Abstract

To date assignments for MSDS 458 have involved Python, Keras, and TensorFlow for a variety of use cases including text and image recognition along with Natural Language Processing. This paper extends that experience by covering a use case in Time Series along with leverage of the fairly new R Keras functions available through RStudio. In addition, performance between deep learning and traditional time series methods (Hyndman et al., 2019;) are compared to see which provides the lowest model loss. Experiments showed for larger data sets, Keras can deliver about 50% or less loss. While traditional methods based on Hyndman can work better (and faster) for small data sets.

Introduction

Assignment goal was to extend experience with deep learning to a time series problem that might be leverageable in the future by an Internet of Things (IoT) type of device. Since R was being leveraged for the first time with Keras, "pre" experimentation was needed to see how deep learning and traditional methods might respond to different data set sizes and to use those outcomes to pick a set that's just right for both. In addition, prior Python cloud based environments were no longer usable with their GPUs to speed experiment cycles. So, a combination of CPU, Amazon AWS, and Google Colaboratory experiments were attempted to see what environments could be made usable in a reasonable time. Because it was not known what model methods for both types (traditional and deep learning) would be best, a series of best guesses were tried after learning more about the data. Data selected was the JENA climate data set which provided ~420K records of temperature (and 13 other predictors) taken every 10 minutes over 8 years. Methods from both 458 and 413 were used to gain insight into the data and help generate hunches for the model types to attempt along with ways to filter down the data (by $1/6^{th}$) to speed run cycles. Appendix B provides highlights of Exploratory Data Analysis (EDA) along with plots of the data showing multiple seasonal and trend patterns. Finally, it was decided to follow the methodology used in MSDS 413 to select the best model. That is the model within each type and across both types with the lowest validation/test MAE (error/loss) was deemed a best model. As found with prior assignments, getting to the best model was a balance between getting the highest performance (lowest

loss), minimizing overfit (so train and validation/test performance were close), and ending up with the simplest model that could train the fastest.

Literature Review

Material found fell into three categories: background understanding of the R/Keras offering along with similar Python sources to help with transition, compare, and contrast (AWS, 2019; AWS Marketplace, 2019; Climate-Data.ORG, 2019; CRAN R Project, 2019; Chollet et al., 2017/2019; Datacamp, 2019; Hyndman et al., 2019; Keras, 2019; Microsoft, 2019). This first group also provided sample data sets that were used early on to narrow down a set that worked for both Keras and "traditional time series methods" (i.e. Hyndman, 2019). The next group of references helped identify a range of problems (classification, predict, etc.) that could be solved with R and Keras, and provided samples to learn from (Chollet et al., 2017a; Dancho et al., 2018; Google, 2019; Kaggle, 2019; letYourMoneyGrow, 2018; Markin, 2019; The Semicolon, 2018; Wajohi, 2018). The last group provided research from similar weather problems and showed how R and Keras could solve them (Barbounis et al., 2006; Cao et al., 2012; Metroblue, 2019). Some of the material also helped confirm early hunches that weather data was highly seasonal.

Methods and Models

Based on literature review and prior work with traditional time series; 5 groups of deep learning methods were tried (Dense, RNN, GRU, LSTM, dropouts, combinations). Baseline (mean/naïve/etc.) and generally best traditional methods (SFTL, TBATS with multi seasonal, Auto Arima, ETS) were attempted as well. Within groups, parameters/hyperparameters, nodes, dropout, layers, and settings were varied through experimentation to try and further lower validation loss. This was done until it became obvious adding more nodes, layers, etc. wasn't generating gains or better fit. Because the data was so seasonal, traditional baseline methods didn't produce a believable forecast that reflected the seasonality pattern of the data. In pre-experimentation, it was found too little data would allow traditional methods to finish fast but not leave deep learning methods with enough data to train properly. On the flip side, too much data favored deep learning methods, but would cause many of the traditional methods to never complete. One partial solution was to filter the data by $1/6^{th}$, as it was found taking a reading every hour

was as good as six taken every 10 minutes. This helped speed runs but didn't entirely fix the problem of too much data for all method attempted. For example, Auto Arima, a usually "go to" traditional method, would never complete (even with parameters set to their fastest/simplest settings). A combination of R/Keras being new (R/Keras not yet available for Google Colaboratory but coming) and inexperience with AWS (could never get RStudio Server to run on AWS). This forced the use of Mac/CPU for all model runs. The result is some training ran long (up to 11 hours) on the deep learning side (with 3 hours on average). Because each type (traditional and deep learning) were different, features/hyperparameters leveraged were often unique to one side versus the other. For example, deep learning methods had a "lookback" feature that was leveraged to help get to lower loss values. This allowed us to tell a model how far back to look at readings to take those into consideration for forecasts. Doing this uncovered the fact using data points near one another were good predictors of the future. But data points too far back (like months away) weren't a good predictor (especially with the seasonality of the data). Appendix C and the attached code/plots show all combinations attempted, why they were attempted, the result of each experiment, and best model in each group and overall.

Results (Model Evaluation)

For the data set selected, smaller sized (3-6 layer) deep learning models performed better in training and validation/test performance (obtaining much lower MAE error/loss values) than their traditional time series counterparts. To obtain those lower loss values, deep learning models needed to run considerately longer; averaging 3 hours to train (with a range from ½ to 11 hours) versus 1 to 10's of minutes for traditional models that completed. Not all traditional models were able to complete training, and those were excluded from consideration (see Appendix C). Best models for the traditional group were those that favored seasonal data: SFTL and TBATS. While there were multiple seasonal periods in the data, it seemed the overwhelming yearly pattern was the most important to focus on.

For the R/Keras group, combinations of GRU, Dense, and dropouts worked the best (including some Bi-directional GRU). Through experimentation, many of the other R/Keras model methods could be tweaked enough to come close (within 10-15%) to loss values for the best model. Appendix C outlines

the specification and performance of all (best) models and shows how close many of the R/Keras models could be made. It was also found adding more layers and nodes had diminishing returns fairly soon. And going outside the 3-6-layer sweet spot, didn't deliver more, it just increased train time and overfit. Overfit was a common problem with R/Keras models, requiring dropouts to fix the problem. Initially 20 epochs were used to train R/Keras models until it was discovered as few as 5-12 epochs were all that was needed to deliver best performance. Considering training happens infrequently, the extra time spent to train R/Keras models was worth the sizable reduction in loss when compared to traditional models (varying from ½-4/5 lower loss for training, and up to a 20 times reduction in validation loss).

Conclusions

Insights and conclusions gained through this assignment include: With real life usable data amounts, a GPU environment is necessary to allow sufficient experimentation (Appendix E). The lack of a GPU environment greatly reduced the number of "small step" experiments when compared to prior assignments, likely not allowing me to uncover "finer point" options. As models get more complex, train times rise quickly with deep learning. Traditional methods work well for smaller data sets and deep learning for larger data and to drive to better (loss) performance. RStudio/Keras is still somewhat new but spreading. Learnings from prior Python/Keras are easily leveraged in R (pipe model format with "%>% was preferred in most samples found) . Number of samples, references, Kaggle examples and environment options are growing, but still a fraction of their Python/Keras counterparts. Today the Chollet R book is quite influential on samples and papers/code citing that often as a source. And the R/Keras combo can be used to solve all model types covered in MSDS 458 (but may require more work as there are fewer examples to borrow from). The process to evolve a deep learning models for time series (R or Python) is much the same as other model types; EDA, determine min data needed to train well, improve accuracy/loss as you manage over/under fit, and eventually you'll reach saturation and know you've reached the end. The one part that may be unique as in IoT. There can be lots more data requiring smarter filtering and lookback techniques to shrink data while maintaining performance. Both deep learning and traditional methods experimented with benefited well from this practice.

Appendix A - References

List of references used in preparation of this project and report including data sources.

AWS (2019). JENA Climate Data Set. Retrieved from https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip

AWS Marketplace (2019). RStudio Environment Options. Retrieved from https://aws.amazon.com/marketplace/search/results?x=24&y=20&searchTerms=rstudio

Barbounis, T., Theocharis, J., Alexiadis, M., Dokopoulos, P., 2006. Long-term wind speed and power forecasting using local recurrent neural network models. IEEE Transactions on Energy Conversion 21 (1), 273–284. Retrieved from https://ieeexplore-ieee-org.ezproxy.galter.northwestern.edu/stamp/stamp.jsp?tp=&arnumber=1597347

Cao, Ewing, Thompson. (2012). Forecasting wind speed with recurrent neural networks. European Journal of Operational Research, 221(1), 148-154. Retrieved from https://search.library.northwestern.edu/primo-explore/fulldisplay?docid=TN_sciversesciencedirect_elsevierS0377-2217(12)00192-0&context=PC&vid=NULVNEW&search_scope=NWU&tab=default_tab&lang=en_US

Chollet, F., Allaire, JJ. (2018). Deep Learning with R (1st ed.). Manning Publications. Retrieved from https://learning.oreilly.com/library/view/deep-learning-with/9781617295546/kindle_split_000.html

Chollet, F., Allaire, JJ. (Dec 2017a). TensorFlow for R: Time Series Forecasting with Recurrent Neural Networks. Retrieved from https://blogs.rstudio.com/tensorflow/posts/2017-12-20-time-series-forecasting-with-recurrent-neural-networks/

Chollet, F., & Safari, an O'Reilly Media Company. (2017). Deep Learning with Python (1st ed.). Manning Publications. Retrieved from https://learning.oreilly.com/library/view/deep-learning-with/9781617294433/kindle_split_000.html

Climate-Data.ORG (2019). CLIMATE JENA. Retrieved from https://en.climate-data.org/europe/germany/thuringia/jena-2131/

CRAN R Project (2019). R Interface to 'Keras. Retrieved from https://cran.r-project.org/web/packages/keras/keras.pdf

Dancho, M., Keydana, S. (2018, June 25). TensorFlow for R: Predicting Sunspot Frequency with Keras. Retrieved from https://blogs.rstudio.com/tensorflow/posts/2018-06-25-sunspots-lstm/

Datacamp (2019). Forecasting Using R. Rob J. Hyndman. Professor of Statistics at Monash University. Retrieved from https://www.datacamp.com/courses/forecasting-using-r

Google (2019). Welcome to Google Colaboratory!. Retrieved from https://colab.research.google.com/notebooks/welcome.ipynb

Hyndman, R., Athanasopoulos, G, (2019). Forecasting: Principles and Practice. Monash University, Australia. Retrieved from https://otexts.com/fpp2/

Kaggle (2019). A Neural Network Approach to Macroeconomic Time Series Forecasting. Retrieved from https://www.kaggle.com/davidchilders/time-series-prediction-in-r-keras/data

Keras (2019). Keras: The Python Deep Learning library documentation. Retrieved from https://keras.io/

letYourMoneyGrow.com (May 2018). Classifying Time Series with Keras in R : A Step-by-Step Example. Retrieved from https://letyourmoneygrow.com/2018/05/27/classifying-time-series-with-keras-in-r-a-step-by-step-example/

Markin, A. (2019). LTSM time series forecasting with Keras. RPubs. Retrieved from https://rpubs.com/andreasme/keras-lstm-notebook

Metroblue (2019). Climate Jena. Free State of Thuringia, Germany. Retrieved from https://www.meteoblue.com/en/weather/forecast/modelclimate/jena_germany_2895044

Microsoft (2019). Introduction to Azure Data Science Virtual Machine for Linux and Windows. Microsoft Azure Offerings. Retrieved from https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/overview#whats-included-in-the-data-science-vm

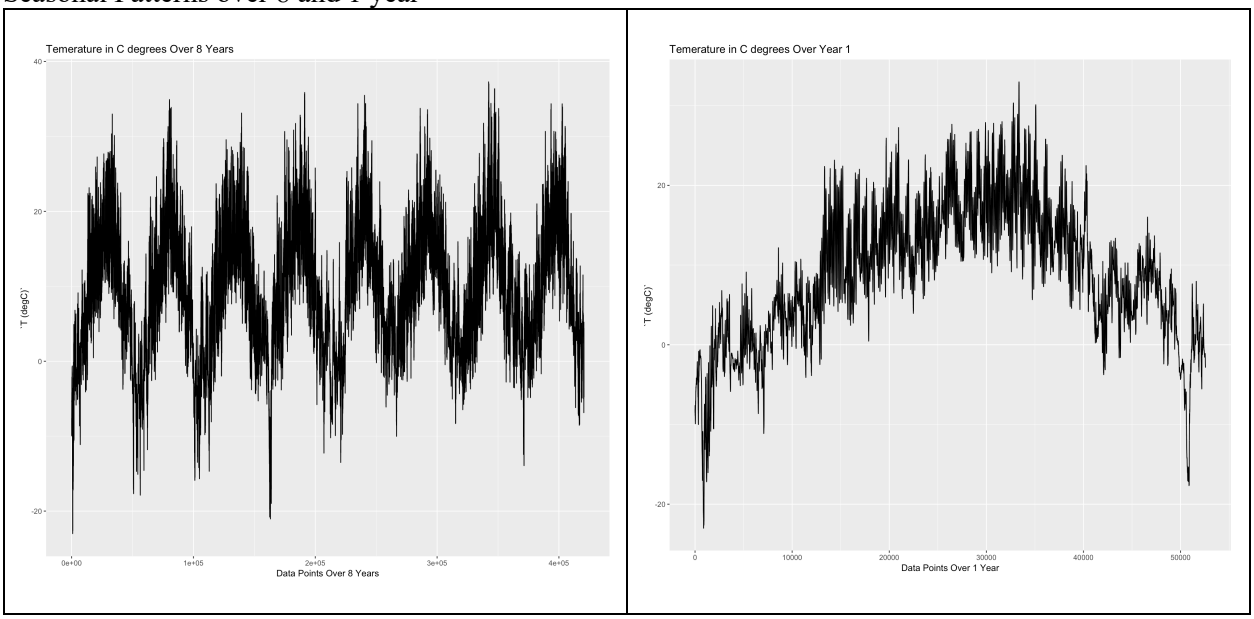
The Semicolon (Mar 2018). Recurrent Neural Networks (LSTM / RNN) Implementation with Keras – Python. Retrieved from https://www.youtube.com/watch?v=iMIWee_PXl8

Wanjohi,R. (2018). Time Series Forecasting using LSTM in R. Retrieved from http://rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-r/

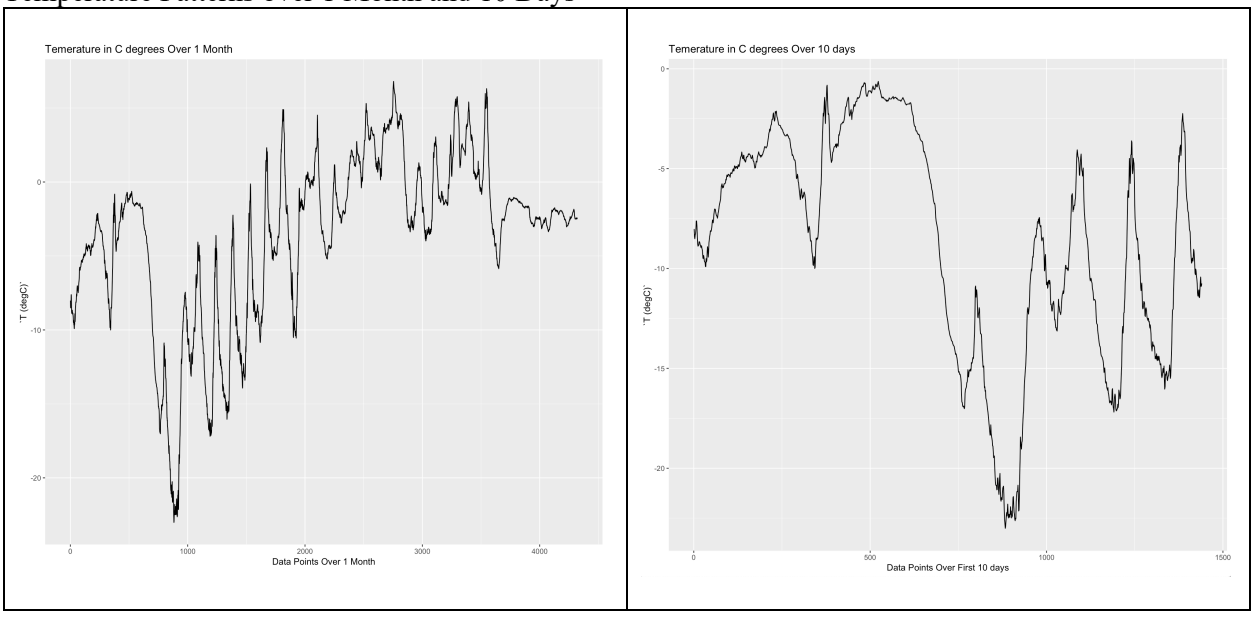
Appendix B – Highlights of EDA and Data Prep: JENA Weather Data Set

Some of the interesting insights gained in EDA included the following; data is very seasonal over the 8 years and the data set had no missing values. The climate represented was cold, as temperature values ranged from -10C to +30C. There are multiple seasonal patterns over the sample period of once every 10 minutes. In addition, data could be filtered down to once and hour (from 420K to 70K records) and not affect model accuracy (as temperatures in the grander scheme of things didn't vary much over 1 hour).

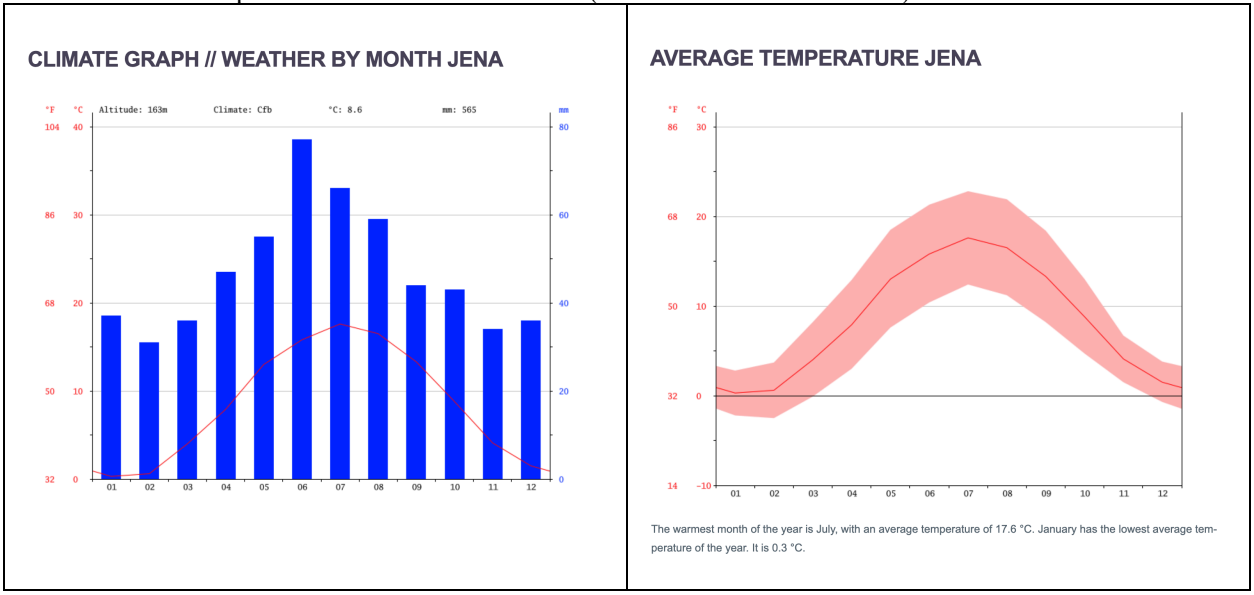
Seasonal Patterns over 8 and 1 year



Temperature Patterns over 1 Month and 10 Days

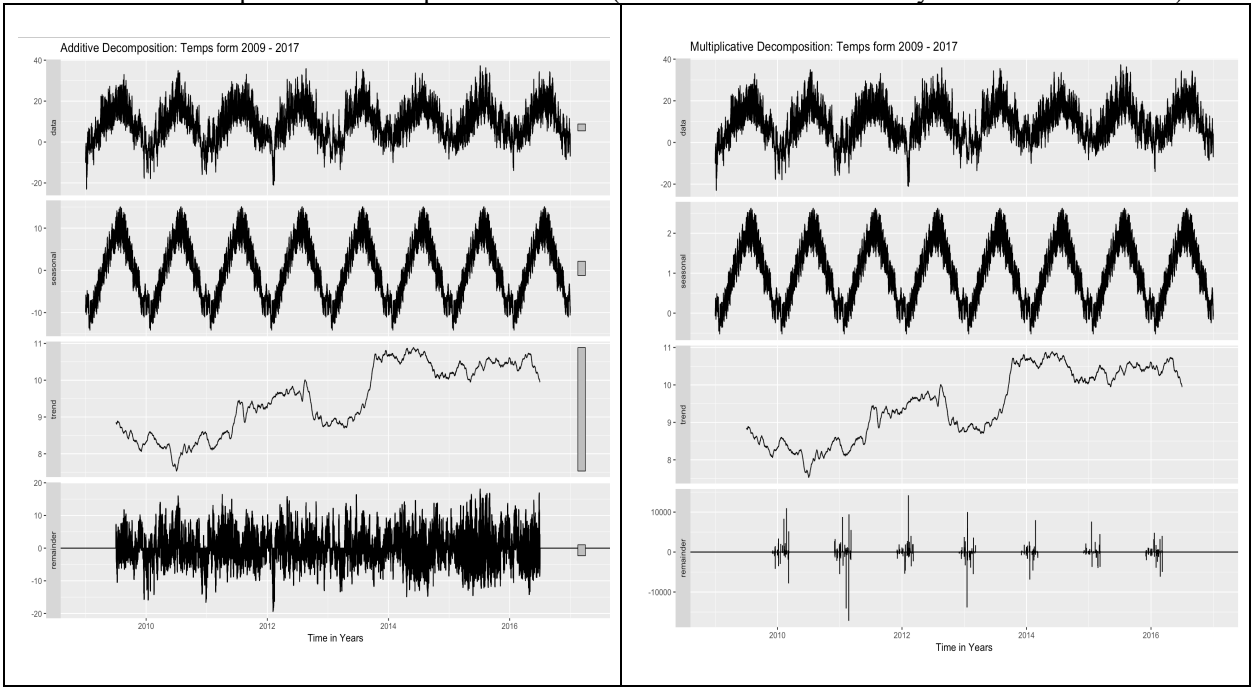


JENA Seasonal Temperature Patterns Over 8 Years (different views from above)



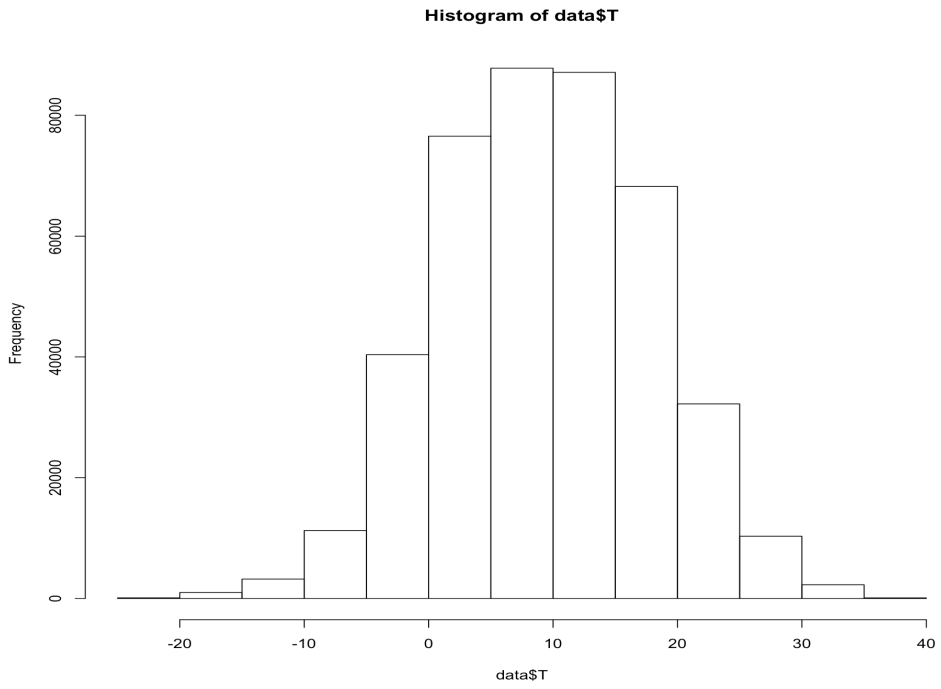
Source: Climate-Data.ORG (2019)

Additional and Multiplicative Decomposition of Data (shows Additive Model may be best w/lowest loss)



Trend over 8 years is temperatures seem to be rising. Is this natural pattern or a sign of the effects of global warming? Smaller residuals for the Additive decomposition indicate a model matched to that pattern may do best. In any case, a seasonal model type, or one that can take into consider one or more seasons in the data is likely to do best with "traditional" methods (like STLF, TBATS, etc.).

High Level Distribution of Temperatures (range, min, max, average over 8 years)



More Detail: Temperature Range/Min/Max Patterns in the Data by Month

JENA WEATHER BY MONTH // WEATHER AVERAGES

	January	February	March	April	May	June	July	August	Sep-tember	October	November	December
Avg. Temperature (°C)	0.3	0.6	4	7.9	13	15.8	17.6	16.5	13.3	8.8	4.1	1.5
Min. Temperature (°C)	-2.2	-2.5	-0.1	3	7.6	10.4	12.4	11.2	8.2	4.7	1.5	-0.7
Max. Temperature (°C)	2.8	3.7	8.2	12.9	18.5	21.3	22.8	21.9	18.4	13	6.7	3.8
Avg. Temperature (°F)	32.5	33.1	39.2	46.2	55.4	60.4	63.7	61.7	55.9	47.8	39.4	34.7
Min. Temperature (°F)	28.0	27.5	31.8	37.4	45.7	50.7	54.3	52.2	46.8	40.5	34.7	30.7
Max. Temperature (°F)	37.0	38.7	46.8	55.2	65.3	70.3	73.0	71.4	65.1	55.4	44.1	38.8
Precipitation / Rainfall (mm)	37	31	36	47	55	77	66	59	44	43	34	36

The difference in precipitation between the driest month and the wettest month is 46 mm. During the year, the average temperatures vary by 17.3 °C.

Source: Climate-Data.ORG (2019)

Appendix C – Model Results

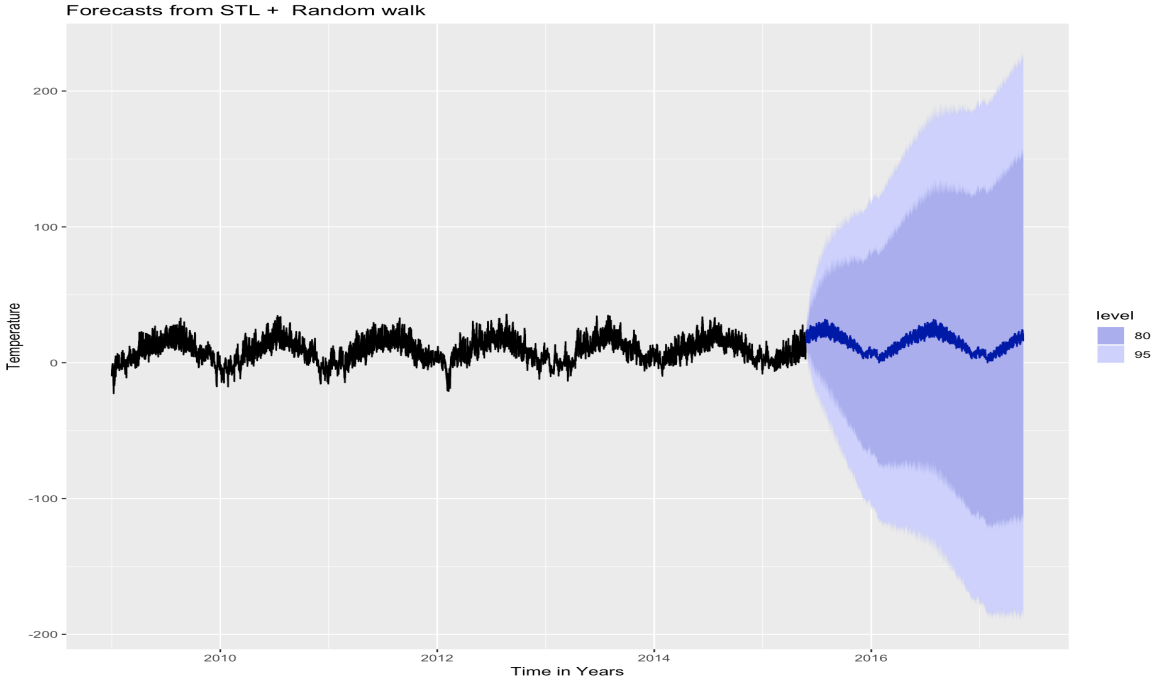
Below is a summary of performance of the best of models attempted for both traditional (using fpp2) and deep learning (using R/Keras). Overall, the deep learning models (except most of the dense models) delivered 1/5 to ½ the loss on training as the fpp2 models. The difference was considerately wider when comparing validation/test loss values (with many deep learning models delivering 10-20 times less loss than traditional models).

Best of Traditional Time Series Models with fpp2

Model	Train Performance Loss/MAE	Validation/Test Loss/MAE	Notes
Baseline – 1a mean	6.879	6.834	Baseline models don't deliver as good performance as seasonal models below. Forecast plots are not believable
Baseline – 1b naive	0.711	8.944	
Baseline – 1c drift	0.711	11.48	
Baseline – 1d Seasonal Naive	Ran forever, never completed		Not usable with large data set and CPU only
Model 1: STL Seasonally Adjusted (see plot below)	0.559	6.183	Best Test Loss, forecast tracks well with seasonal characteristics
Model 2: Auto Arima	Ran forever, never completed		Not usable with large data set and CPU only
Model 3: ETS	0.5462	8.1	2nd Best Train Loss
Model 4: TBATS, Multi-Seasonal	0.51	Not obtained	Best Train Loss

Key: Green: Best, Orange: Second Best

Forecast Pattern for Best Traditional Test MAE Model (fits well, believable forecast follows season)



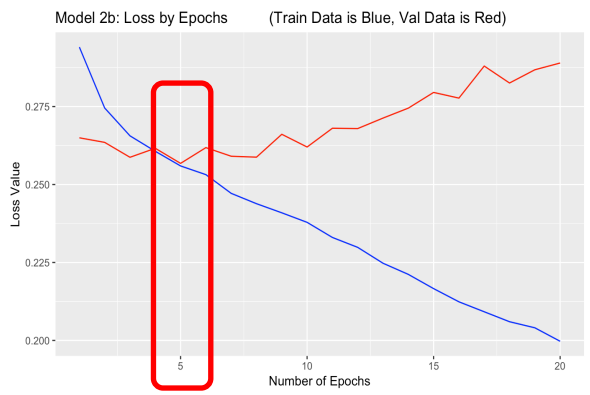
Best of Deep Learning Models with R/Keras

Model	Train Time	Num Epochs	Train Loss (MAE)	Val Loss (MAE)	Notes
1: 2 Dense layers	26.2 min	20	0.1994	0.1994	Best Train Loss
1a: 2 Dense layers	29.1 min	20	.22	1.04	
1b: 3 Dense layers	28.8 min	20	.1344	.3384	
1c: 3 Dense layers, 2 dropout layers between them	29.8 min	20	1.13	1.09	
2: 1 GRU, 1 Dense	2.59 hrs	20	.201	.2878 at 20 epoch .2543 at 5 epoch	2nd best validation loss at 5 epoch. 3rd best Train Loss
2a: 1 LSTM, 1 Dense	2.97 hrs	20	.1748	.3144	
2b: 1 GRU, 2 Dense, dropouts	2.588 hrs	20	.197	.2873 at 20 epoch .2541 at 5 epoch	Best validation loss at 5 epoch. 2nd best Train Loss BEST OVERALL MODEL
2c: 1 GRU, 4 Dense, dropouts	47.35 min	6	.2457	.2647	
2d: 1 GRU, 4 Dense, more nodes, dropouts	1.46 hrs	10	.2678	.2732	
3: 1 GRU, varied hyper parms	2.82 hrs	20	.2745	.2562 at 12 epoch	3rd best validation loss at 10 epoch
4: 2 GRU,1 Dense	6.01 hrs	20	.2646	.2626	
4a: 3 GRU,1 Dense, varied hyper parms	4.39 hrs	8	.2803	.265	
5: 1 Bi-directional GRU, 1 Dense	2.6 hrs	20	.2206	.3813	
5a: 2 Bi-directional GRU, dual dropouts, 1 Dense	6.7 hrs	25	.2658	.2723	While not absolute best, good near best loss for most epochs
5b: 3 Bi-directional GRU, dual dropouts, 1 Dense	7.91 hrs	15	.2529	.2529	Good over all epochs performance
5c: 4 Bi-directional GRU, dual dropouts, 1 Dense	11 hrs	15	.2458	.3114	

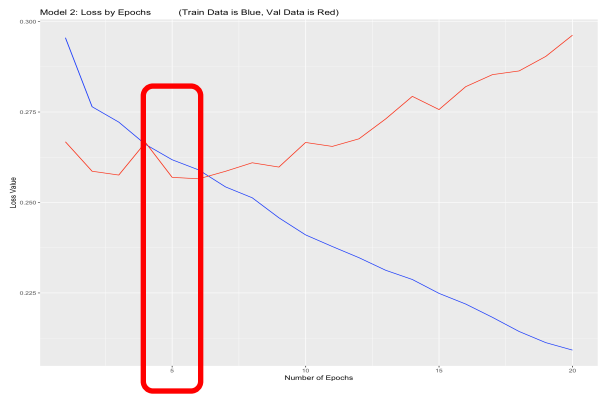
Key: Green: Best, Orange: Second Best, Purple: Third Best, Red: Longest Train Time, Blue: Shortest Train Time

Plots for Best and Better Performing Models

Model 2b:Best at 5 epochs (.2541 validation loss)

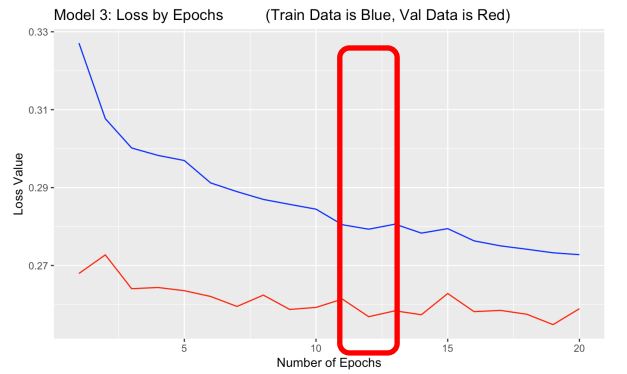


Model 2: Second best at 5 epoch (.2543 at 5 epoch)

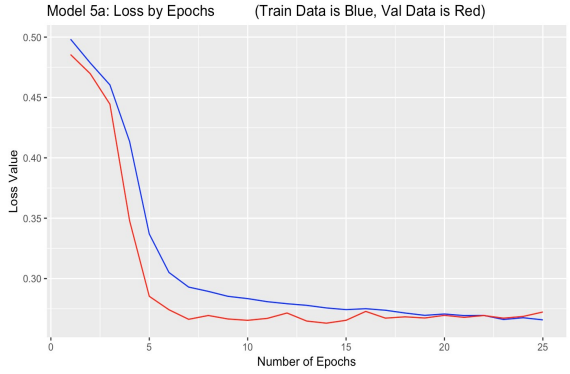


Plots for Best and Better Performing Models

Model 3: Third best at 10 epochs (.2562 at 12 epoch)



Model 5b:Near best loss performance through much of model



Model 2B Architecture

```
#Layer (type)                                 Output Shape                                 Param #
#=================================================================================================================
#gru_6 (GRU)                                  (None, 32)                                   4512
#_________________________________________________________________________________________________________________
#dense_4 (Dense)                              (None, 32)                                   1056
#_________________________________________________________________________________________________________________
#dense_5 (Dense)                              (None, 1)                                    33
#=================================================================================================================
#Total params: 5,601
#Trainable params: 5,601
#Non-trainable params: 0
```

Model 2 Architecture

```
#Layer (type)                                 Output Shape                                 Param #
#=================================================================================================================
#gru_28 (GRU)                                 (None, 32)                                   4512
#_________________________________________________________________________________________________________________
#dense_34 (Dense)                             (None, 1)                                    33
#=================================================================================================================
#Total params: 4,545
#Trainable params: 4,545
#Non-trainable params: 0
```

Model 3 Architecture

```
summary(model3)    # View the summary of the model
#Layer (type)                                 Output Shape                                 Param #
#=================================================================================================================
#gru_29 (GRU)                                 (None, 32)                                   4512
#_________________________________________________________________________________________________________________
#dense_35 (Dense)                             (None, 1)                                    33
#=================================================================================================================
#Total params: 4,545
#Trainable params: 4,545
#Non-trainable params: 0
```

Model 5b Architecture

```
model5b <- keras_model_sequential() %>%
  layer_gru(units = 64, dropout = 0.1, recurrent_dropout = 0.5, return_sequences = TRUE,
            input_shape = list(NULL, dim(data)[[-1]])) %>%
  layer_gru(units = 64, dropout = 0.1, recurrent_dropout = 0.5, return_sequences = TRUE,
            input_shape = list(NULL, dim(data)[[-1]])) %>%
  layer_gru(units = 64, activation = "relu",dropout = 0.1, recurrent_dropout = 0.5) %>%
  layer_dense(units = 1)

summary(model5b)    # View the summary of the model
#Layer (type)                                 Output Shape                                Param #
#=============================================================================================================
#gru_3 (GRU)                                  (None, None, 64)                            15168
#_____________________________________________________________________________________________________________
#gru_4 (GRU)                                  (None, None, 64)                            24768
#_____________________________________________________________________________________________________________
#gru_5 (GRU)                                  (None, 64)                                  24768
#_____________________________________________________________________________________________________________
#dense_1 (Dense)                              (None, 1)                                   65
#=============================================================================================================
#Total params: 64,769
#Trainable params: 64,769
#Non-trainable params: 0
```

Note: Dropouts don't show as separate layers for GRU summary like they may have for Python and other models. To see the dropout values, the reader needs to review the code (sample above and in attached files).

Appendix D – Files Submitted

File Name	Purpose
MSDS458 - Assign 4 Report - Mike Ryder - V2.pdf	PDF version of this report
MSDS458_Assign_4_Mike_Ryder-FPP2.R	R code for the traditional fpp2 models
Traditional fpp2 Model Plots.pdf	Output and plots for the traditional models from above code
MSDS458_Assign_4_Mike_Ryder_V2.Rmd	R Notebook code for the R/Keras models
Deep Learning Model Plots - V3.pdf	Output and plots for the above deep learning models
jena_climate_2009_2016_V2.csv	JENA Climate Data File
MSDS458 – Assign 4 - File List.PDF	This list of files submitted with assignment
MSDS458 – Assign 4 - Mike_Ryder.zip	Zip file with all the above files included

Notes:

Two code files are provided, one for "traditional" time series and the other for deep learning methods.

The two were split to make it clear what libraries and methods were leveraged for each.

The code is fully commented for all models tried along with the capture of text outcomes. Plot outcomes are captured in corresponding separate files and included (as the "knit" process seemed to clear all plots upon save). Models are put together in Groups to help identify what plots go with what code.

Appendix E – Environments Available or Coming

One thing that's been evident throughout MSDS 458 (no matter what base language we use) is environments can be "brittle" and consume much time to keep working or set up. For example, in Python, updating one library can later create what seems like random errors later forcing one to stop model building and turn focus on environments (sometimes for more time than model building). This seems similar to the pain most Data Scientists feel about data cleaning.

In addition, building the best model with real quantities of data requires GPU environments, lots of memory and processing power to be able to iterate quickly enough to get to a best model in a time reasonable to the business. From the MSDS experience, it's evident running on a local system, no matter how big and bad is likely not going to fill the need. What seems like a natural solution to the above problems are cloud environments set up in seconds with perfectly matched libraries and as much power/GPUs one needs in a "pay as you go" model. Through much experimentation, it doesn't seem like we're there yet, especially on the R/Keras side.

The one vendor that's done an excellent job is Google Colaboratory, very fast to sign up and once logged in a free CPU, GPU, and TPU environment is operating in a familiar notebook IDE that works with little problem. At the moment, there isn't an official R/Keras variation to the Python/Keras/Jupyter Notebook (but web rumors say it's coming).

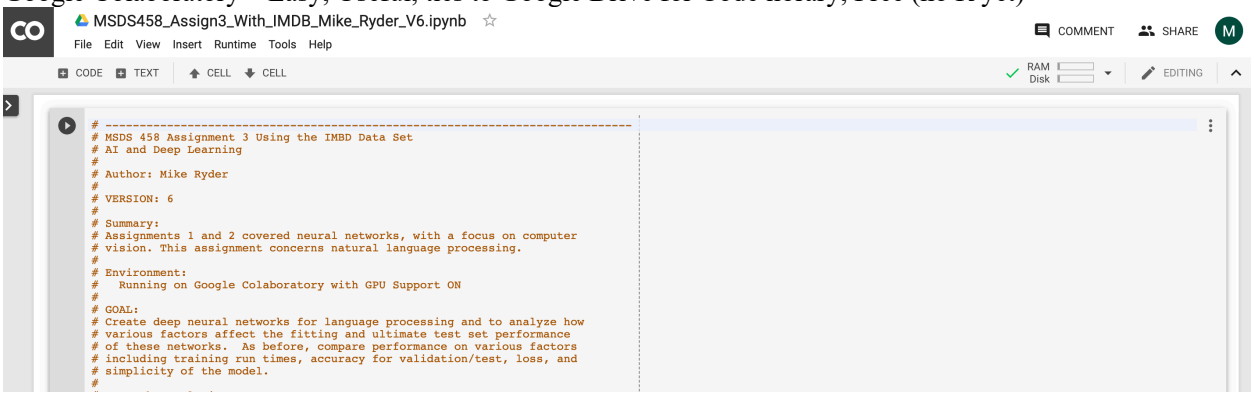
The other vendors that come to mind are Amazon AWS and Microsoft Azure. AWS in particular has many RStudio offerings in its marketplace in what seems like a partnership between RStudio and AWS. However, AWS (and likely Azure) take a considerably more complex process to stand up one of these environments, they can be costly, and the process is far from best in class. What's truly needed is a one button, click on a "recipe" and the environment is up. Because of time, I did experiment to set up several RStudio recommended AWS p2.large environments. However, through much trying, RStudio never came up and access to the environments (and lots of security settings, process work) was quite painful. I did not try the same process with Azure, but the long menu of choices seems more similar to AWS than the one/few click Google offering.

The barrier to set up these environments does seem to get in the way of "selling" more environments and requires specialized AWS knowledge. It's hopeful in time, these will become more one click recipe based,

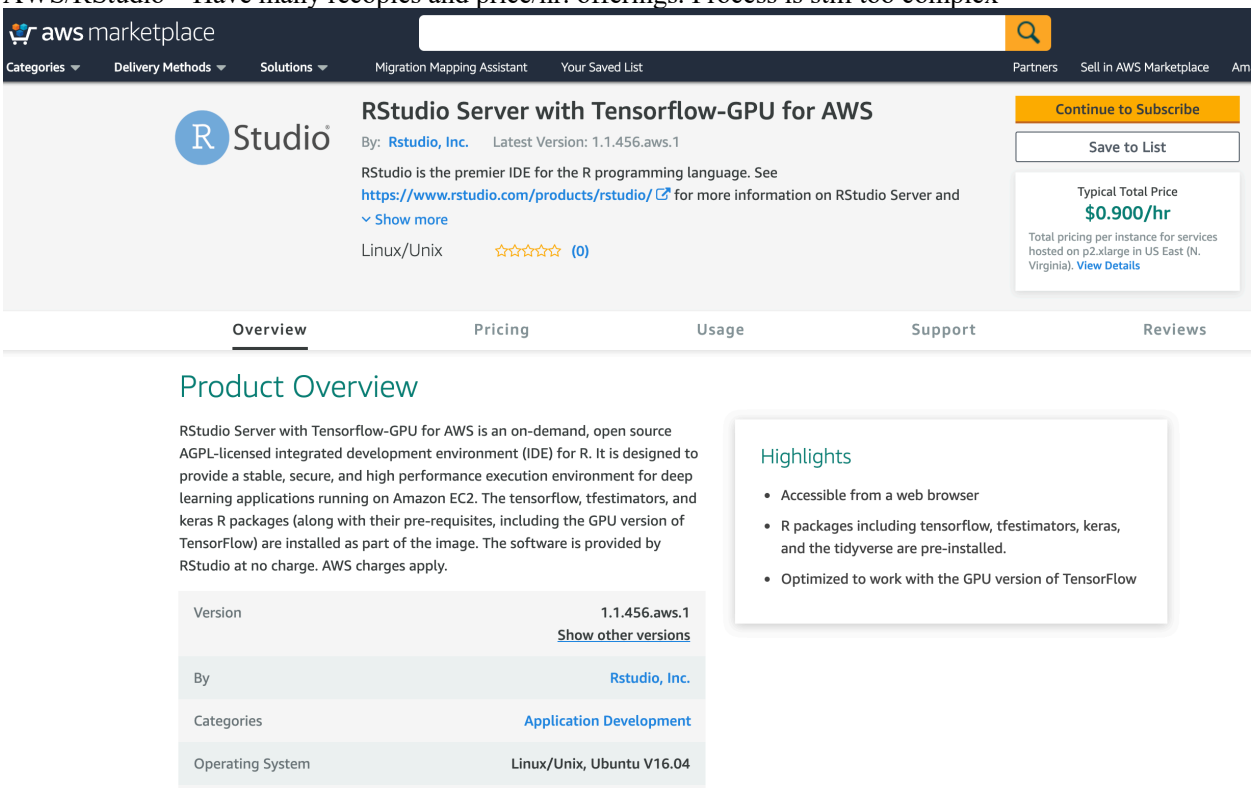
especially helpful to Data Scientists who may not have the Data/Environment Engineering experience and really want to get on to model building.

The reference pages contain links to all 3 environments for the reader to start with. Shown below are some of the more relevant pages to preview what's there today.

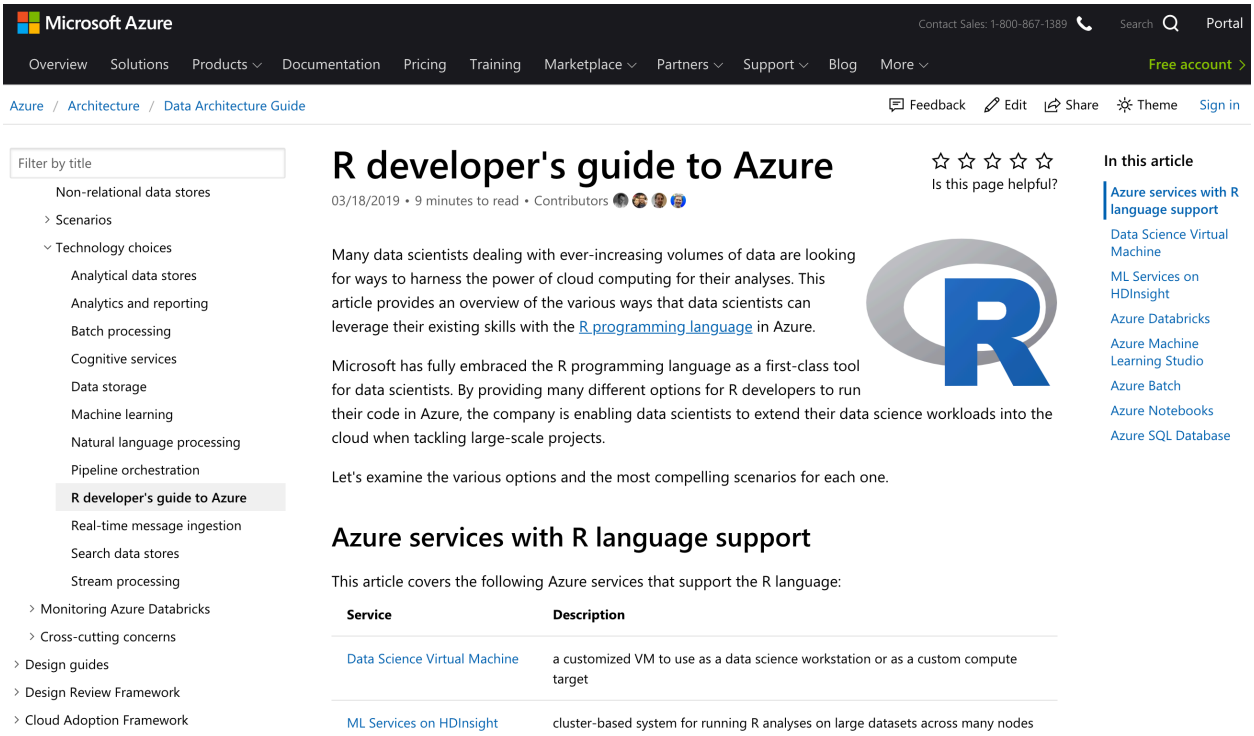
Google Colaboratory – Easy, Useful, ties to Google Drive for Code library, Free (no R yet)



AWS/RStudio – Have many recopies and price/hr. offerings. Process is still too complex



Microsoft takes a model build similar to AWS – Mix of their tools and open source offerings



Azure - Many familiar tools/IDEs *(in retrospect, maybe I should have tried Azure instead of AWS)*

Jupyter Notebook Server with following kernels,	Y	Y
* R	Y	Y
* Python	Y	Y
* Julia	Y	Y
* PySpark	Y	Y
* Sparkmagic	N	Y (Ubuntu only)
* SparkR	N	Y
JupyterHub (Multi-user notebook server)	N	Y
JupyterLab (Multi-user notebook server)	N	Y (Ubuntu only)
Development tools, IDEs, and Code editors		
* Visual Studio 2019 (Community Edition) with Git Plugin, Azure HDInsight (Hadoop), Data Lake, SQL Server Data tools, Node.js, Python, and R Tools for Visual Studio (RTVS)	Y	N
* Visual Studio Code	Y	Y
* RStudio Desktop	Y	Y
* RStudio Server	N	Y
* PyCharm Community Edition	N	Y