94867 Decision Analytics for Business and Policy

Wikipedia Web Traffic Prediction - Project Report

Team Member: *Shuting(Tina) Feng, Yunxin(Silvia) Gu, Yingzhe(Dylan) Jin, Zouyiyun(Chris) Peng*

Additional packages to be installed

(pip install command available on corresponding colab notebooks)

- `tsfresh`
- `shap`
- `pystan~=2.14`
- `fbprophet`

References

- https://en.wikipedia.org/wiki/Wikipedia
- https://neptune.ai/blog/select-model-for-time-series-prediction-task
- https://tsfresh.com/
- https://tsfresh.readthedocs.io/en/latest/text/feature_filtering.html
- https://tsfresh.readthedocs.io/en/latest/text/forecasting.html
- https://towardsdatascience.com/tutorial-on-lstm-a-computational-perspective-f3417442c2cd#:~:text=LSTM%20equations,-The%20figure%20below&text=The%20LSTM%20has%20an%20input,the%20next%20time%20step%20LSTM.
- https://www.tandfonline.com/doi/full/10.1080/00031305.2017.1380080?scroll=top&needAccess=true&role=tab

Contribution

Member code name: C - Chris Peng, D - Dylan Jin, T- Tina Feng, S - Slivia Gu

Task	**Members**	**Deliverable**
Propose potential topic area	C, D, T, S	Project Proposal
Preliminary data collection on specific topics	C, D, T, S	
Project proposal document	C, D, T, S	
Data cleaning	C	Project Data and Model Plan
Exploratory Data Analysis	C	Modeling
Modeling: Traditional Supervised Learning	T	
Modeling: LSTM	S	
Modeling: Prophet	D	
Model Comparison	D	
Presentation sides	C, D, T, S	Project Presentation
Presentation	C, D, T, S	
Project report document	C, D, T, S	Project Report

I. Problem Statement

1. Question

In the linear programming lecture, we were introduced to the google ads example. In the example, the estimated requests of each page were given to build up the linear optimization model. We are inspired by this interesting example and hope to dig into how the estimated visits can be calculated in real cases.

Wikipedia is a multilingual free online encyclopedia written and maintained by a community of volunteers through open collaboration and a wiki-based editing system, which has huge daily traffic. We would like to predict the future daily views, as known as web traffic, of Wikipedia articles based on historical daily views with different access, such as desktop and mobile web. The goal is to achieve as accurate predictions as possible, which can then offer relevant business insights like strategies to increase conversion rate and attract more donations.

2. Benefits

For most online businesses, web traffic can directly relate to conversion numbers, which are vital to business growth. If the web traffic can be predicted and forecasted, they can easily prepare for future fluctuation and adjust their strategy accordingly. For example, if the website engineers can be informed that there will be a peak visit in the following week, they can strengthen the endurance of the websites and thus avoid collapsing. Also, since Wikipedia is a free, open-source platform that mainly relies on individual donations, predicted web traffic can be used as a reference to help the staff decide where (usually articles with high web traffic) to place donation requests and thus maximize potential revenue.

3. Challenges

The prediction of the web traffic is challenging because 1) analytically, the interaction between users and the Internet is chaos since we cannot simulate every action that a user takes for every second; 2) technically, site performance data isn't always accurate, such as missing tracking code on web pages, data sampling with skewed results, and ad block services that filter out valuable ad-related data; 3) from a policy perspective, moral concerns of tracking user's browsing behavior may occur.

Overall, we need to find a method that can process large-scale and time-sensitive data and output useful insights and predictions for future usage.

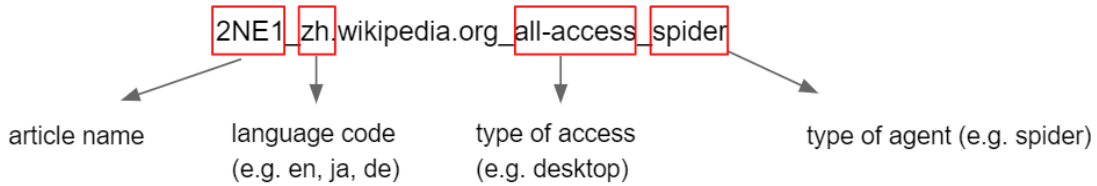
II. Data Summary

1. Source

The question we are interested in is to predict the future daily views, as known as web traffic, of certain websites based on historical daily views with different access, such as desktop and mobile web. With this goal, the main data source we will use is a Kaggle challenge dataset: Web Traffic Time Series Forecasting (https://www.kaggle.com/competitions/web-traffic-time-series-forecasting/data), with which we will forecast future web traffic for Wikipedia articles.

2. Description

We use train_2.csv, which includes web traffic data for various Wikipedia pages during 2015 and 2017, for our project. In this dataset, each row corresponds to a particular article and each column corresponds to a particular date. And some entries are missing data. The page names contain the Wikipedia project in different languages (e.g. en.wikipedia.org), types of access (e.g. desktop), and types of agents (e.g. spider). Specifically, each article name has the following format: 'name_project_access_agent' (e.g. 2NE1_zh.wikipedia.org_all-access_spider').

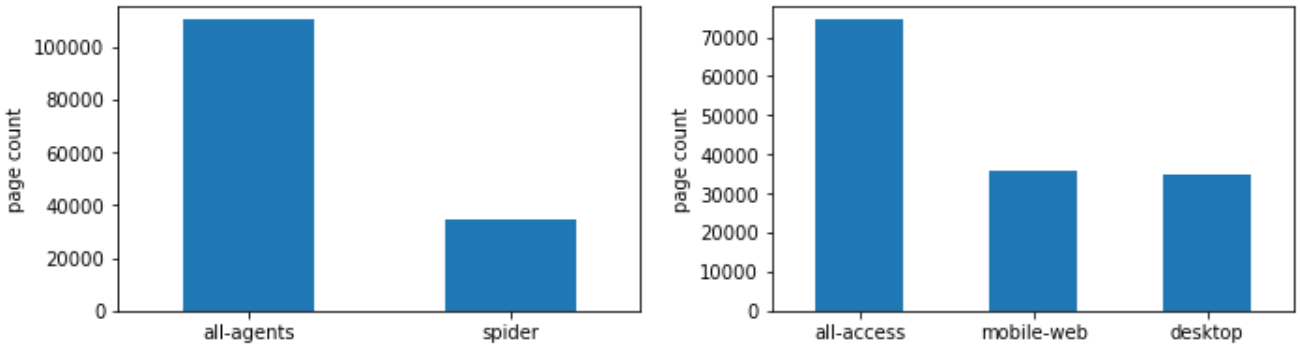


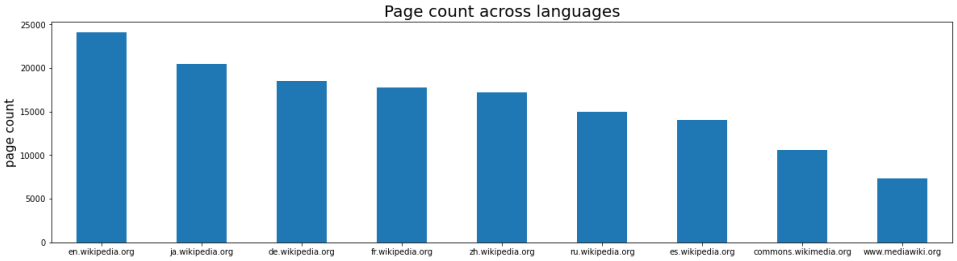
This dataset consists of approximately 145k time series. Each of these time series represents a number of daily views of a different Wikipedia article, starting from July, 1st, 2015 up until September 10th, 2017.

Unfortunately, the data source for this dataset does not distinguish between traffic values of zero and missing values. A missing value may mean the traffic was zero or that the data is not available for that day, which we will address in the data cleaning process (refer to Fig 1.1 in the Appendix for the raw dataset overview).

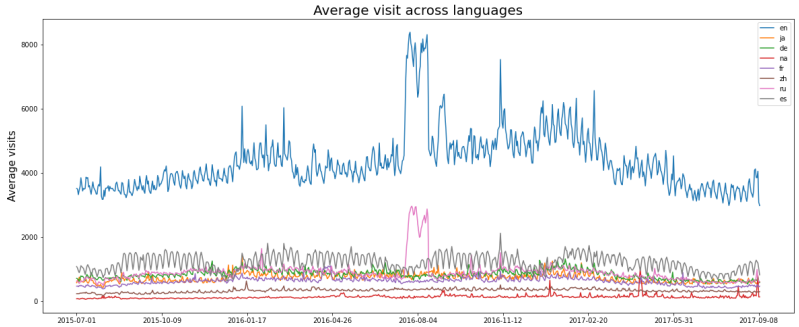
3. Exploratory Data Analysis

For page counts, the number of pages with spider agents is almost 3 times fewer than that of pages with all agents, while the number of pages with all access is almost 2 times more than that of pages with other access. And among all the language parameters, the number of English pages is the most.



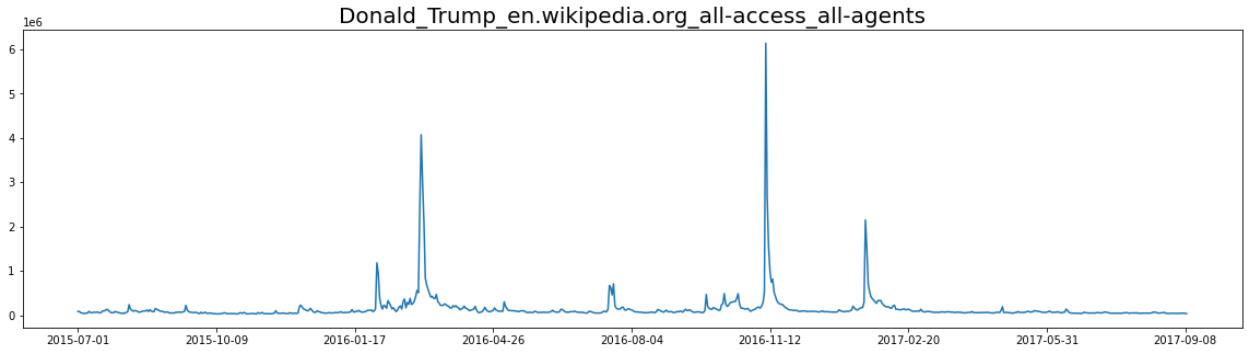


The daily average visits across languages are as below. English page shows a much higher average view, as might be expected since Wikipedia is a US-based site. The English and Russian plots show very large peaks around 2016 August.

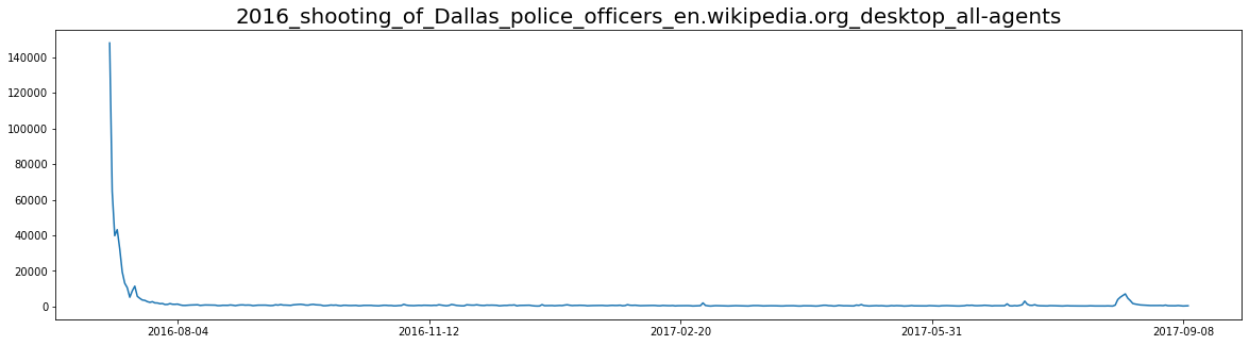


For individual pages, the data is also not smooth. There are sudden gigantic spikes, which clearly suggest the

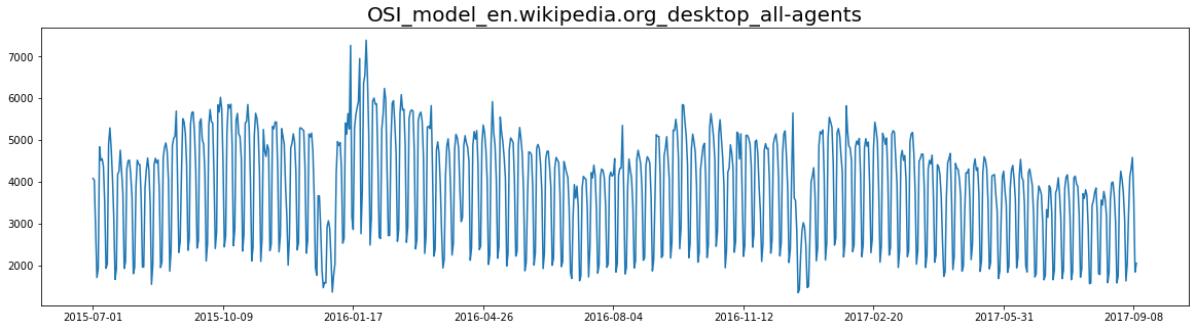
effects of current events on Wikipedia views. For example, the page for Donald Trump received a large number of viewers around the beginning and the end of 2016, which is obviously affected by the US election as shown below.

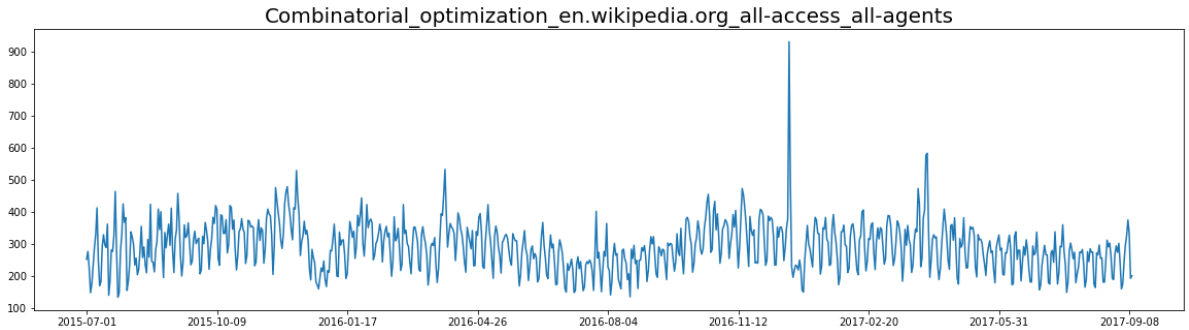


The 2016 shooting in Dallas occurred, then a Wikipedia page was quickly constructed and received a huge number of views in a short time. The number of views mostly decayed away in 2 weeks.



As the examples above show, politics and news-related articles usually have huge ups and drops in the number of visits, which is hard to capture and predict. However, visits in some articles are more steady. For example, as in the figures below, visits in the OSI model and combinatorial optimizations are more ideal to help us to train the model.





III. Analytical Formulations

1. Data Cleaning

According to the data summary above, visits to event-based articles are volatile and hard to capture. It is necessary to clean the data and the cleaning process is conducted in the following order.

1) **Select English pages**

   Since English pages are more explainable and have larger instances, we selected pages with "en" language code, which denotes English pages.

2) **Delete pages with Spider access**

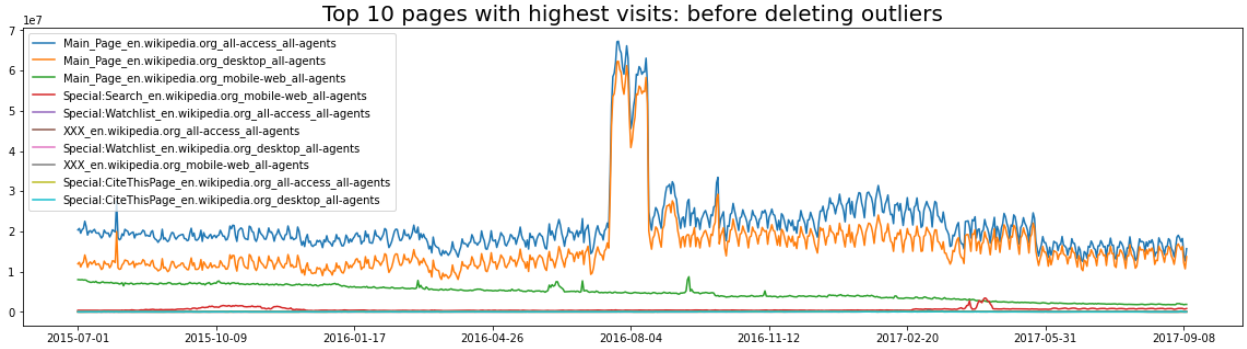
   Considering the objective of this project, pages with only Spider access are not meeting the expectation, since visits to these pages can not represent views on them. Thus, the pages with only Spider access should be excluded from our training dataset.

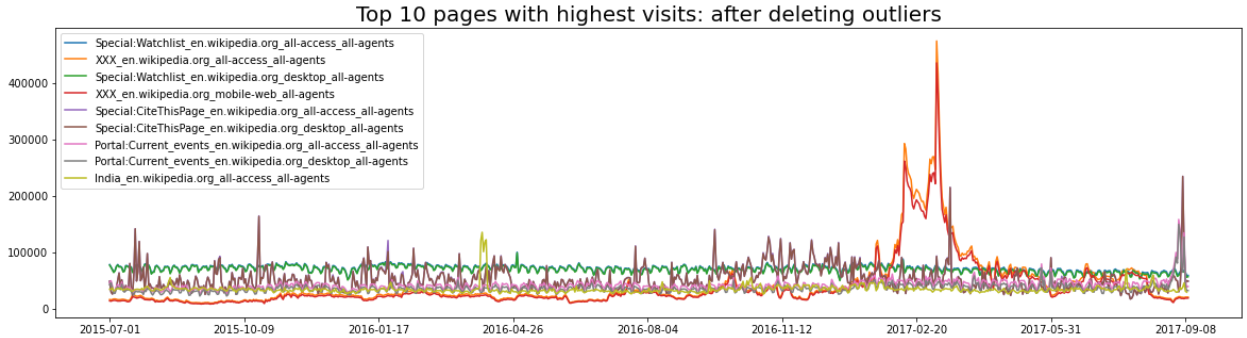
3) **Delete pages with extreme peaks**

   As in the data summary part, some individual pages have gigantic spikes because of events or emergencies, instead of regular trends. To better understand and capture the trends, we delete pages with extreme peaks by calculating the percentage changes of the historical visits. We only keep pages with relatively steady trends that do not have a percentage change greater than 300% in a day and a percentage change greater than 500% in a week.

4) **Delete outliers**

   After the steps above, the dataset contains 1528 pages. After sorted by visits in the recent 365 days from high to low, the "Main_Page" and "Special:Search" pages have extreme high visits compared to other pages. We count these pages as outliers and exclude them from the training dataset.





5) **Select the top 1200 pages with the highest average views**

   As mentioned in the data summary, the data source for this dataset does not distinguish between traffic values of zero and missing values. Pages with few visits are not beneficial to our training process, since their historical visits may contain multiple zeros. To address this problem, we select the top 1200 pages with the highest average views to reduce the confusion between traffic values of zero and missing values.

6) **Select the top 1000 pages with the lowest standard deviation**

To further decrease control abnormal fluctuations and the training dataset size, we sort the datasets by the standard deviation in historical views from lowest to highest. We select the top 1000 pages with the lowest standard deviation (refer to Fig 1.2 for the cleaned dataset).

2. Traditional Supervised Learning

2.1 Methodology

Supervised machine learning models are often the go-to methods for many prediction tasks, and their versatility also gives them the ability to handle time series forecasting. However, different from a standard supervised machine learning task, which is built upon the assumption that each observation is independent, time series data can suffer from the issue of lacking independent variables. For example, given one time series of one website, within which every single entry is the number of visits to that page on a certain day, these entries especially those that are close or next to each other (e.g. visit on day 1 and visit on day 2) can be highly correlated -- in other words, there are often no independent variables in time series data. Also, each entry can be a potential target since the number of visits is exactly what we hope to predict.

To solve this problem and thus fit the time series forecasting into the framework of traditional supervised learning, we need to generate independent variables (features). To do so, we can extract information from the time series such as seasonality, average traffic of the last 7 days, etc. as independent variables to help predict visits (traffic on a chosen day).

While we can manually extract the above features by analyzing and preprocessing the data, luckily, there is a package called “tsfresh”, which helps in extracting features from time series data such as mean, max, min, median, 0.4 quantile, 0.7 quantile, linear trend attribute intercept, etc. Once we extract these features, the problem is converted to a machine learning problem instead of a time series one, and we can apply the traditional ML models we are familiar with.

It’s noteworthy that using this method, we can predict web traffic on a certain day for websites in bulk (in our case, 1000 websites) since after conversion, every row is an independent observation with extracted features and one target traffic.

2.2 Time Series Feature Extraction

As mentioned above, instead of manually extracting features such as day, week, and year, we use the tsfresh package to automatically extract relevant time series features for each page. The key to this step is to reformat the original data frame into the table format as shown in Fig 2.1, and after using tsfresh’s extract_features() and select_features() functions, we can get a feature data frame referred to in Fig 2.2, where each row is still an independent observation. Some resulting features include view_median, view_mean, view_autocorrelation__lag_5, etc, which can be referred to in Fig 2.3. However, despite the convenience of feature extraction via tsfresh, which helps add complexity to model training (due to the high dimensionality of features generated), we may still need to manually add features that imply weekly or monthly patterns as they were not captured by tsfresh functions and thus lowering the precision of predicted results. We will discuss this more in detail in the outcome section.

2.3 Modeling & Evaluation

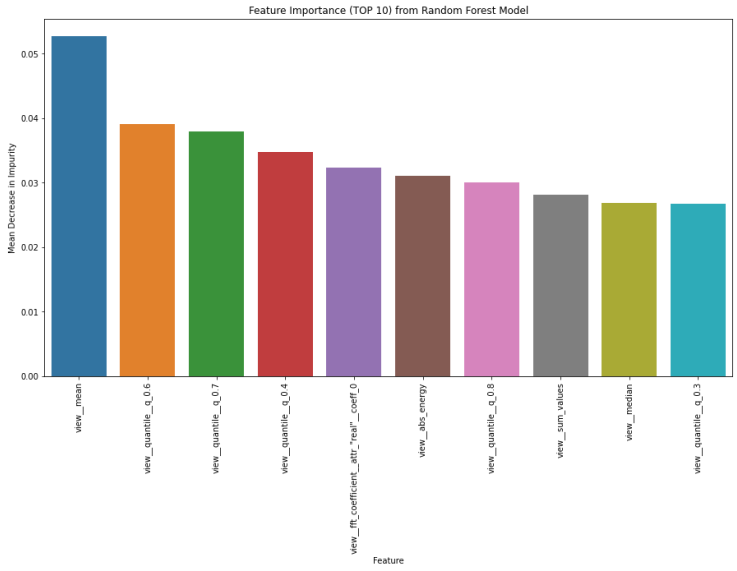
Out of 803 days of data from 7/1/2015 to 9/10/2017, we would use data FOR the first 73 days starting from 12/17/2016 (since there exist many pages with no data before this date and we don’t want to rely on imputed data only to train the model) in all 1000 pages for model training, in which the first 72 days are used to predict visits on the 73rd day.

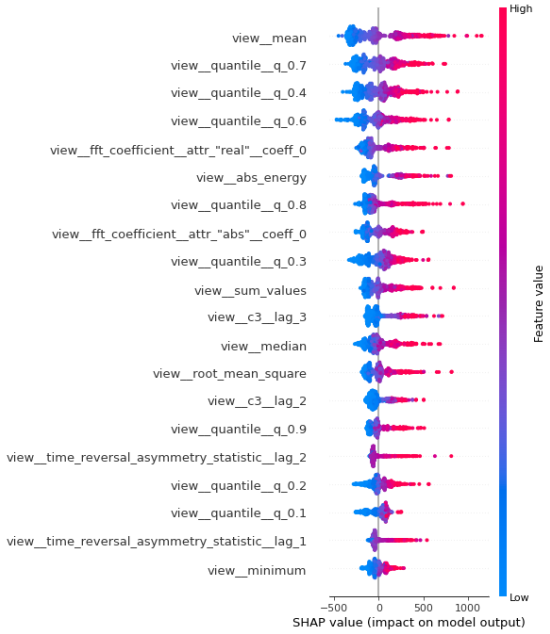
We would compare four models including ridge regression, decision tree, random forest, and extreme gradient boosting (XG Boost). We choose Ridge Regression to increase the model complexity on top of simple linear regression as we may first assume a linear relationship for the time series; plus we'd also like to keep all features since feature selection/filtering is already done during feature extraction using tsfresh. We choose Decision Tree because it is easy to interpret, understand, and visualize. The reason to choose Random Forest is that it is a robust model that allows fitting nonlinear relationship and is also very easy to use. And lastly, we also pick XG Boosting as it enables a more complex ensemble.

According to the evaluation metrics – RMSE and R-squared – from the four models trained, we choose random forest as our best model given that it yields the best train RMSE-test RMSE pair and train R2-test R2 pair.

	Train RMSE	Test RMSE	Train R-Squared	Test R-Squared
Ridge Regression	1267.97	2626.85	0.93	0.48
Decision Tree	1770.94	1741.20	0.85	0.77
Random Forest	895.07	1579.51	0.96	0.81
XGBoost	787.26	1831.82	0.97	0.75

Based on the TOP 10 feature importance plot from Random Forest on the left, we found that the view_mean and quantile-related features contribute the most to the predicted results. And the Shap Plot of the same model on the right also displays similar information, and on average, most features have a positive impact on the predicted result as shown by the color scale (higher feature values are on the spectrum of positive impact). Stakeholders can also focus on one specific observation (webpage) of their interest to see the specific numerical contribution of every feature (referred to in Fig 2.4, here only the TOP 10 are shown) to the predicted visit, allowing more interpretability.





Overall, despite the complexity resulting from the 300+ extracted features, the supervised machine learning model, specifically Random Forest, can still be interpretable in this context, especially if we can get clear of the meaning behind every single extracted feature.

Note that for model training, ideally, we would have trained one model for each sliding window, so that we could exploit all available data for model training, but due to the time constraint from feature extraction via tsfresh for all pages in every single window, we only used sliding window for the purpose of testing. In other words, we would

use all future dates, which are not involved in the model training, as unseen data to test the best model via a total of 123 sliding windows.

2.4 Outcome of Best Model

After applying the best model to all 123 sliding windows for further testing, we are able to achieve an average RMSE of 41.19 and average R-squared of 0.79. We can then aggregate all predicted results to each of the 1000 pages and compare the predicted time series with the actual one. Below we picked 6 pages that can help illustrate the performance of the best machine learning model (random forest). Based on all 6 plots shown in Table 2.1, predictions from Random Forest are able to catch the overall trend of the time series when the trend is relatively more smooth. However, it seems difficult for the model to capture some extreme peak values as shown in the three plots in the first row (refer to Table 2.1). Also, according to the other three plots in the second row (refer to Table 2.1), the model fails to capture repeated patterns like weekly or monthly ones. This may result from the fact that we rely too heavily on the tsfresh feature extraction function, which turns out not to extract these features. This reminds us that despite the easy automation of high dimensional time series features, it would be still worthwhile to manually extract features, such as the day of the week, after detailed observations.

3. LSTM

3.1 Algorithmn of LSTM

LSTM stands for long short-term memory networks, used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems.

LSTM employs various gates to decide what information to keep or discard. Also, it adds a cell state, which is like a long-term memory of LSTM.

- **Hidden state & new inputs** — hidden state from a previous timestep (h_t-1) and the input at a current timestep (x_t) are combined before passing copies of it through various gates.
- **Forget gate** — this gate controls what information should be forgotten. Since the sigmoid function ranges between 0 and 1, it sets which values in the cell state should be discarded (multiplied by 0), remembered (multiplied by 1), or partially remembered (multiplied by some value between 0 and 1).
- **Input gate** helps to identify important elements that need to be added to the cell state. Note that the results of the input gate get multiplied by the cell state candidate, with only the information deemed important by the input gate being added to the cell state.
- Update cell state —first, the previous cell state (c_t-1) gets multiplied by the results of the forget gate. Then we add new information from [input gate × cell state candidate] to get the latest cell state (c_t).
- Update hidden state — the last part is to update the hidden state. The latest cell state (c_t) is passed through the tanh activation function and multiplied by the results of the **output gate**.

Equation for a single time stamp:

$$f_t = \sigma_g\,(W_f \times\ x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\,(W_i \times\ x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\,(W_o \times\ x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\,(W_c \times\ x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t)$$

3.2 Neural Network Architecture

In We used the 60-day rolling window to predict page views on every 61 days according to historical data of the previous 60 days consecutively.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 60, 100)           40800

 lstm_1 (LSTM)                (None, 100)               80400

 dense (Dense)                (None, 25)                2525

 dense_1 (Dense)              (None, 1)                 26

=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
_________________________________________________________________
```

3.3 Outcome of LSTM

Out of the 1000 pages, we selected 5 representative instances with different trends and patterns to test the model.

	Page 0	Page 1	Page 2	Page 3	Page 4
R-squared	0.491	-0.2073	0.5157	-0.4613	0.8422
RMSE(normalized feature)	0.048	0.0524	0.0423	0.0426	0.0234

4. Prophet

Prophet is an open-source forecasting tool developed by Facebook. It is based on the idea that a time series can be modeled as a combination of patterns, trends, and seasonality, plus some random noise. Prophet uses a decomposable time series model with three main components: a piecewise linear or logistic trend, a yearly seasonal component modeled using Fourier series, and a weekly seasonal component using dummy variables. Prophet is designed to be intuitive to use, and it can handle missing data, large outliers, and multiple seasons.

According to the description on the official website, we just use the csv file to store two columns. The name of the first column is 'ds', and the name of the second column is 'y'. The first column represents the timestamp of the time series, and the second column represents the value of the time series. Through the calculation of the Prophet, yhat, yhat_lower, and yhat_upper can be calculated, which respectively represent the predicted value of the time series, the lower bound of the predicted value, and the upper bound of the predicted value. The two tables are represented by the two figures below.

	ds	y
77376	2015-07-01	3004.0
222439	2015-07-02	2769.0
367502	2015-07-03	2876.0
512565	2015-07-04	2829.0
657628	2015-07-05	2964.0

4.1 Algorithm of Prophet

In the field of time series analysis, there is a common analysis method called Decomposition of Time Series, which divides the time series into several parts, namely seasonal item s, trend item g, and residual item ε. In the Prophet algorithm, the author also takes holidays h into consideration, that is to say, for all t>0, there is

$$y = g(t) + s(t) + h(t) + \varepsilon$$

Among them, g(t) represents the trend item, which represents the changing trend of the time series on the non-period; s(t) represents the seasonality item, or called the seasonal item, generally speaking, it is in the unit of week or year; h(t) represents the holiday item, indicating whether it exists on the day Holidays; ε indicates the error item or called the remaining item. The Prophet algorithm is to obtain the predicted value of the time series by fitting these items and finally adding them up.

4.2 Outcome of Prophet

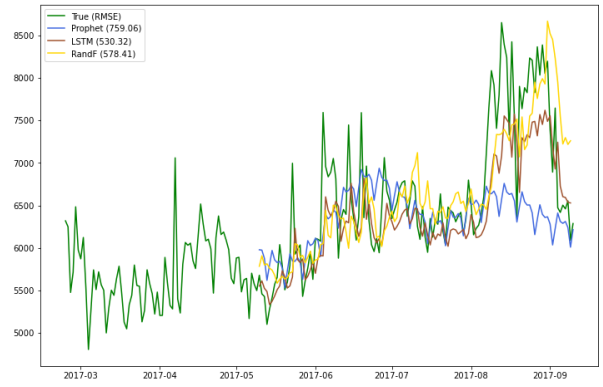
Since implementing Prophet is easy, we only showcase the outcome of prediction in this section. In brief, we select the last 30 days of data as test data and all of the former data as training data that are fed into the model to generate the prediction. Using the same 5 instances as in LSTM, the predicted results for all of the 5 selected pages are shown in the appendix, and we can find it can predict not only the long-term trend but also the weekly, and monthly seasonality very well.

	Page 0	**Page 1**	**Page 2**	**Page 3**	**Page 4**
RMSE	759.06	603.14	295.30	501.87	457.09

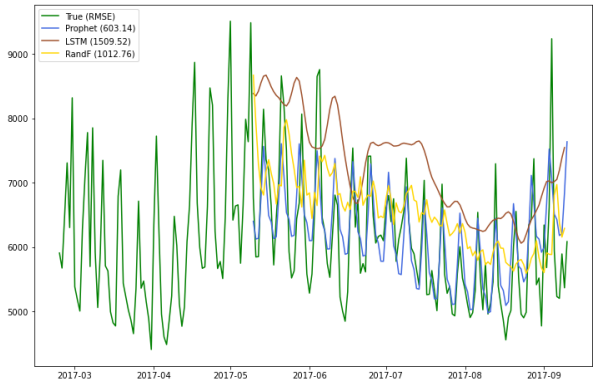
IV. Conclusion & Analysis

Overall, these 5 pages gave us a great baseline to compare different models in different kinds of distribution of a dataset. For example, LSTM and Random Forest can easily predict the overall trend and reach a relatively small RMSE. As we can see, Prophet sometimes failed to predict the rocket increase in a specific period. But what Prophet can do better is the seasonality: it did surprisingly great in catching these kinds of weekly/biweekly/or monthly seasonality.

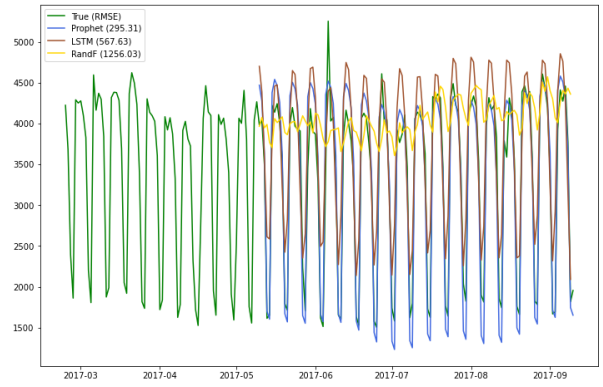
Comparison for Page 1:



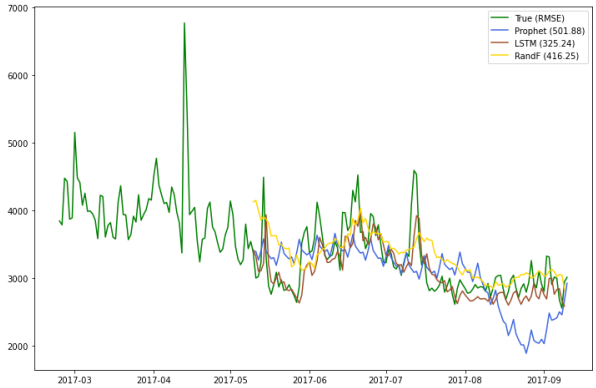
Comparison for Page 2:



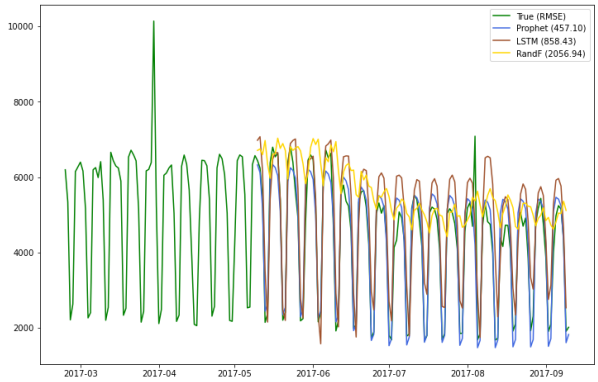
Comparison for Page 3:



Comparison for Page 4:



Comparison for Page 5:



As an overall comparison, Random Forest is a non-parametric method, which means that it does not make any assumptions about the underlying data distribution. This makes it a flexible and robust method that can handle a wide variety of data types. However, it can also be computationally expensive, especially for large datasets in time series data.

LSTM is a powerful tool for modeling complex, non-linear time series relationships in data, and it can handle a wide variety of data types. However, it requires a large amount of training data and tuning and can be computationally intensive.

Prophet is designed to be intuitive to use and can handle missing data, large outliers, and multiple seasons. However, it is limited to univariate time series data and may not perform well for data with complex patterns.

V. Impact and Future Discussion

The analysis of our project can have various impacts on different stakeholders. It can help attract donations for non-profit knowledge communities (advertising on high-page view pages). It can provide insights to decision-makers like non-profit organizations and the government. It can also guide creators to optimize content quality based on page view change and help developers to select similar pages to conduct AB testing.

For future improvement, we could do more feature engineering on the page view data to generate some non-time-related features like the language of pages. We could also do some neural architecture searches to try different NN models for different pages. And it's also worthwhile to try traditional time series methods like ARIMA and GARCH.

VI. Appendix

1. Code and Instruction for implementation details

1. Save the shared folder "94867 DABP Project" into your own drive
    a. You can find all colab notebooks in the "code" folder and all datasets in the "data" folder
2. Run all four notebooks in the following order:
    a. 1-web-traffic-eda.ipynb
    b. 2-web-traffic-ml.ipynb
    c. 3-web-traffic-lstm.ipynb
    d. 4-web-traffic-prophet&comparison.ipynb (must be run at the end)

    ***For each notebook, change the data reading path into the path of your own drive
3. After changing the paths, run all codes to generate plots and results.

2. Plots