M5 Forecasting: Estimation and analysis of the unit sales of Walmart retail goods

Jake Barrett
Queen Mary University of London: School of Electronic Engineering and Computer Science

Gerardo Moreno Ramirez
Queen Mary University of London: School of Electronic Engineering and Computer Science

Jesus Sicairos
Queen Mary University of London: School of Electronic Engineering and Computer Science

Muhammad Hassaan Anwar
Queen Mary University of London: School of Electronic Engineering and Computer Science

I. INTRODUCTION

The Makridakis Competition, or the M Competition, is a series of competitions which first took place in 1982 [6] and is still running with the current one being the M5 Competition. The idea is to test the accuracy of state-of-the-art forecasting methods, compare these methods with each other to gain further insight and seek improvement. Forecasting has played a major role in business and governments to deal with as well as possible, the uncertainty of predictions so the most appropriate actions can be put in place for the future. Consequently, much work is being done in the world of AI and Machine Learning to improve these forecasting methods as much as possible. In this project, we cover some of the time series methods used in past competitions [7][8][9] to investigate the dataset in the M5 Competition provided by Walmart.

First, we build a special kind of RNN model called LSTM which attempts to predict the unit sales of individual stores. This is done by considering a univariate model and then using a more complex multivariate model, which we hope increases the test accuracy. Next, from the scikit-learn library in Python, we build an RF model which analyzes the item HOUSEHOLD_1_272_CA_3_validation, whilst selecting the most important features which help predict the sales of this particular product. We also compare the Random Forest (RF) results with other scikit-learn models. Lastly, we use an ARIMA model to predict the unit sales of the category HOBBIES whilst considering the fact that the dataset may not be stationary and pre-processing may be required before making progress.

II. PROJECT AIMS

- Find appropriate literature to give an insight into appropriate methods for the problem we're to undertake.
- Dissect the dataset to get an overview and determine which features would be beneficial to analyze.
- Once we've determined what features we want to investigate, we'll preprocess the data so that it's fit for doing time series analysis for the RNN, RF and ARIMA models.
- Build the models and utilize the appropriate libraries.
- Find where the models can improve or find other methods to improve the accuracy of the results.
- Analyze and discuss key the findings based on the final results.
- Discuss the limitations and potential improvements found when conducting the project to help the progress of future work.

III. LITERATURE REVIEW

In [9], Makridakis *et al.* aim to discuss the findings of and provide conclusions to the former M4 Competition. They find the best performing model, achieved by Slawek Smyl, was one which utilized an RNN forecasting engine and highlighted its effectiveness. Redd *et al.* [12] overcame one of the main difficulties which Smyl faced, being the slow CPU approach and they're able to achieve similar results but by using GPU implementation which performs up to 322× faster. With this in mind, we'll implement Google's TPU through Colab notebooks where possible when training our RNN model. How deep should you go [10] is a question often posed in the deep learning community [5], so we'll consider this when building our RNN.

RF is one of the highest performing algorithms in time series prediction. RF outperforms ARIMA Model with their predictive ability [4]. Moreover, RFs are ensemble machine learning methods that allows them to produce better predictive performance than other constituent learning algorithms [14]. RFs are an ensemble of B trees $T_1(X), \ldots, T_B(X)$, where $X = \{x_1, \ldots, x_p\}$ is a p-dimensional vector of molecular descriptors or properties associated with a molecule. Outputs of all trees are aggregated to produce one final prediction $\hat{Y}$. For classification problems, $\hat{Y}$ is the class predicted by the majority of trees. In regression, $\hat{Y}$ is the average of individual tree predictions [13].

ARIMA has been widely used to forecast social, economic, engineering and equity problems since it assumes that future values have a functional relationship with past values [11]. As other researchers state, this model can make accurate forecasts in short periods of time [15]. For the purpose of this project, we decided to implement an ARIMA model to predict the total sales of products that belong to a specific category for a single store in California.

Some literature aiding our coding development is the book by Hunt *et al.* [3] which gives advice about developing one's skills in programming. The well-known principle in coding coined by the authors, "Don't Repeat Yourself" (DRY), is one we tried to stick to as much as practically possible when writing our code.

IV. DATA MANAGEMENT AND EXPLORATION

Dataset. Walmart created the files for the M5 competition sales_train_validation.csv, calendar.csv and sell_prices.csv.

The first contains the daily unit sales for all 3,049 products over the last 1,913 days and the date of the first day being 2011-01-29 and the last being 2016-06-19. Each product is categorized by one of 7 departments, 3 categories, 10 stores and 3 states. The second contains columns describing the dates and events for each day a product is sold. The last contains information regarding the price of each item for each store on a certain day. A detailed summary of all features of each file can be seen in Appendix B.

Exploration. Our investigation for the RNN model began by looking at the number of unit sales per day for each Walmart store. We imported the dataset into Python as a pandas data frame. Then, we dropped the redundant columns from the original dataset and summed the unit sales per day and grouped them by *store_id*.

TABLE I. AGGREGATED NO. UNIT SALES PER DAY FOR EACH STORE

Day	CA_1	CA_2	CA_3	CA_4	TX_1	TX_2	TX_3	WI_1	WI_2	WI_3
1	4337	3494	4739	1625	2556	3852	3030	2704	2256	4038
2	4155	3046	4827	1777	2687	3937	3006	2194	1922	4198
▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
1913	6113	6082	7721	3271	4033	4292	3957	4874	5127	4325

Next, we produced separate plots for stores based on the state they're located in. The idea was to visually identify whether stores from the same state follow any obvious trend to justify univariate and multivariate models. We'd then go on to see whether introducing multiple variables has any significant impact on our predictions.

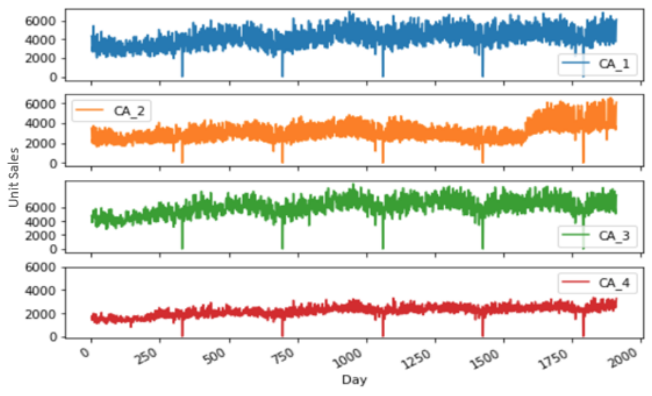


Fig. 1. Subplots for Walmart stores in California.

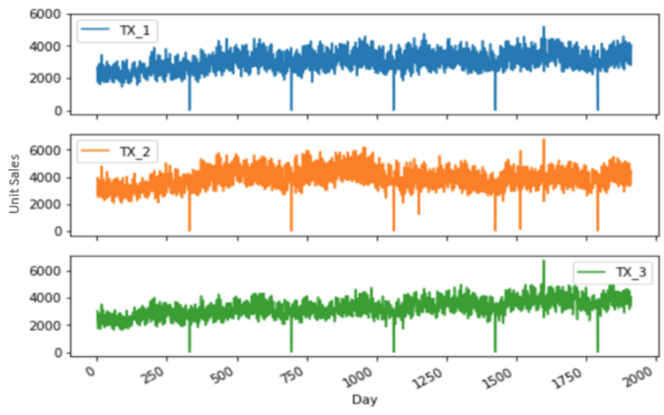


Fig. 2. Subplots for Walmart stores in Texas.

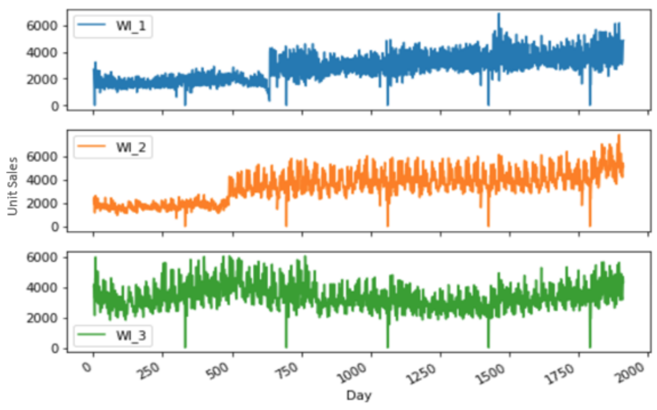


Fig. 3. Subplots for Walmart stores in Wisconsin.

We also explored, for the RF model, the top 10 highest revenue items in Walmart stores for each category and state through the entire history of the dataset. As in Table III, the product FOODS_3_090_CA_3_validation is the most famous food item in CA with total dollar sales of $4988.90, etc.

TABLE II. HIGHEST REVENUE ITEMS PER CATEGORY FOR EACH STATE

id	*cat_id*	*state_id*	*dollar_sales*
FOODS_3_090_CA_3_validation	FOODS	CA	4988.80
FOODS_1_096_TX_3_validation	FOODS	TX	4702.06
FOODS_1_096_WI_2_validation	FOODS	WI	11857.20
HOBBIES_1_354_CA_1_validation	HOBBIES	CA	4343.22
HOBBIES_1_354_TX_3_validation	HOBBIES	TX	14776.14
HOBBIES_1_158_WI_2_validation	HOBBIES	WI	3217.20
HOUSEHOLD_1_106_CA_3_validation	HOUSEHOLD	CA	4512.42
HOUSEHOLD_1_106_TX_1_validation	HOUSEHOLD	TX	2242.02
HOUSEHOLD_1_110_WI_2_validation	HOUSEHOLD	WI	2640.05

Subsequently, we looked at the time series prediction of HOUSEHOLD_1_272_CA_3_validation. We stored the daily sales of this item throughout the timeline in a data frame and transposed the matrix. Then, we joined this item to calendar.csv to get the details of events happening on each day which gave us flexibility in answering questions like why sales in one day was greater than another.

High loss is generally caused by models that are unable to learn from the training data. This means we have either used too few features such that the model is unable to map the data points causing under-fitting or the model is memorizing the mappings such that it may behave poorly on unseen data causing over-fitting. In our case, since we have around 60 features to train on, we're likely to overfit. In Table III, we've created an importance table that allows us to see the importance of each variable in our dataset. The results suggest that variable *d* is most important, representing the day, followed by *snap_CA*, *snap_TX* and *snap_WI*. The most important years were *2011*, *2014* and the most important months were *3*, *1* (March and January).

TABLE III. IMPORTANCE OF EACH VARIABLE

Variable	Importance
d	0.39
snap_CA	0.04
snap_WI	0.04
snap_TX	0.04
Saturday	0.04
2011	0.04
2014	0.04
Wednesday	0.03
3	0.03
Monday	0.02
Sunday	0.02
1	0.02
2	0.02
12	0.02
2012	0.02
2013	0.02

The goal for the ARIMA model was to predict the sales a store makes for products that belong to a specific category; in our case we base our focus on the HOBBIES category of CA_1. We started by loading the data into a pandas data frame and grouped our data by store and then by category. This was possible with the help of the pandas group_by function. Lastly, we added all the values for each day in each group.

TABLE IV. AGGREGATED UNIT SALES FOR HOBBIES FOR THE FIRST 5 DAYS

Day	HOBBIES
1	556
2	498
3	415
4	392
5	268

V. METHODOLOGY

A. RNN

We used RNNs in this project as they're able to capture the patterns of multivariate input data which is suitable for the task we're undertaking [1]. They can easily handle sequential data like ours and it considers the previously received inputs, unlike a typical Feed Forward neural network. We use TensorFlow with the Keras RNN API, which is designed for modelling sequential data for time series problems. We utilize a particular kind of RNN network called LSTM [2]. LSTMs have the same chain-like structure as RNNs but have four interacting layers instead of one single layer as seen in Fig. 4.

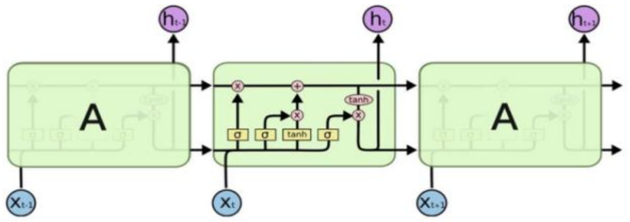


Fig. 4. LSTM chain-like structure with four interacting layers.

We aim to build upon and amend this structure so that it fits the dataset we're working on. We'll consider changing the following to optimize our model; the batch size fed into the network, the number of epochs used for training and the learning rate (LR) of the optimizer in place.

B. RF

To complement the competence of RFs over other machine learning models as mentioned in Section III, we constructed a comparison chart evaluating the performance of scikit-learn models on our training and test sets. To make the predictions, we split 80% of the data for training on scikit-learn's Random Forest Regressor. We used 1,000 estimators with random state set to zero. After fitting the model, we obtained an MAE of 5.85 on our test set. For simplicity, we're representing just 3 layers of the tree.

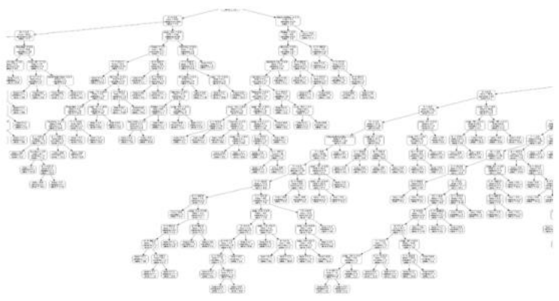


Fig. 5. Example output of a RF Algorithm.

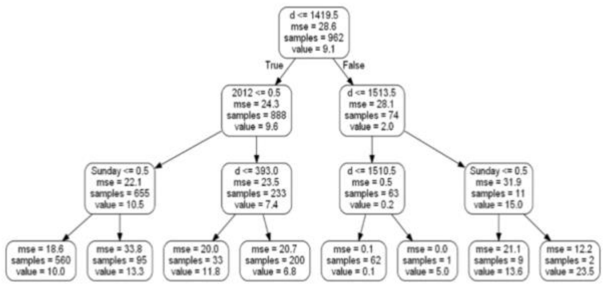


Fig. 6. A sample Decision Tree out of the RF.

C. ARIMA

ARIMA is a forecasting algorithm based on the idea that information in the past of a time series can be sufficient to accurately predict values in the future. The model has the general notation ARIMA(p, d, q), where p is the order of Auto Regressive term and refers to the number of time lags t to be used as predictors, d is the minimum number of differencing required to make a non-stationary series stationary and q is the order of the Moving Average term indicating the required number of lagged forecast errors. It can be represented by the following equation:

$$\phi(L)(1-L)^d X_t = \mu + \Theta(L)\epsilon_t, \tag{1}$$

where ϵ_t is the white noise distribution process $WN(0, \sigma^2)$ and L is the backward-shift operator with:

$$\phi(L) = 1 - \phi_1 L - \cdots - \phi_p L^p, \tag{2}$$

$$\Theta(L) = 1 - \Theta_1 L + \cdots + \Theta_q L^q, \tag{3}$$

with $\phi_p, \Theta_q \neq 0$.

Implementation. We utilize Google's cloud servers and our code was created using Colab notebooks. For training, we use the CPUs Google offer. For simplicity, we use scikit-learn RandomForestRegressor module. For the ARIMA model, we must ensure that our data is stationary because the AR part of ARIMA is a linear regression model. To verify the stationarity, we use the autocorrelation function that is included in the statsmodels.graphics.tsaplots package. The RNN is based on the TensorFlow Keras API but we utilize the TPUs Google offer to speed up more complex tasks in the multivariate case.

VI. Testing and Results

A. RNN

1. Univariate Model

Simple LSTM. We introduce a simple LSTM model with a single 8-unit LSTM layer for a single-step prediction. We shuffle the tensors, cache the dataset and use batches of size 128 to propagate through the network so that training is faster and the memory requirement is less. We use a stochastic gradient descent optimizer which simply computes:

$$\theta_{t+1} = \rho\theta_t - \alpha\nabla_\theta, \tag{4}$$

where θ is the weight vector at a specified time, t is the time being a certain day, ρ is the momentum which helps accelerate the gradient descent process to achieve faster convergence, α is the LR of the optimizer and ∇_θ is the gradient with given weights. Over 100 epochs, we train our model and experiment with the LR using values 0.01 and 0.05.

TABLE V. Architecture of simple LSTM model

Layer Type	Output Shape	Parameters #
LSTM	(None, 8)	320
Dense	(None, 1)	9

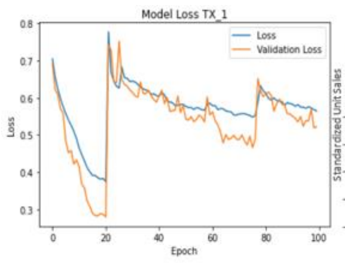
It's common practice to rescale the dataset before training a neural network [16] to help avoid the problem of getting stuck at local optima. We do this by the following standardization:

$$Z = \frac{x-\mu}{\sigma}, \tag{5}$$

where Z is the standardized dataset, x is the original dataset, μ is the mean and σ is the standard deviation of the samples. The loss function we use is MAE, defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{6}$$

where n is the total number of samples, y is the true future, $\hat{y}$ is the model prediction and $i = 1, \ldots, n$. We see the best prediction for the univariate single-step model come from store TX_1 with LR 0.05 as shown in Table VIII and illustrated in Fig. 8 and Fig. 9.



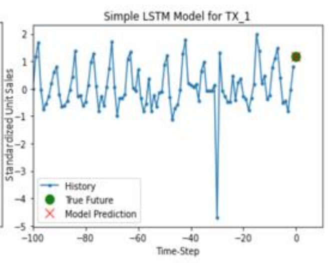
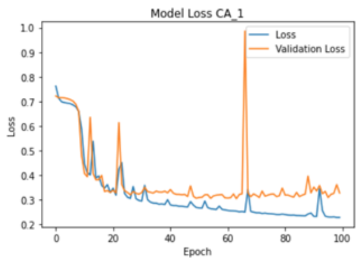


Fig. 7. (Left) Degradation of univariate MAE as epochs increase for TX_1.

Fig. 8. (Right) Simple LSTM univariate single step prediction for TX_1.

We also introduce multiple steps into the univariate time series where we attempt to predict 20 future steps. Since this is a more complex task, we add a 16-unit LSTM layer, with the original LSTM layer increased to 32 and the dense layer to 20. We see the best prediction comes from CA_1 and with LR 0.05 as shown in Table VIII and illustrated in Fig. 9 and 10.



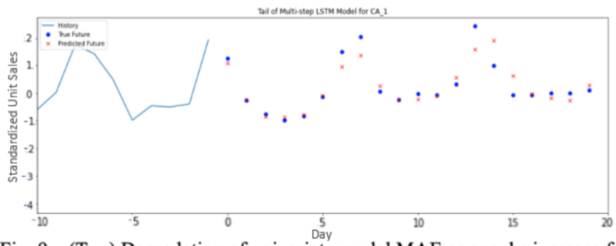


Fig. 9. (Top) Degradation of univariate model MAE as epochs increase for CA_1.

Fig. 10. (Bottom) Simple LSTM univariate model predictions for CA_1 showing the tail of the plot.

TABLE VI. Architecture of univariate multistep LSTM

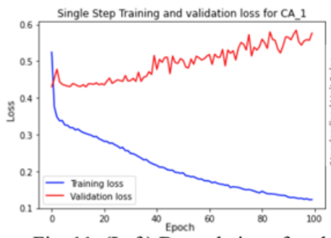
Layer Type	Output Shape	Parameters #
LSTM	(None, 100, 32)	4352
LSTM	(None, 16)	3136
Dense	(None, 20)	340

2. Multivariate Model

LSTM. Since the multivariate task is a higher order of complexity, for the single-step model, we increase the number of units in the LSTM from 8 to 32. All other parameters of the model remain, with the only experimental adjustment being the LR. When doing the multivariate prediction for a store, we only use the data in stores which belong in the same state, meaning when we make a prediction on the unit sales of TX_1, we only consider the data in stores TX_1, TX_2 and TX_3. For the multivariate, multistep model, we adopt the same architecture as the single-step.

TABLE VII. Architecture of multivariate LSTM

Layer Type	Output Shape	Parameters #
LSTM	(None, 32)	4736
Dense	(None, 1)	33



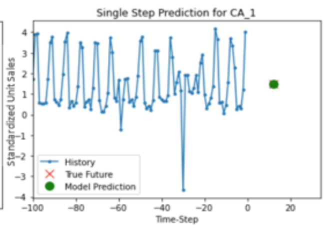
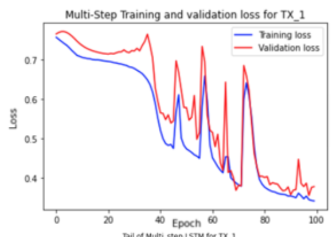


Fig. 11. (Left) Degradation of multivariate MAE as epochs increase for CA_1 LR 0.05.

Fig. 12. (Right) LSTM multivariate single step prediction CA_1 LR 0.05.



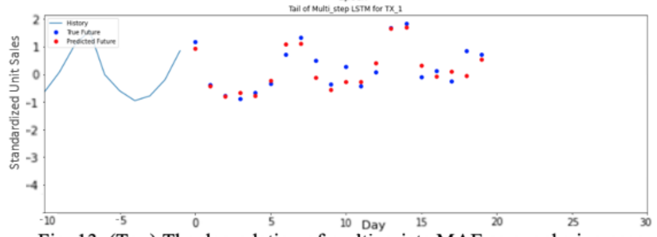


Fig. 13. (Top) The degradation of multivariate MAE as epochs increase store TX_1 LR 0.01.
Fig. 14. (Bottom) LSTM multivariate multi step predictions for TX_1 showing the tail of the plot LR 0.01.

TABLE VIII. MODEL PERCENTAGE ERROR OF LTSM MODELS COMPARED WITH BASELINE FOR SINGLE/MULTI-STEP AND UNI/MULTIVARIATE MODELS

Percentage Error of Model Predictions* (%)	Single step				Multi-step (20 steps)			
	Univariate		Multivariate		Univariate		Multivariate	
Store ID	LR 0.01	LR 0.05	LR 0.01	LR 0.05	LR 0.01	LR 0.05	LR 0.01	LR 0.05
CA_1	10.75	**3.11**	1.64	**0.11**	6.76	**5.66**	**7.11**	9.36
CA_2	1.63	**0.58**	13.39	**9.46**	**6.12**	6.32	**6.49**	11.18
CA_3	**0.47**	4.04	11.99	**7.46**	**5.98**	6.77	8.82	**7.84**
CA_4	1.11	**0.79**	**1.64**	2.07	**6.50**	6.86	**5.90**	8.12
TX_1	0.93	**0.03**	3.48	**2.07**	**6.68**	8.14	**5.42**	12.88
TX_2	**0.40**	2.02	**1.90**	9.95	**7.45**	15.26	8.76	**8.10**
TX_3	2.12	**0.28**	**3.98**	5.01	**7.08**	8.35	**6.36**	8.81
WI_1	7.07	**5.02**	**12.81**	15.68	9.94	**9.76**	**9.52**	10.51
WI_2	3.77	**0.06**	**12.95**	18.49	**11.64**	13.31	**11.62**	11.87
WI_3	3.15	**2.78**	14.72	**13.34**	9.94	**8.75**	**9.61**	11.90

* The best % of 3 model runs were considered using different seeds. The numbers in bold represent the lowest % error between LR 0.01 and 0.05 for each model and on each store.

From the results in Table VIII, we find for the multi-step models that decreasing the LR generally leads to improvement in predictive performance on each store. It is unclear whether this improvement exists with the single-step models. In fact, the univariate model favors the higher LR. We see in the single-step case, for 9 out of 10 stores, the univariate model achieves the lowest percentage error (over LR 0.01 or 0.05) than the multivariate model, which may mean we're overcomplicating matters by fitting redundant features. This cannot be said for the multi-step case, where 5 out of 10 stores for the multivariate model perform better than their univariate counterpart. This leads us to believe that for multi-step, for some stores, we under-fit the data by oversimplying the problem using a univariate model but we manage to capture the trend of the data better by using a more complex multivariate model.

B. RF

We set a threshold of 0.2, so that the variables which have importance factors greater than 0.2 will be considered for analysis as in Table III. We analysed the item HOUSEHOLD_1_272_CA_3_validation and when we fitted our model, we realized that reducing the number of features had a bad impact on our MAE, causing it to increase to 11.44.

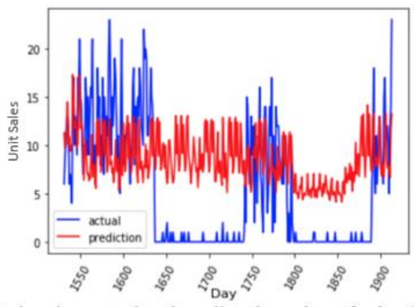


Fig. 15. Plot showing actual and predicted number of sales (y) for each day (x) for item HOUSEHOLD_1_272_CA_3_validation.

To improve our results, we used several techniques:

Data Scaling. The input features were scaled using Sklearn with the MAE being 8.49 after fitting the model and calculating predictive labels.

Feature Selection. To increase the performance, we chose another group of features now with "*snap_CA*", "*snap_TX*", "*snap_WI*", "*Chanukah End*", "*Christmas*", "*Columbus Day*"," *LentStart*," ,"*VeteransDay*", "*Day of the week*", "*month*" and "*year*". The month, year and day of the week were all one-hot encoded.

Test Split Sizes.

a. Test Split 20%

The data frame we obtained after feature selection step had 35 columns. The test size was set to 20%, so that 80 % of the data should be used for training. We saw a decrease of the MAE to 5.94.

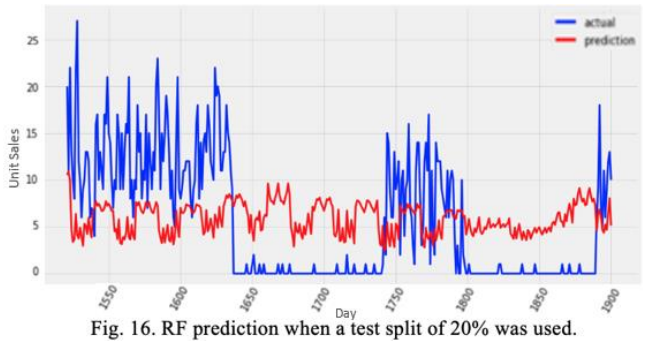


Fig. 16. RF prediction when a test split of 20% was used.

b. Test Split 1%

Here, we took a test split of 0.01, meaning that 99% of the data would be used for training. The result obtained was far better than the previous case with test MAE of 4.33. The results indicate that, with greater test split, we're compromising on our training quality since less data is available for training. This is shown through the decrease of the MAE when we decreased the test split.

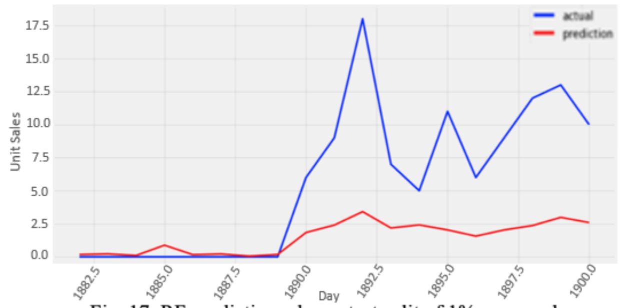


Fig. 17. RF prediction when a test split of 1% was used.

Comparing with other Scikit Models. We were aware that other models in scikit-learn could potentially perform better than RF. To check this, we evaluated the performance of all models in scikit-learn library against our data as in Table IX.

TABLE IX. EVALUATION METRIC SCORES FOR SCIKIT-LEARN MODELS

Model Name	r2_test	r2_train	rmse_test	rmse_train
LinearRegression	0.06	0.30	5.32	4.88
SGDRegressor	-5.08E+28	-1.26E+25	1.23974E+15	2.0658E+13
LogisticRegression	1.03	2.12	7.83	10.27
VotingRegressor	1.03	2.12	7.83	10.27
GradientBoostingRegressor	0.42	0.66	6.54	3.40
RandomForestRegressor	0.93	0.93	7.64	1.50
BaggingRegressor	-0.94	0.91	7.66	1.71
ExtraTreesRegressor	0.96	1.00	7.70	0.00
AdaBoostRegressor	0.39	0.34	4.29	4.71
DecisionTreeRegressor	0.91	1.00	7.60	0.00
SVR	1.25	0.16	8.25	5.34
MLPRegressor	0.47	0.27	4.01	4.98
LGBMRegressor	0.87	0.73	7.53	3.00
Bayesian Ridge Regression	0.13	0.29	5.12	4.91

Bayesian Ridge Regression (BRR). It's evident that BRR is one of the best performing models. To further understand how this model performs on our dataset, we decided to visualize its performance on different test splits.

a. Test Split 20%

Using BRR with a test split of 0.2, we obtain an MAE of 5.84.

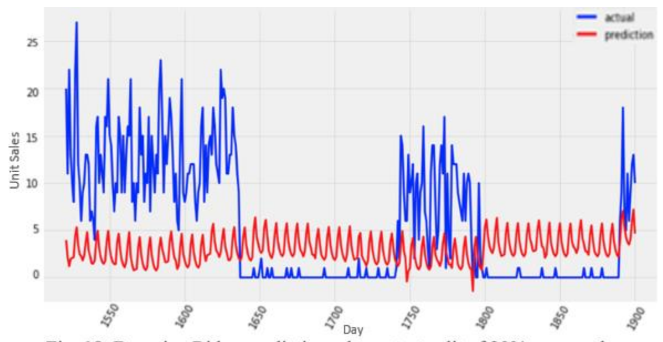


Fig. 18. Bayesian Ridge prediction when a test split of 20% was used.

b. Test Split 1%

Now, using a test split of 0.01, our results are better. We obtain an MAE of 3.90 which is a significant performance increase compared with the same split for the RF prediction.

Experiments clearly show that BRR and RFs both need more data for reliable predictions. If we try to increase the test split, this will give the model less data to train on, making the predictive performance worse but having a higher certainty of it. If we decrease the test split, we'll get a better prediction due to more training data but higher variance results for unseen data.

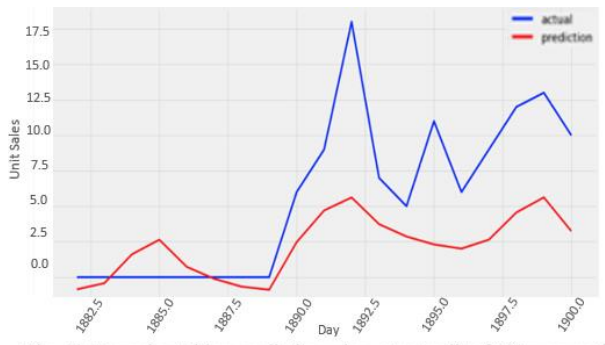
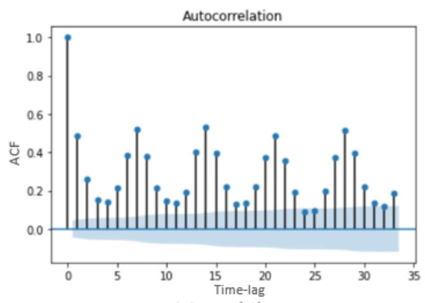


Fig. 19. Bayesian Ridge prediction when a test split of 1% was used.

C. ARIMA

We first checked whether the dataset was in fact stationary. In Fig. 20, we found that it wasn't stationary since the bars after time-lag 0 don't lie within the critical interval. To fix this, we used differencing and depending on the complexity of the series, it may be needed more than once, as in Fig. 21. We made use of a built-in function called `diff` and were able to implement our ARIMA model to make some predictions.



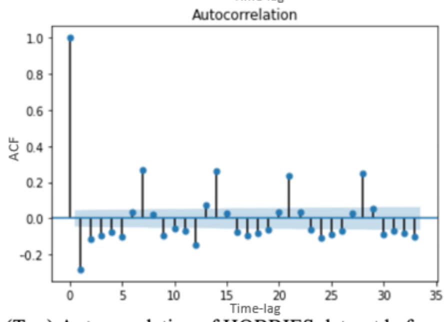


Fig. 20. (Top) Autocorrelation of HOBBIES dataset before differencing.
Fig. 21. (Bottom) Autocorrelation of HOBBIES dataset after differencing.

We then use the ARIMA function that comes with the `statsmodels.tsa.arima_model` package. We used only 365 days of historic data, since ARIMA models tend to perform better in shorter timeframes. We split the 365 days into training and testing sets, approximately 90% and 10% respectively. To train the model, we send the train data as a parameter and specify each parameter required for the ARIMA(p, d, q) model. After this, we fit our model by calling the `fit` function on our trained model. To make predictions, we call the function `forecast` on our fitted model and specify the number of predictions we want to make. In our case, we set that number to 24 with our prediction shifted to better fit the data.

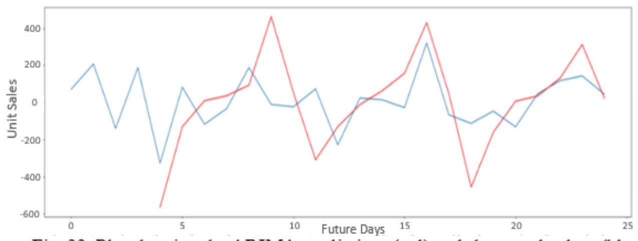


Fig. 22. Plot showing the ARIMA predictions (red) and the actual values (blue).

We observe that our model is able to capture the pattern of the data to some extent. The MSE obtained with this approach was 239.97. So, we decided to implement another model that works in a similar way to see if we could lower that value. We opted for the autoregressive (AR) model. The process is similar to the ARIMA model with the only difference being the training of the model. We use the AR function instead and we only send the training data as a parameter. Again, we fit our model and finally we make predictions by calling the predict function on the fitted model.

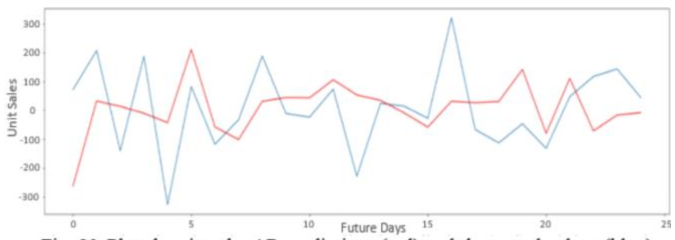


Fig. 23. Plot showing the AR predictions (red) and the actual values (blue).

We're able to see an improvement in performance compared to that of the ARIMA model, achieving an MSE of 131.57.

VII. Conclusion

From the analysis using the RNN model, we find that a more complex multivariate model better captures the trend of the sales for only half of the Walmart stores than the simpler univariate model which only considers the sales of one store. We find that RF was not the best model when predicting the HOUSEHOLD_1_272_CA_3_validation product. Other scikit-learn models such as BRR has improved performance. Lastly, we found that ARIMA, when predicting the sales in the category HOBBIES and implementing differencing to the dataset, is able to capture the basic trend but not as well as the AR model.

Limitations. We found that with building deeper RNNs, computational complexity increases and therefore so does training time or the resources required. ARIMA required a fair amount of preprocessing of the data in order to eventually build a model fit for training, specifically with differencing to ensure the model is stationary. We found with RF, finding the best split for reliable training yet good test accuracy proved to be a challenge.

Future work. If we could do further analysis, we would utilize Google's TPUs further to build a deeper, more complex RNN model since time-series forecasting is not a simple task. It's unclear from our results that introducing a more complex multivariate model improves the performance. It would be interesting to see if a more complex model than the one we implemented would improve performance. However, a deep model isn't necessarily a better one [10]. Moreover, we could've tried a hybrid model, as Smyl in the M4 competition "*which blended exponential smoothing and ML concepts into a single model of remarkable accuracy*" [9] In the M4 competition paper, discussed is the "*best model versus combining against single methods*" where Makridakis *et al.* consistently find higher accuracy from combining than from a single model. Combining methods together produces diversity within a model, which helps cancel out random noise, thus leading to improved performance. Hence, combining RF, ARIMA and RNN could lead to even better results.

References

[1] H.Hewamalage *et al.*, "Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions."

[2] S.Hochreiter *et al.*, "Flat Minima," *LONG SHORT-TERM MEMORY*, Jan.1997.

[3] A.Hunt *et al.*, *The pragmatic programmer: from journeyman to master*. Boston:Addison-Wesley, 2015.

[4] M.J.Kane *et al.*, "Comparison of ARIMA and Random Forest time series models for prediction of avian influenza H5N1 outbreaks," *BMC Bioinformatics*, Aug.2014.

[5] Y.LeCun *et al.*, "Deep learning," *Nature*, May.2015.

[6] S.Makridakis *et al.*, "The accuracy of extrapolation (time series) methods: Results of a forecasting competition," *Journal of Forecasting*, Apr.1982.

[7] S.Makridakis *et al.*, "The M2-competition: A real-time judgmentally based forecasting study," *International Journal of Forecasting*, Apr.1993.

[8] S.Makridakis *et al.*, "The M3-Competition: results, conclusions and implications," *International Journal of Forecasting*, 2000.

[9] S.Makridakis *et al.*, "The M4 Competition: 100,000 time series and 61 forecasting methods," *International Journal of Forecasting*, Jan.2020.

[10] E.Malach *et al.*, "Is Deeper Better only when Shallow is Good?," Mar.2019.

[11] J.N.K.Rao *et al.*, "Time Series Analysis Forecasting and Control," Sep.1972.

[12] A.Redd *et al.*, "Fast ES-RNN: A GPU Implementation of the ES-RNN Algorithm."

[13] V.Svetnik, *et al.*, "Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling," *Journal of Chemical Information and Computer Sciences*, Nov.2003.

[14] Y.Takefuji *et al.*, "Effectiveness of ensemble machine learning over the conventional multivariable linear regression models," Jan.2016.

[15] H.Tanaka, "Fuzzy data analysis by possibilistic linear models," *Fuzzy Sets and Systems*, Dec.1987.

[16] W.Williams *et al.*, "SCALING RECURRENT NEURAL NETWORK LANGUAGE MODELS."

Appendix

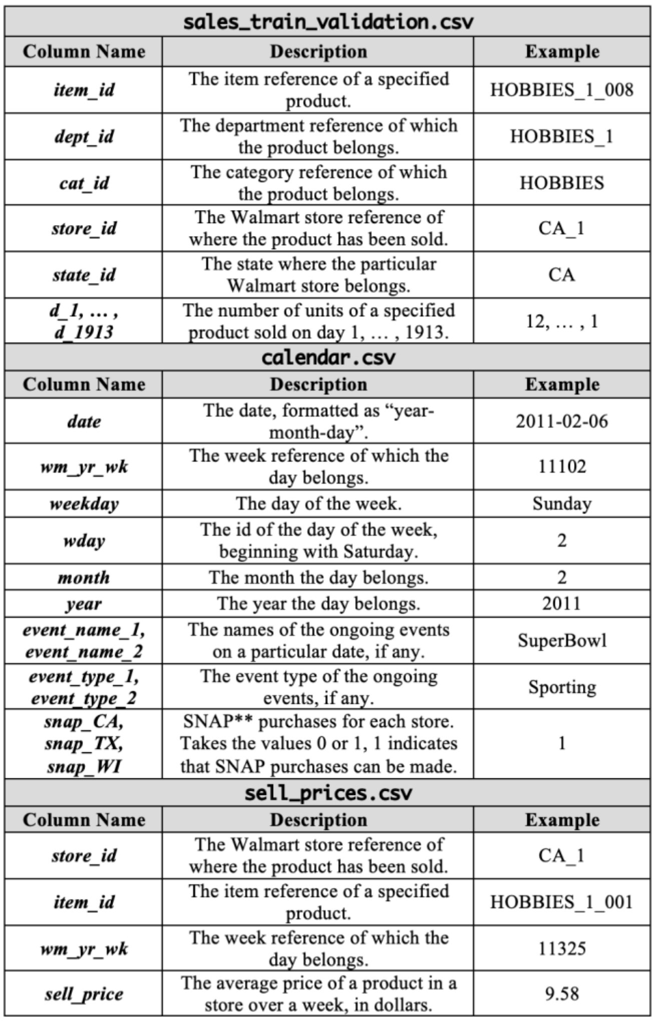
A. Code Implementation

We used notebooks to create our well-documented codes so that it is easy for the reader to follow. They were used to generate the results for each model and can be found on our GitHub repository:

https://github.com/GerardoMoreno96/ECS784P_DataAnalyticsProject

B. The Dataset*

TABLE I. DESCRIPTION OF EACH COLUMN IN THE DATASET FILES



sales_train_validation.csv		
Column Name	**Description**	**Example**
item_id	The item reference of a specified product.	HOBBIES_1_008
dept_id	The department reference of which the product belongs.	HOBBIES_1
cat_id	The category reference of which the product belongs.	HOBBIES
store_id	The Walmart store reference of where the product has been sold.	CA_1
state_id	The state where the particular Walmart store belongs.	CA
d_1, … , d_1913	The number of units of a specified product sold on day 1, … , 1913.	12, … , 1

calendar.csv		
Column Name	**Description**	**Example**
date	The date, formatted as "year-month-day".	2011-02-06
wm_yr_wk	The week reference of which the day belongs.	11102
weekday	The day of the week.	Sunday
wday	The id of the day of the week, beginning with Saturday.	2
month	The month the day belongs.	2
year	The year the day belongs.	2011
event_name_1, event_name_2	The names of the ongoing events on a particular date, if any.	SuperBowl
event_type_1, event_type_2	The event type of the ongoing events, if any.	Sporting
snap_CA, snap_TX, snap_WI	SNAP** purchases for each store. Takes the values 0 or 1, 1 indicates that SNAP purchases can be made.	1

sell_prices.csv		
Column Name	**Description**	**Example**
store_id	The Walmart store reference of where the product has been sold.	CA_1
item_id	The item reference of a specified product.	HOBBIES_1_001
wm_yr_wk	The week reference of which the day belongs.	11325
sell_price	The average price of a product in a store over a week, in dollars.	9.58

* Link for the dataset here: https://www.kaggle.com/c/m5-forecasting-accuracy/data

** More information on SNAP purchased can be found here: https://www.fns.usda.gov/snap/supplemental-nutrition-assistance-program

Bayesian Network Structure Learning Analysis

1.

TABLE I. SIX-PIECE INFORMATION GENERATED IN TERMINAL WINDOW OF NETBEANS UNDER STEP 4, EVALUATION

Evaluation
Nodes: 7
Sample size: 1913
TrueDAG arcs: 11
TrueDAG independencies: 10
LearnedDAG arcs: 7
LearnedDAG independencies: 14

2.

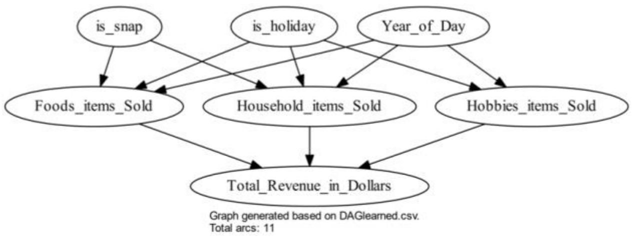


Fig. 1. Knowledge-based causal graph about created variables from our dataset to illustrate some meaninful causal links. Variables were created by performing the appropriate table aggregations and joins from the existing dataset.

Question 1. As a group, we used our knowledge of the dataset and common sense of the scenario. Since the sale happening on a SNAP purchase day (*is_snap)*, a holiday (*is_holiday*) or a certain year (*Year_of_Day)* cannot cause one another, we deduced that they share no direct link. Also, the number of sales for a category cannot cause a holiday but a holiday may influence the sales of an item (e.g. Christmas causes more sales of turkeys). The same applies to *Year_of_Day* and *is_snap* (except where SNAPs aren't allowed for Hobbies_items). The number of items sold clearly causes the total revenue.

3.

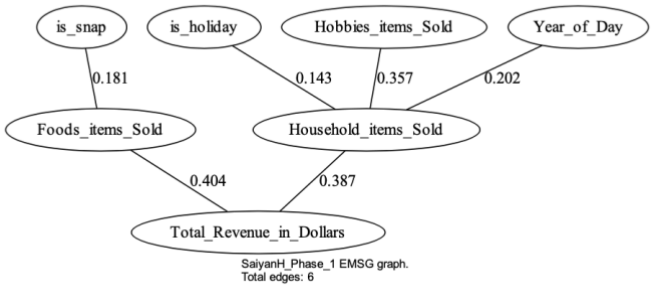


Fig. 2. SaiyanH phase 1 graph produced at Stage 4.

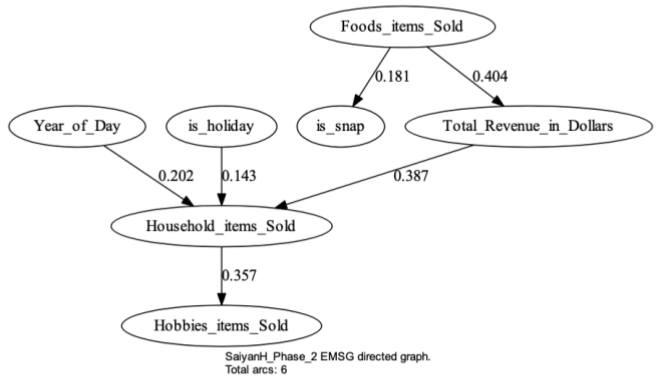


Fig. 3. SaiyanH phase 2 graph produced at Stage 4.

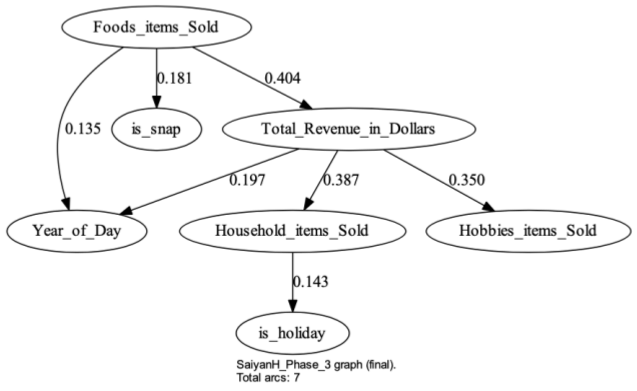


Fig. 4. SaiyanH phase 2 graph produced at Stage 4.

Question 2. The associational learning Phase_1 generates an undirected graph based on the MeanMax Marginal Discrepancy (MMD) scores of all edges. It takes 1 node, then checks all of its edges to other nodes and finally selects the edge that maximizes the average through the following formulae:

$$MMD_{MN}(A \rightarrow B) = \left(\sum_{j}^{s_A}\left[\left(\sum_{i}^{s_B}\left|P(B_i) - P(B_i|A_j)\right|\right)\Big/ s_B\right]\right)\Big/ s_A$$

$$MMD_{MN}(A \leftarrow B) = \left(\sum_{i}^{s_B}\left[\left(\sum_{j}^{s_A}\left|P(A_j) - P(A_j|B_i)\right|\right)\Big/ s_A\right]\right)\Big/ s_B$$

$$MMD_{MX}(A \rightarrow B) = \left(\sum_{j}^{s_A}\max_{i}\left|P(B_i) - P(B_i|A_j)\right|\right)\Big/ s_A$$

$$MMD_{MX}(A \leftarrow B) = \left(\sum_{i}^{s_B}\max_{j}\left|P(A_j) - P(A_j|B_i)\right|\right)\Big/ s_B$$

However, the constraint-based learning Phase_2 produces a directed graph from the undirected graph by SaiyanH classifying each node triple into conditional dependence, independence or insignificance based on the MMD scores in Phase_1, taking the form of common-cause or common-effect. Any undirected edge is reassessed with BIC scoring and do-calculus tests.

Question 3. By utilizing the score-based learning technique BIC, Phase_3 takes the directed graph from Phase_2 and analyses neighbouring graphs by adding, removing or reversing nodes based on the Hill-Climbing and Tabu search methods. The search stops if all Tabu neighbouring graphs are searched with no improvement to the score or when the number of escapes exceeds $V(V-1)$. It then returns the graph which maximizes the scoring function. These graphs must be acyclic and cannot have multiple independent subgraphs with the search space restricted by conditional independence. If both graphs are identical, then the graph in Phase_2 already meets this criterion.

4.

TABLE II. FOUR CSV FILES GENERATED IN STEP 4

CSV Files
marginalDep.csv
conditionalDep.csv
conditionalIndep.csv
conditionalInsignificance.csv

Question 4.

TABLE III. NUMBER OF SCORES GENERATED IN EACH CSV FILE

File	No. of scores
marginalDep.csv	21
conditionalDep.csv	7
conditionalndep.csv	12
conditionalInsignificance.csv	86

The marginalDep.csv table has 21 score rows since we have 7 variables with $7C2$ pairings. In the Phase_2, conditional independency tests on pairs of nodes conditioned on the remaining nodes are performed. There are $7C2$ pairs with 5 remaining nodes to condition on, therefore leaving us with $5 \times 7C2 = 105$ triples to performs tests on. In Phase_2, we're using a conservative approach [2] to measure either conditional dependency, independency or insignificance and so fewer triples will be classed as conditional dependent or independent because of the strict dependency condition. This means most of triples will be classified as conditionally insignificant.

5.

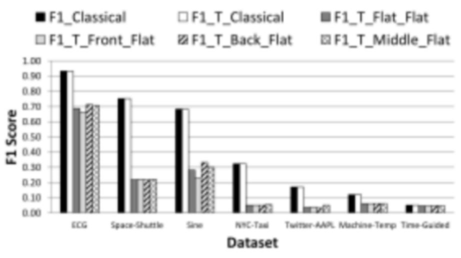
TABLE IV. STATS FROM METRICS, SCORING FUNCTIONS, INFERENCE-BASED EVALUATION AND STRUCTURE LEARNING

The Scores are:
Precision score: 0.500
Recall score: 0.318
F1 score: 0.389
SHD score: 8.500
DDM score: -0.455
BSF score: 0.218
BIC/MDL score -15230.174
# of free parameters 250
Structure learning elapsed time: 1 seconds total
(Phase 1 = 0 secs, Phase 2 = 1 secs).

Question 5. Seen from Table V, all scores we obtained are lower. Potentially, our model didn't discover enough "true" arcs than total arcs, generating a low F1-Score. We expected this since Tatbul *et al.* explain how real anomalies, which are range-based, are rarely captured [3].

TABLE V. OUR SCORES VS AVERAGE SCORES OF THE CASE STUDIES [2]

F1-Score		SHD-Score		BSF-Score	
Our Result	**Case Studies Average**	**Our Result**	**Case Studies Average**	**Our Result**	**Case Studies Average**
0.389	0.555	8.5	71.25	0.218	0.481



(c) F_1-Score

Fig. 5. Model F1-Score in [3] vs classical point based model

Our model performs better in SHD-Score, meaning it requires fewer edge insertions, deletions and arc reversals to transform the learned graph into the true graph. We expected a BSF-Score lower than average due to the time-series nature of our dataset. Our value sits just above the "semi ignorant" and below the "accurate" ranges [1].

Question 6. The time complexity of SaiyanH increases rapidly with the number of nodes and sample size. The time-complexity we obtained is consistent with the ones in [2] since we have only 7 nodes in our graph and the sample size of 1,193 is relatively low. As seen in Table 2 of [2], our model is similar to the models with 8 nodes, 8 true edges and 18 free parameters with our sample size lying between 1k and 10k. It was not surprising to see that our model achieved a runtime of 1 second being the same as the runtime we obtained.

Question 7. In Step 3, the model simply generates the BIC-Score obtained from our graph. In Step 4, the model creates the graph itself and was able to modify the graph by rearranging the edges such that it maximizes the BIC-Score. Initially, we expected to obtain better scores in Step 4 since the graph we generated was through intuition and we were aware it might not be a "true" reflection of our dataset. The graph produced in Step 4 represents all the dependencies between variables and so we expected a more accurate representation.

Question 8. In Step 3, the model requires more free parameters to represent the knowledge of our dataset, because the graph is perhaps not an accurate representation of the data, therefore requiring more free parameters to model our data. In Step 4, the graph produced has the maximized BIC-Score. This means that the graph is a better representation of the "true" knowledge and hence requires fewer free parameters to represent the knowledge behind our dataset. This is in agreement with our initial expectations since Step 4 should improve our model built based on our subjective knowledge alone in Step 3.

REFERENCES

[1] A.Constantinou, "Learning Bayesian Networks that enable full propagation of evidence", 2020.
[2] A.Constantinou, "Evaluating structure learning algorithms with a balanced scoring function."
[3] N.Tatbul *et al.*, "Precision and Recall for Time Series" Jan.2019.

APPENDIX

A. Datasets, Graphs and Results

We used Bayesys JAVA Netbeans to do Bayesian analysis on our dataset. The knowledge-based graph, SaiyanH phase graphs, csv files, text outputs and the dataset we compiled for the analysis can be found on our GitHub repository:

https://github.com/GerardoMoreno96/ECS784P_DataAnalyticsProject