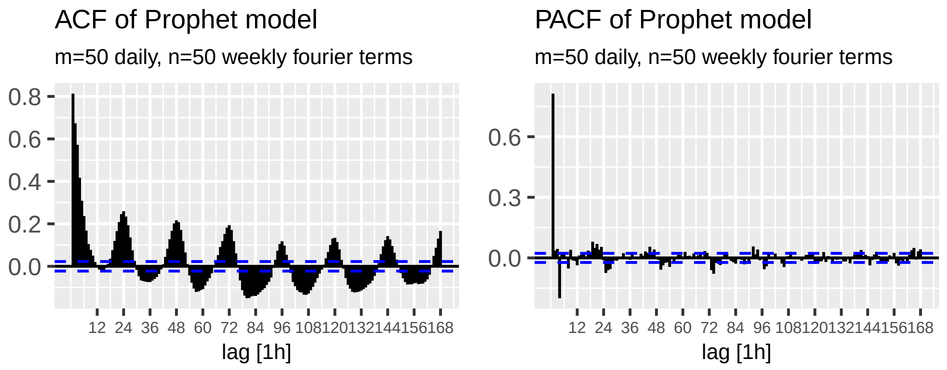

Figure 10: Residuals for a Prophet model with $m = 50$ (daily) and $n = 50$ (weekly) fourier terms

As can be seen from the PACF of the residuals in Figure 10, there is still autocorrelation not captured by the model. However, given that our goal is forecasting/prediction rather than inference, it should not pose a problem for the analysis.

fasster

One interesting feature of the residual PACFs from both the SARIMAX model (Figure **??**) and the Prophet model (Figure 10) is that the seasonal "spikes" are parabolic in shape. That is, the PACF starts high at lag 24, then decreases until reaching a min around lag 72 or lag 96 before rising again to another peak at lag 168 (which is the one week lag). This shape might be why the AR and fourier terms are not capturing all of the autocorrelation in the data: the strength of the correlation appears to be a non-linear function of time.

One reason for this shape in the PACF plot could be the presence of switching seasonality. More specifically, it's possible that the seasonality (not just the series itself) is different on working days and non-working days. A hand-wavy analysis would say that, given there are seven days in a week, any single day is, on "average", $7/2 = 3.5$ days away from any other day. For weekdays specifically, they are 3.5 days away from a weekend (again, on average).

Therefore, the trough in the PACF curve occurring between 3 and 4 days might be due to that lag most frequently correlating weekdays with weekend days, resulting in lower correlation than when weekdays are compared with other weekdays, which would occurr most frequently at lags < 3 and > 4. This is certainly not a technical explanation of the pattern, but it is plausible. The previously employed techniques (SARIMAX and Prophet) don't allow for changing seasonality. The fasster methodology allows for the seasonal (and trend) component(s) of the model to be switched based on the "state" of a given observation, which is defined via a discrete variable passed to the model (O'Hara-Wild and Hyndman [2022]). In our case, this variable would be *Workday*, which identifies the observation as a workday or non-workday (weeeknd or holiday). Different seasonality coefficients can then be fit based on whether a bike count is observed on a workday or non-workday. This is an improvement over the previous models, which simply include the *Workday* variable as a regression coefficient, reducing its effect to a shift in the level of the series.

Machine Learning

Tree Methods

Random Forests Random forests is an ensemble method for regression and classification tasks that is built on decision trees. It extends on the idea of bootstrap aggregation, or bagging, which is used to reduce the variance of a statistical learning method via averaging. In the context of regression trees, this is done by constructing B unpruned trees from B bootstrap samples and averaging the predictions as displayed in Eq. (8).

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f^{*b}}(x) \tag{8}$$

One notable drawback of bagging is that the individual trees can be very correlated depending on how strong the predictors are. Random forests addresses this issue by randomly selecting a subset of the features at each split and thus de-correlating the trees.

Random Forests (RF) is implemented using the ranger package for increased computational speed. Note that RF is an algorithm that is known to provide good results in the default settings (Fernández-Delgado et al. [2014]). The arguably most influential hyperparameter is *mtry*, the number of randomly drawn features that are available at each split. In the regression case, p/3 is the default setting. When executing ranger via caret it automatically performs a grid search of *mtry* its entire parameter space. By default, the algorithm evaluates 3 points in the parameter space (smallest and largest possible *mtry*, as well as their mean) with 25 bootstrap iterations as an evaluation strategy and chooses the value with the lowest MSE.

The hyperparameters of our model are evaluated using TSCV, yielding an optimal *mtry* value of 53. Such a high value is usually indicative of a high number of relevant predictors (Probst et al. [2019]).

XGBoost XGBoost is an advanced implementation of Gradient Boosting, which is an ensemble method for regression and classification tasks that combines multiple weak learners into a stronger learner. In the context of decision trees, a weak learner is defined as a tree with a small number of terminal nodes. The trees are grown sequentially and they are fit on the residuals of the current fit as opposed to the outcome Y. This has the effect of capturing signal that is not yet accounted for by the current set of trees. In addition, each weak learner is shrunken down by some shrinkage factor before it is used, making boosting a "slow" learning approach.

The first step is to initialize the model with a constant value γ, which can be obtained by solving the following optimization problem:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma) \tag{9}$$

After specifying the number of base learners M, the following steps are repeated for each base learner from $m = 1$ to $m = M$:

First, the pseudo-residuals are calculated for each ith training example.

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \tag{10}$$

Then a base learner $h_m(x)$ is fit to the pseudo-residuals using the modified training set $\{(x_i, r_{im})_{i=1}^n\}$.

Lastly, the model is updated as follows:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{11}$$

$$\text{where } \gamma_m = \text{argmin}_\gamma \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

XGBoost is a more regularized form of Gradient Boosting which uses L1 and L2 regularization to improve model generalization capabilities. It also allows for parallel processing, and has a built-in routine for handling missing values via its sparsity-aware split finding algorithm.

There are a variety of booster parameters in XGBoost that can be optimized via TSCV. Rather than tuning all parameters simultaneously, it often helps to make small changes incrementally (Banerjee). The general idea is to start with a high learning rate and a small number of base learners, then tune other parameters, and finally decrease the learning rate while proportionally increasing the number of trees. The other adjusted parameters are the maximum depth of a tree *max_depth*, the minimum required loss reduction *gamma*, the fraction of columns to be subsampled *colsample_bytree*, the minimum sum of weights of all observations required in a child *min_child_weight*, and the fraction of observations to be randomly sampled per tree *subsample*.

Long Short-Term Memory (LSTM) Recurrent Neural Network

LSTM is a variety of Recurrent Neural Network (RNN). The benefit of the Long Short-Term Memory (LSTM) network over other recurrent networks comes from "constant error back propagation" (Hochreiter and Schmidhuber [1997]), an improved method of back-propagating the error.

The Constant Error Carousel (CEC) is the magic of the LSTM in that it prevents vanishing gradients. It is denoted as follows:

$$c_{t+1} = c_t \times \text{forget gate} + \text{new input} \times \text{input gate}$$

In the case of regular RNNs during backpropagation, the derivative of an activation function, such as a logistic, will be less than one. Therefore over time, the repeated multiplication of that value against the weights will lead to a vanishing gradient.

In the case of an LSTM, we only multiply cell state $\times$ forget gate, which acts as both the weights and the activation function for the cell state. As long as forget gate $= 1$, the information from the previous cell state passes through unchanged. This is why LSTM can deal with more intricate problems than the RNN, by keeping a constant flow of error throughout the backpropagation from cell to cell.

Data processing It is a good idea for machine learning algorithms that fit a model that uses a weighted sum of input variables, such as linear regression, logistic regression, and artificial neural networks (deep learning) to normalize the data (Brownlee [2020]). Here, by using the `scikit-learn` object *MinMaxScaler*, the normalization scales each input variable separately to the range $[0, 1]$, which is the range for floating-point values where we have the most precision. A value is normalized as follows:

$$y = \frac{x - \min\{x\}}{\max\{x\} - \min\{x\}}$$

Model Training `keras`, a Python library developed by Google, is used for model building. Each input data was a list of lagged hours of bike rental count and the output data for that particular input was the bike demand for the next hour. One-hour, 12-hour, and 24-hour lagged timesteps are used for the input data.

The following hyperparameters must be tuned:

- **nodes(neurons)**: units accepting a vector of real-valued inputs and producing a single real-valued output.
- **batch size**: how many obs. are used at each step, converge faster with a larger value.
- **epoch**: an iteration over all training obs.
- **dropout**: regularization method to prevent over-fitting
- **learning rate**: define how quickly the network updates its parameters

Forecast performance is assessed using Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and the learning curve. A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

Lag 1 Hour Model **Model Architecture**:

1) LSTM with 50 neurons in the first visible layer
2) Dropout 50%
3) 1 neuron in the output layer for predicting Bike Demand.
4) The input shape will be 1 time step with 12 features.
5) I use the Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent.
6) The model will be fit for 80 training epochs with a batch size of 12.

Although in Figure **??** we can see both errors converge fairly fast, the proposed network is not a good fit since there is a gap between the training error and test error.

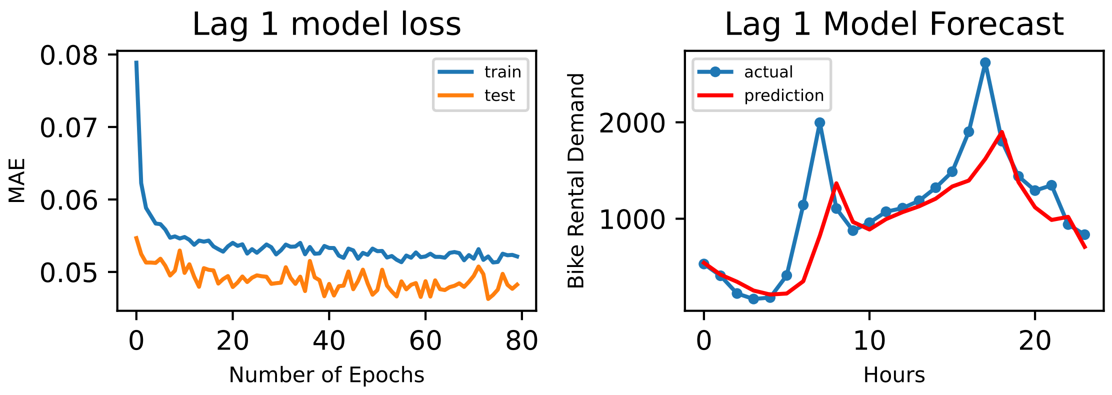

Figure 11: LSTM Lag 1 Model Loss and Forcasted Bike Rental Demand of one day

Lag 12 Hours Model **Model Architecture**:

1) LSTM with 50 neurons in the first visible layer
2) Dropout 50%
3) 1 neuron in the output layer for predicting Bike Demand.
4) The input shape will be 12 time step with 12 features.
5) I use the Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent.
6) The model will be fit for 60 training epochs with a batch size of 12.

The Loss vs Epoch curve is shown in Figure **??** in which the progress of mode while training is represented. Both training and testing loss decreases in a smooth fashion fairly quickly.

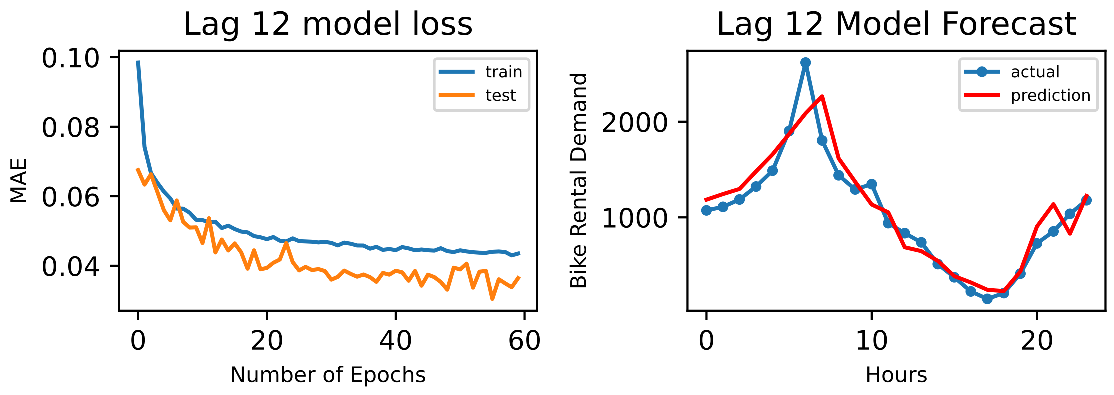

Figure 12: LSTM Lag 12 Model Loss and Forcasted Bike Rental Demand of one day

Lag 24 Hours Model **Model Architecture**:

1) LSTM with 50 neurons in the first visible layer
2) Dropout 50%
3) 1 neuron in the output layer for predicting Bike Demand.
4) The input shape will be 24 time step with 12 features.
5) I use the Mean Absolute Error (MAE) loss function and the efficient Adam version of stochastic gradient descent.
6) The model will be fit for 60 training epochs with a batch size of 12.

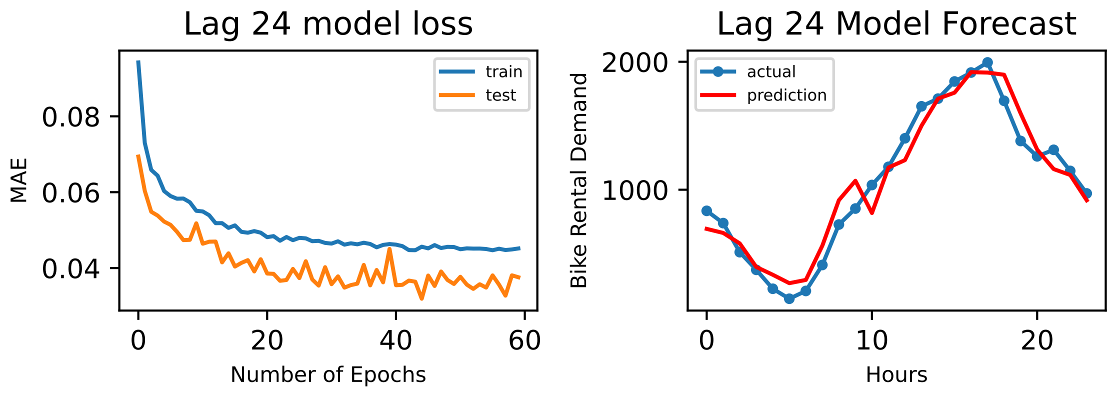

Figure 13: LSTM Lag 24 Model Loss and Forcasted Bike Rental Demand of one day

Evaluation

Error

Given our stated use for these models is forecasting (prediction), when discussing model evaluation we must first define the notion of forecast (prediction) error rate. There are a number of different metrics to use for forecast error, each with their own benefits and drawbacks (Hyndman and Koehler [2006]). We will use Mean

Absolute Error (MAE) to select the best model from among the list of candidate models. Unlike percent errors, which have the general form $100 \times \frac{\hat{e}}{y_t}$, MAE is defined when $y_t = 0$. Further, MAE is on the same scale as the original dataset (number of bikes per hour), which gives it nice properties of interpretability. We will also report Root Mean Squared Error (RMSE), as it is a common error metric. Using MAE will result in forecasts of the median while RMSE results in forecasts of the mean (Hyndman and Athanasopoulos [2021]). Because our data appear skewed, the median is a more appropriate measure of center. Therefore, MAE is the best error metric for selecting the best forecasting model and estimating test error.

For the purposes of discussing model error rate, let y_t be the observed bike count in period t, $\hat{y}_t$ be the forecast bike count in period t. Then,

$$\hat{e}_t = y_t - \hat{y}_t \tag{12}$$

$$\text{Mean Absolute Error} = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}|e_{ij}| \tag{13}$$

$$\text{Root Mean Squared Error} = \sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}e_{ij}^2} \tag{14}$$

Where $m = 24$ is the number of periods for which bike count is forecast (the horizon) and $n = 50$ is the number of iterations (folds) in the time series cross-validation.

Model Selection

Models will be selected and evaluated in the context of their use for forecasting rather than statistical inference. As such, we will use time series cross-validation (TSCV). TSCV consists of selecting a point or series of points from the dataset as test sets, then selecting all prior points as the training set (Hyndman and Athanasopoulos [2021]). Using the example in Figure F, the first iteration trains on the first 5 observations (blue) and generates forecasts on the next 3 observations (green). In the second iteration the model trains on the first $5 + 3 = 8$ observations and forecasts on the next 3 observations. This continues until the final iteration, where the model trains on all but the last 3 observations and then generates forecasts for the final 3 observations. In this example 5 is the initialization value (the number of training observations in the first iteration), 3 is the step size (the number of observations that are added to the training set each iteration), and 3 is also the horizon (the number of periods for which we generate a forecast). Each of the forecasts is then compared with actual (observed) values to evaluate the forecast.

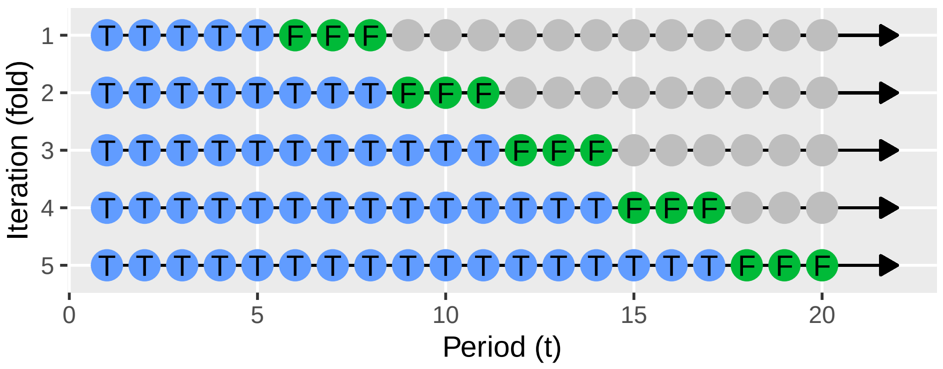

Figure 14: Visualization of time series cross-validation example

Our evaluation will use use the last 50 days of data as the test folds, with each test fold having 24 observations. The first iteration will train on the 7560 observations from 2017-12-01 00:00:00 to 2018-10-11 23:00:00 and

Table 2: Train and test set (fold) details for the time series cross-validation process used to calculate estimated test error (MAE and RMSE).

fold	Train Start	Train End	Train n	Test Start	Test End	Test n
1	2017-12-01 00:00	2018-10-11 23:00	7560	2018-10-12 00:00	2018-10-12 23:00	24
2	2017-12-01 00:00	2018-10-12 23:00	7584	2018-10-13 00:00	2018-10-13 23:00	24
3	2017-12-01 00:00	2018-10-13 23:00	7608	2018-10-14 00:00	2018-10-14 23:00	24
...	...	...	...	...	...	...
24	2017-12-01 00:00	2018-11-03 23:00	8112	2018-11-04 00:00	2018-11-04 23:00	24
25	2017-12-01 00:00	2018-11-04 23:00	8136	2018-11-05 00:00	2018-11-05 23:00	24
...	...	...	...	...	...	...
50	2017-12-01 00:00	2018-11-29 23:00	8736	2018-11-30 00:00	2018-11-30 23:00	24

Table 3: Test Error by Model Type

Model	MAE	RMSE
Baseline		
Baseline	190	330
Time Series Methods		
auto SARIMAX	208	319
auto ARIMAX w/ Fourier	212	314
Prophet	162	233
Tree Methods		
Random Forest	165	232
XGBoost	129	172
LSTM Models		
LSTM Lag 1	172	287
LSTM Lag 12	130	178
LSTM Lag 24	134	182

test (forecast) on the 24 observations from 2018-10-12 00:00:00 to 2018-10-12 23:00:00. The second iteration will train on the 7584 observations from 2017-12-01 00:00:00 to 2018-10-12 23:00:00 and test (forecast) on the 24 observations from 2018-10-13 00:00:00 to 2018-10-13 23:00:00. This continues until the final iteration where it trains on the 8736 observations from 2017-12-01 00:00:00 to 2018-11-29 23:00:00 and tests (forecasts) on the 24 observations from 2018-11-30 00:00:00 to 2018-11-30 23:00:00. This means our total sample size for calculating the error metrics for each model will be $24 \times 50 = 1200$. Table 1 depicts the time periods used and sample sizes for a sample of the folds.

Results

A common-sense, non-machine learning baseline serves as a sanity check and is often used to establish a baseline of comparison for more advanced machine learning models.

In this case, bike demand can be assumed to be season with a daily (24 hours) period. Given hourly data, a common-sense baseline is to predict the bike demand at time t to be equal to the bike demand at time $t - 24$, the same hour the previous day. In the time series paradigm this is known as a Seasonal NAIVE (SNAIVE) model.

$$\hat{x}_t = x_{t-24} \tag{15}$$

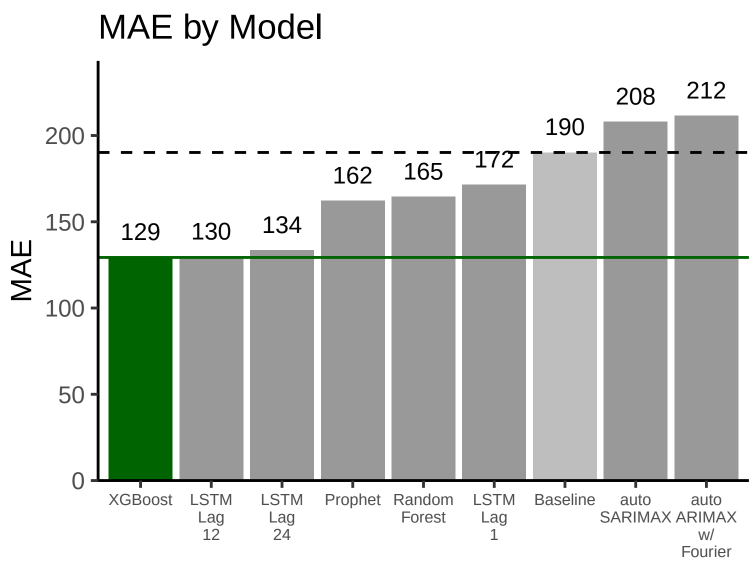

Figure 15: Mean Absolute Error by Model

Table 3 and Figure **??** report the MAE and RMSE for each model. Perhaps unsurprisingly, the machine learning methods performed better than the time series methods, on average. The XGBoost model performed the best (smallest MAE), with $MAE = 129$ bikes demanded per hour. This was followed by LSTM Lag 12 and LSTM Lag 24. Somewhat surprisingly, the Prophet time series model had the fourth-smallest MAE, outperforming the Random Forest (3 units). The baseline SNAIVE model outperformed both ARIMA-based models, which further reinforces the difficulty that ARIMA methods have with such high-frequency data.

Conclusion

References

Prashant Banerjee. A guide on xgboost hyperparameters tuning. URL https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning/notebook.

G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. ISSN 00359246. URL http://www.jstor.org/stable/2984418.

Jason Brownlee. How to use standardscaler and minmaxscaler transforms in python, 06 2020. URL https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/.

Sathishkumar V E and Yongyun Cho. A rule-based model for seoul bike sharing demand prediction using weather data. *European Journal of Remote Sensing*, 53(sup1):166–183, 2020. doi: 10.1080/22797254.2020.1725789. URL https://doi.org/10.1080/22797254.2020.1725789.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(90):3133–3181, 2014. URL http://jmlr.org/papers/v15/delgado14a.html.

Chang Gao and Yong Chen. Using machine learning methods to predict demand for bike sharing. In Jason L. Stienmetz, Berta Ferrer-Rosell, and David Massimo, editors, *Information and Communication Technologies in Tourism 2022*, pages 282–296, Cham, 2022. Springer International Publishing. ISBN 978-3-030-94751-4.

Victor M. Guerrero. Time-series analysis supported by power transformations. *Journal of Forecasting*, 12(1):37–48, 1993. doi: https://doi.org/10.1002/for.3980120104. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/for.3980120104.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735.

Rob Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 3rd edition, 2021. URL OTexts.com/fpp3.

Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(3):1–22, 2008. doi: 10.18637/jss.v027.i03. URL https://www.jstatsoft.org/index.php/jss/article/view/v027i03.

Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL https://www.sciencedirect.com/science/article/pii/S0169207006000239.

Denis Kwiatkowski, Peter C.B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1):159–178, 1992. ISSN 0304-4076. doi: https://doi.org/10.1016/0304-4076(92)90104-Y. URL https://www.sciencedirect.com/science/article/pii/030440769290104Y.

G. M. LJUNG and G. E. P. BOX. On a measure of lack of fit in time series models. *Biometrika*, 65(2):297–303, 08 1978. ISSN 0006-3444. doi: 10.1093/biomet/65.2.297. URL https://doi.org/10.1093/biomet/65.2.297.

Mitchell O'Hara-Wild. *fable.prophet: Prophet Modelling Interface for 'fable'*, 2020. URL https://fable.tidyverts.org. R package version 0.1.0.

Mitchell O'Hara-Wild and Rob Hyndman. *fasster: Fast Additive Switching of Seasonality, Trend and Exogenous Regressors*, 2022. URL https://github.com/mitchelloharawild/fasster. R package version 0.1.0.9100.

Mitchell O'Hara-Wild, Rob Hyndman, and Earo Wang. *fable: Forecasting Models for Tidy Time Series*, 2021. URL https://fable.tidyverts.org. R package version 0.3.1.

Philipp Probst, Marvin N. Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery*, 9(3):e1301, 2019. doi: https://doi.org/10.1002/widm.1301. URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1301.

J. Schuijbroek, R.C. Hampshire, and W.-J. van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257(3):992–1004, 2017. URL https://EconPapers.repec.org/RePEc:eee:ejores:v:257:y:2017:i:3:p:992-1004.

R.H. Shumway and D.S. Stoffer. *Time Series: A Data Analysis Approach Using R*. A Chapman & Hall book. CRC Press, Taylor & Francis Group, 2019. ISBN 9780367221096.

Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018. doi: 10.1080/00031305.2017.1380080. URL https://doi.org/10.1080/00031305.2017.1380080.

UCI. Seoul bike sharing demand data set, Mar 2020. URL https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand.