**Business Problem**

Given Ride Request Data (A dump of all ride requests with their pick up & drop coordinates) from the last 1-year, can we predict how many rides we’ll get in the future?

**How does the data look like?**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ts** | **number** | **pick\_lat** | **pick\_lng** | **drop\_lat** | **drop\_lng** |
| 07-04-2018 07:07 | 14626 | 12.31362 | 76.65819 | 12.2873 | 76.60228 |
| 07-04-2018 07:32 | 85490 | 12.94395 | 77.56075 | 12.95401 | 77.54377 |
| 07-04-2018 07:36 | 5408 | 12.8996 | 77.5873 | 12.93478 | 77.56995 |
| 07-04-2018 07:38 | 58940 | 12.91823 | 77.60754 | 12.96897 | 77.63638 |
| 07-04-2018 07:39 | 5408 | 12.89949 | 77.58727 | 12.93478 | 77.56995 |
| 07-04-2018 07:43 | 5408 | 12.89942 | 77.58733 | 12.93478 | 77.56995 |
| 07-04-2018 07:43 | 50266 | 12.89868 | 77.60434 | 12.87795 | 77.5959 |
| 07-04-2018 07:52 | 58940 | 12.91823 | 77.60754 | 12.96897 | 77.63638 |
| 07-04-2018 07:52 | 58940 | 12.91823 | 77.60754 | 12.96897 | 77.63638 |

**ts** – Time Stamp of the Ride Request

**number** – Customer ID

**pick\_lat** – Latitude of the pick-up location

**pick\_lng** – Longitude of the pick-up location

**drop\_lat** – Latitude of the drop location

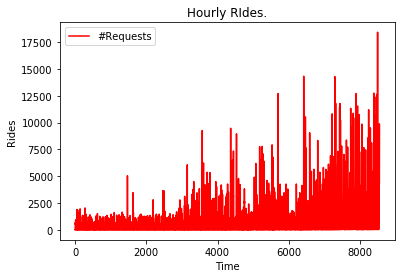
**drop\_lat** – Longitude of the drop location

The rides column does not have any missing data. Nor does it have any erroneous data (negative # rides) etc. Hence, data treatment (Missing/Erroneous value treatment) was not required.

**Exploring the Data & thinking about a Solution**

Let’s take a look at what the data is and what are some of the key components of the given data.

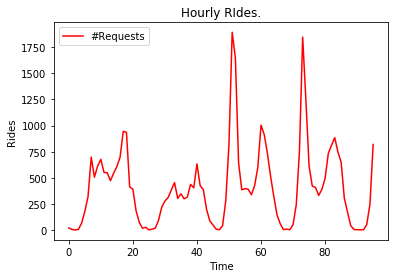
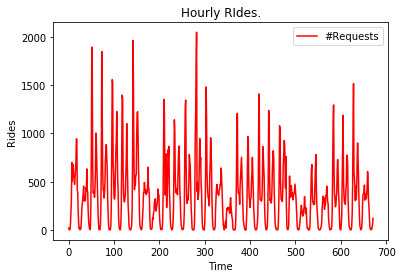
**Trend:**



The above chart depicts the hourly trend of the # ride requests (hereafter shortly referred to as RR) – starting from 04/2018 to 04/2019. Over the course of this 1 year, the level of hourly RR undergoes a slow and steady increase. For instance, 2500 RR an hour in 04/2018 would have been phenomenal whereas the same in 04/2019 is below par as per the given data.

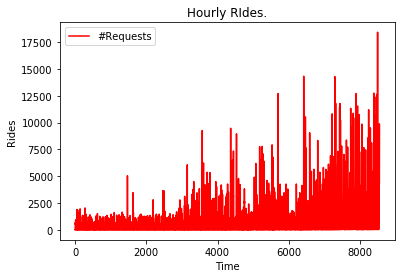
**Seasonality/Periodicity:**

This is the hourly RR data over a period of ~4 weeks starting 04/2018. If you look at the data closely, you’d observe that there is a cyclic pattern of how RR increases and decreases and more interestingly, the time interval between increases and decreases are roughly the same. We commonly refer to this as periodicity or seasonality. So, we can conclude that a strong seasonality is present at a weekly level.



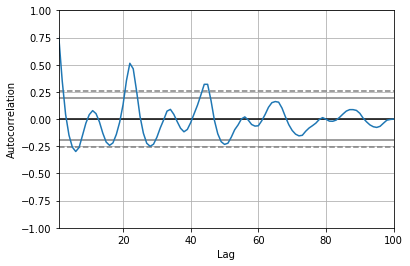
Similarly, the second chart is a zoomed-in version of the first one – spread across 4 days. This again suggests that seasonality is present at an intra-day level as well – we see the RR going up and down within 24 hours.

**Occasional Spikes and troughs:**



The above chart shows some oddities in the Hourly RR data where we see some sudden jumps and some sudden drops.

**Autocorrelation:**



The above graph is the autocorrelation plot. Put in simple words, it tells us to what extent the RR at any given point in time depends on the RR in its recent past. The way to interpret this graph is…more the amplitude of oscillation in the curve, more the correlation to its recent past.

We see that the amplitude dies down around a lag value (x-axis) of 50, thereby suggesting that the RR at any given hour is strongly dependent on the hourly RR from the previous ~2 days.

Given all the above factors, we should keep in mind a few things:

* A good forecasting solution should capture the increasing trend component in the data – the future forecasts after 04/2019 should slowly go upwards compared to before
* It should capture the weekly seasonality patter and more importantly the hourly seasonality - thus, a good time slice to analyse and predict the data at would be at an hourly level
* It should capture the oddities observed in the data
* The solution should try to quantify the relationship between RR at any given time and RR of at least the previous ~2 days

**The Solution**

Now comes the hard part – arriving at a forecasting solution. Before we jump into that, let’s look at what actually could happen with a good demand forecasting solution. Demand forecasting is of utmost importance to any ride hailing service which can help the business in the following ways:

* Efficient capacity planning to ensure that there are no unforeseen shortages or surpluses, thereby avoiding negative customer/captain experience
* Incentivize just the right number of captains to take up more rides during peak times and countering it during non-peak hours
* Refined customer pricing & marketing strategy
* Hardware capacity planning - Under-provisioning may lead to outages or slow service that can create a negative customer experience but over-provisioning can be costly. Forecasting can help find the sweet spot.
* Revenue planning & tracking in advance can give enough buffer time to the business to plan strategies

Now, coming to the solution, like a book-ish data science project, a benchmark solution was anchored to & on top of that, a variety of solutions were tried out to see which one performs best on the given data and is able to explain the intricacies & patters contained in the data.

As is the case with most models, a train-test split was created to ensure that the model does not overfit itself on the given data that its predictions become absurd when measured on unseen data. For some solutions in our arsenal, we didn’t really need a train-test split for some of the more complex solutions, we’ve adopted this so that the model’s performance is generalized.

The following section of the document briefly explains each solution, describes the related codes and files attached along with the document.

1. Moving Average:

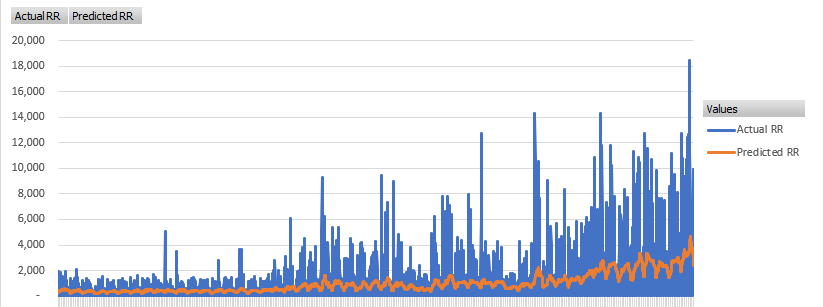
The benchmark solution we have is a very basic moving average solution – where we say that the RR at any given time is simply the average of the previous “N” hourly RR. Inferring from the autocorrelation plot we looked at earlier in the section, we have fixed N to be 48 hourly timesteps = 2 days. Refer to the Benchmark Solution Section in the Jupyter Notebook file for more details on the codes.

Below is a short summary of this solution – the total actual RR in the given time period vs the predicted RR in the given time period and RMSE (how off is our prediction from the actuals).

|  |  |
| --- | --- |
| **Moving Average** | |
| Actual RR | 83,66,405 |
| Avg. Actual RR/hr  (AARRH) | 985 |
| Predicted RR | 82,97,970 |
| RMSE | 1,296 |

*NOTE: The rides quoted do NOT include those data points for which the specified number of lags/timesteps in the solution are not present. This has been adjusted for in both the actuals and predicted to maintain consistency.*

The predicted RR over the 1-year time period is pretty close to the actual RR. This could mislead us into thinking that our solution is great but when we look at the Avg. Actual RR/hour (AARRH) & RMSE, we see that our predictions are off (either higher or lower) by roughly ~1.3k on an average base of ~1k.



What is happening here is…if we want to look at the forecast for a day or a week or a month, this solution would be absurd because of the inaccuracy in prediction. However, these inaccuracies even out over the given time period which is why the overall forecast seems really close to the actuals.

1. Classical Model (ARIMA):

ARIMA stands for Auto Regressive Integrated Moving Average. Not going into the detailed theory behind it, we basically say something like the RR at any given time is a linear function of the previous “N” hourly RR values (AR part) and the previous “M” hourly RR deviations (MA part). While this technique is widely used in forecasting, it has a few issues – one of which is it does not bode well for data where a strong seasonality is present. Unfortunately for ARIMA, that is the case with our data.

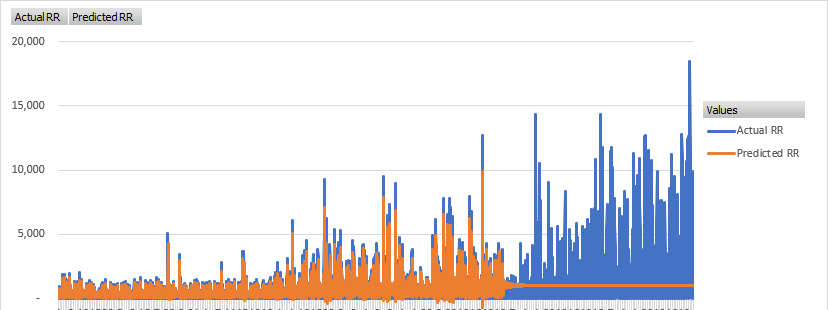
In order to determine some of these parameters of what N and M should be (and others), we have run a simple parametric grid search to get certain combinations which work better than others. Refer to the ARIMA Section in the Jupyter Notebook file for more details on the codes.

Below is a short summary of this solution – the total actual RR in the given time period vs the predicted RR in the given time period and RMSE (how off is our prediction from the actuals).

|  |  |
| --- | --- |
| **ARIMA** | |
| Actual RR | 83,81,556 |
| Avg. Actual RR/hr (AARRH) | 981 |
| Predicted RR | 63,07,082 |
| RMSE | 1,322 |

The predicted RR over the 1-year time period is ~63lac whereas the actual RR is ~84lac. The overall forecast is a bit off which is also confirmed by an AARRH of ~1k and an RMSE of ~1.3K.

As expected, we can see a generic prediction which just oscillates in a very narrow margin. Evidently, ARIMA is also not able to pick up on any seasonal patterns in the test data.



So, we need a better solution as our ARIMA model does not perform well on the given data.

1. Classical Model (Exponential Smoothening):

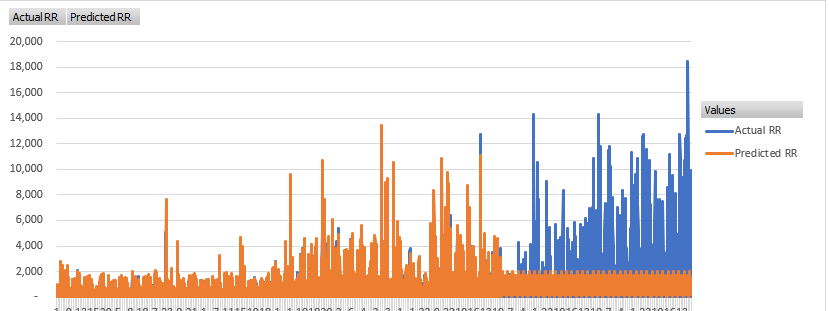
Exponential Smoothening is a classical time series analysis technique where we say that the RR at any given time is a function of the previous “N” hourly RR values in an exponential fashion – i.e. it is strongly dependent on the very previous hourly RR value and as we go further back and back, the dependency gets weaker and weaker. While this technique is widely used in forecasting, the pre-defined packages available are rarely used as a standalone forecasting tool. Generally, ES is hand coded to the raw data before other advanced techniques are used. However, we have used the pre-defined Python implementation for simplicity.

We have again run a very simple parametric grid search to get certain combinations which work better than others. Refer to the Exponential Smoothening Section in the Jupyter Notebook file for more details on the codes.

Below is a short summary of this solution – the total actual RR in the given time period vs the predicted RR in the given time period and RMSE (how off is our prediction from the actuals).

|  |  |
| --- | --- |
| **ES** | |
| Actual RR | 83,81,556 |
| Avg. Actual RR/hr (AARRH) | 981 |
| Predicted RR | 57,17,601 |
| RMSE | 1,351 |

The predicted RR over the 1-year time period is ~57lac whereas the actual RR is ~84lac. The overall forecast is a further off (than ARIMA). However, the individual predictions are slightly better than ARIMA, which is confirmed by an AARRH of ~1k and an RMSE of ~1.3k. As seen below, the predictions oscillate in a very low range as compared to the actuals, thus making the model ineffective.



So, this technique does not perform much better than our previous one and we are still in need of a better solution.

1. Deep Learning (LSTM – Long Short Term Memory):

Deep Learning and LSTMs are the heavyweights when it comes to forecasting. To put very simply, Deep Learning is a subset of Machine Learning where the machine learns patterns and intricacies in the data and uses them for predictions without being explicitly programmed to do so. Deep Learning slightly differs from traditional ML techniques because the way it learns resembles how a human brain/neuron learns.

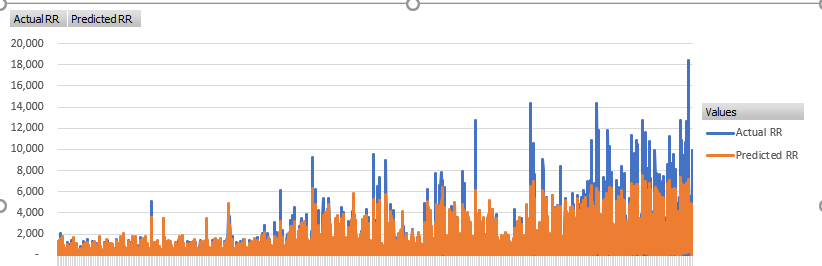
LSTMs in specific, automatically learns from the training data, complex relationships between the RR at any given time is a function of the previous “N” hourly RR values (called lags) unlike some of the previously seen techniques where there is a fixed patter that the relationship has to abide by in order to predict accurately.

To get the optimal set of parameters for the LSTM, we have gone with some industry best practices and intuition. We could not run a parametric grid search due to the computational expense of Deep-Learning techniques like LSTMs. Refer to the Deep Learning Section in the Jupyter Notebook file for more details on the codes.

Below is a short summary of this solution – the total actual RR in the given time period vs the predicted RR in the given time period and RMSE (how off is our prediction from the actuals).

|  |  |
| --- | --- |
| **Overall LSTM** | |
| Actual RR | 82,81,176 |
| Avg. Actual RR/hr (AARRH) | 997 |
| Predicted RR | 74,48,199 |
| RMSE | 677 |

The predicted RR over the 1-year time period is ~74lac whereas the actual RR is ~83lac. The overall forecast is much better than (than ARIMA & ES). Also, the individual predictions are significantly better than all our previous techniques – as observed from an AARRH of ~1k and an RMSE of just 677.



While this is our best solution yet by a mile and we can see (from the below chart) that our LSTM has captured some bit of the huge volatility present in the data which our ARIMA & ES models were not able to, perhaps with better hyperparameter tuning, the model’s performance would improve further.

1. Hybrid Solution – Granular Forecasting through Segmentation & Deep Learning:

Much like the previous solution, this solution uses LSTMs too albeit after a very important step. Since the RR data that we have is quite complex with many innate patterns and characteristics that are hard for models to identify, we have tried to un-knot the entangled knot a bit. In terms of the data available to us, there is nothing except latitudes and longitudes. Hence, we’ve decided that we’ll make the most of what’s there with us. Can these latitudes and longitudes tell us something that wasn’t obvious before? Yes – geography of these ride requests. Since, geography isn’t explicitly defined, let’s try and group these latitudes and longitudes to create regions/clusters where we can make our models focus individually and forecast better.

We’ve resorted to the tried and tested k-Means algorithm (MiniBatchKMeans is implemented because of memory constraints) to segment the raw data we have into different groups based on pick-up latitudes and longitudes. We’ve iteratively arrived (through something known as the elbow method) at the optimal number of groups and that is 12.

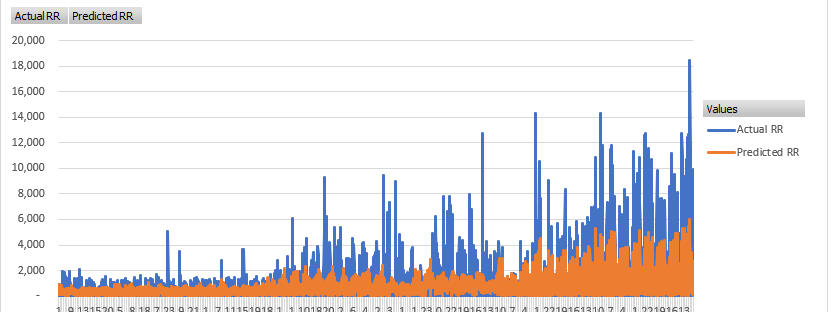
NOTE: The optimal number could be much higher but we had to go with a number that doesn’t make our computer crash. :P Memory constraints, again!

Once we did that, we created the hourly ride data for each of the 12 clusters, created a 70-30 train-test split and train a separate LSTM on each of them while optimizing on the prediction error on the test data. For the LSTM parameters, we have again gone with some intuitive ones – very low number of epochs (iterations) and a much simpler architecture as compared to the overall LSTM. We could not run a parametric grid search for the same reason. Refer to the Granular Forecasting Section in the Jupyter Notebook file for more details on the codes.

Below is a short summary of this solution – the total actual RR in the given time period vs the predicted RR in the given time period and RMSE (how off is our prediction from the actuals).

|  |  |
| --- | --- |
| **Cluster-wise LSTMs** | |
| Actual RR | 83,81,556 |
| Avg. Actual RR/hr (AARRH) | 981 |
| Predicted RR | 68,51,640 |
| RMSE | 151 |

The predicted RR over the 1-year time period is ~69lac whereas the actual RR is ~83lac. The overall forecast is slightly worse than our overall LSTM but far better than any other technique. However, when we look at the accuracy of the individual predictions, we can see that this model phenomenally outperforms every other solution we’ve seen before. While the AARRH is slightly above 100, the RMSE is only 151. This is ~4.5x better than our overall LSTM.



From the below chart, we can see how our set of LSTMs adapts to the highly volatile data in a nuanced fashion. The predictions are closer than ever before and complex patters and intricacies in the data are being captured. The secret lies in the segmentation part – which enables the separation of one complex time series into many smaller, simpler ones and therefore, makes each LSTM’s task of learning the patterns from the data much simpler. Therefore, the LSTMs themselves seem more powerful.

Again, with better parameter tuning for each LSTM, the performance will be magnified significantly. Hence, this solution will be our final solution.

**Further scope of improvement**

The business wanted to predict the future rides & we’ve given them a decent solution that can predict the hourly ride requests. Phew! However, the question we need to ask ourselves is – is this the best that can be done? Certainly not. Here’s why and how!

|  |  |  |
| --- | --- | --- |
| **Type of Improvement** | **Why?** | **How?** |
| Computing Power | Current model is built with a lot of computing restrictions and a blanket LSTM technique | Training models on machines with higher computing power or on distributed servers (using PySpark/MlLib) can help improve the forecast drastically |
| Assumptions | One of the key assumptions is the time slice at which we want to forecast. More the granularity, better it is. | Transition from a hourly time slice to a minute wise time slice, may be? Provided there’s enough data each minute to train a good forecasting model. |
| Input Data | Previous timesteps can’t explain a lot of the complexities in the data | Addition of exogenous data (like weather pattern, special occasions like Diwali, Christmas etc, company’s marketing investment and ant campaign history) will provide the model will a lot more information on why/how the RR varies. It could help explain most of the oddities we observed that the current solutions can’t capture entirely |
| Objective & Model | Predicting the hourly number of rides is good but we are missing out on interaction effects between one ride and another | Currently, most organizations do temporal (time-based) demand forecasting. However, it does not capture how one ride now affects the likelihood of rides nearby in 30 mins or 1 hour etc. Spatio-Temporal forecasting is a highly advanced technique that’s still largely under academic research.  Read further: <https://www.researchgate.net/publication/317710830_Short-Term_Forecasting_of_Passenger_Demand_under_On-Demand_Ride_Services_A_Spatio-Temporal_Deep_Learning_Approach> |