The Future of Cycling in Seattle:   
A Data Science Approach to Ridership Forecasting

Group 2 Team Members:

Justus Ngunjiri, Arthur Setiawan, Yen-Ting Chen, Robert Barrentine

DATA 5100 | Dr. Fischer | FQ22

8 December 2022

Problem Statement

Cycling is a big deal in Seattle. According to a 2019 study by Nielsen for the Seattle Times, “an estimated 169,000 adults in the Seattle area use a bicycle for transportation.” (Balk, 2019) That number is expected to grow considerably in the coming years as Seattle continues to attract transplants to fill new jobs. The Seattle Department of Transportation (SDOT) is betting on bicycles to adapt for a denser Seattle. Since 2014, SDOT has been publishing regular updates on their Seattle Bicycle Master Plan (BMP) which aims to accommodate growth and associated mobility needs with investments in bicycle infrastructure. (BMP, 2021) In order to assess the impact of the BMP initiatives, SDOT has installed 12 bike counters throughout the Seattle area on trails and roadways. (Bike counters, n.d.) This data is made publicly available at data.seattle.gov and is granular to the hour.

The proposal of this project is to utilize data to answer a series of questions related to the Seattle cyclist population. As the Seattle cyclist population is expected to grow over the next several years, is it possible to accurately predict the growth using statistical and machine learning models? The seasonality of ridership throughout the year is an interesting aspect to this dataset. Is it also possible to accurately model seasonal swings in a dataset? Also, how has the pandemic affected the past few years’ data trends and does that change the short-to-medium term projections?

All code used for data analysis and modeling is hosted in our GitHub repository at this [link](https://github.com/arthursetiawan/DATA_5100_Project). Throughout this paper, links to individual code pages will also be linked for ease of navigation.

Data Source

This project will utilize data extracted from the Seattle Department of Transportation (SDOT) database. SDOT’s database is online with an open access policy across various transportation data categories. They include, parking, traffic flow maps, planning and maintenance, and transit and commuting. Through SDOT, most of these datasets, depending on the form (historical or live), can also be accessed through the city’s integrated geodata host site. This host is called City of Seattle GIS. The datasets are collected through the city’s geographical information system programs where the data is captured live and documented for future reference. One such program is the maintenance of Bicycle and Pedestrian counts across three main bridges namely: Fremont Bridge, Spokane Street and 2nd Avenue. See Figure 1 below of the Fremont Bridge. Depending on the location and development interests, some of the biker and pedestrian counter sites capture either cyclist count data, or a combination of cyclist and pedestrian count data.

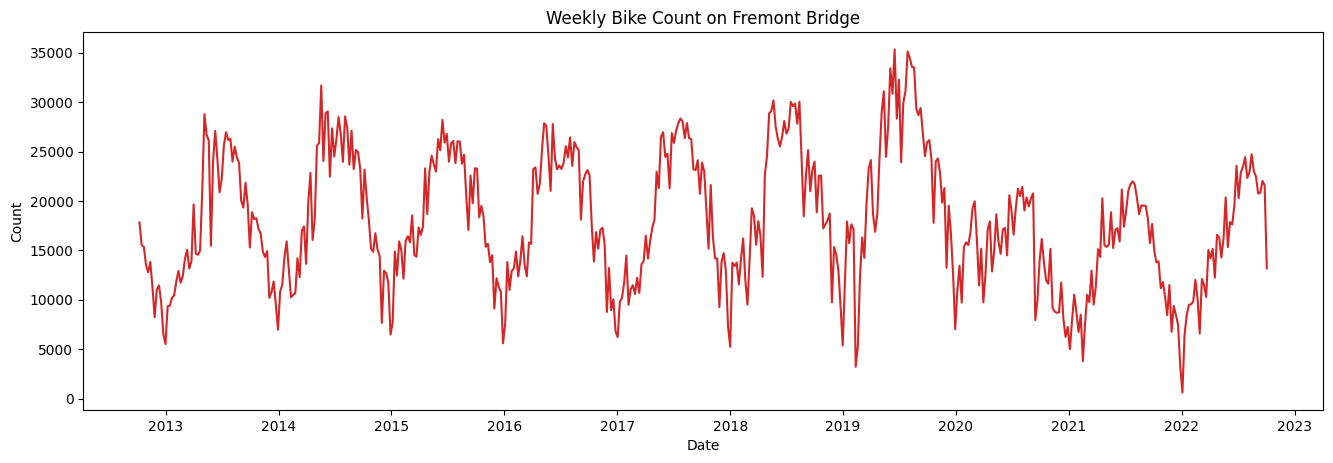


*Figure 1.* Photo of Fremont Bridge taken at sunset from Aurora Bridge. *Photo credit: Chris Papetti, 7 September 2022 at 7:41PM*

Given that our interest is the Fremont Bridge, we will navigate to the database and extract the Fremont Bridge cyclist data. This program maintains data for cyclist traffic only, thus filtering to exclude pedestrian data will not be necessary. (Seattle government, 2022) The program was started in late 2012, thus there are entries from the same year to the current live date.

When we are doing any data pre-processing, there are 4 steps to be concerned with. First of all, we’d have to drop off any columns and keep the columns that we are interested in. Then, we need to convert the date-time index using the pandas function. Afterwards, using the quadratic method we interpolate the estimates into the missing values. Finally, the summarization on the time frame is done to conclude the resampling with a different time window. With the above 4 steps, we are capable of completing the data pre-processing.

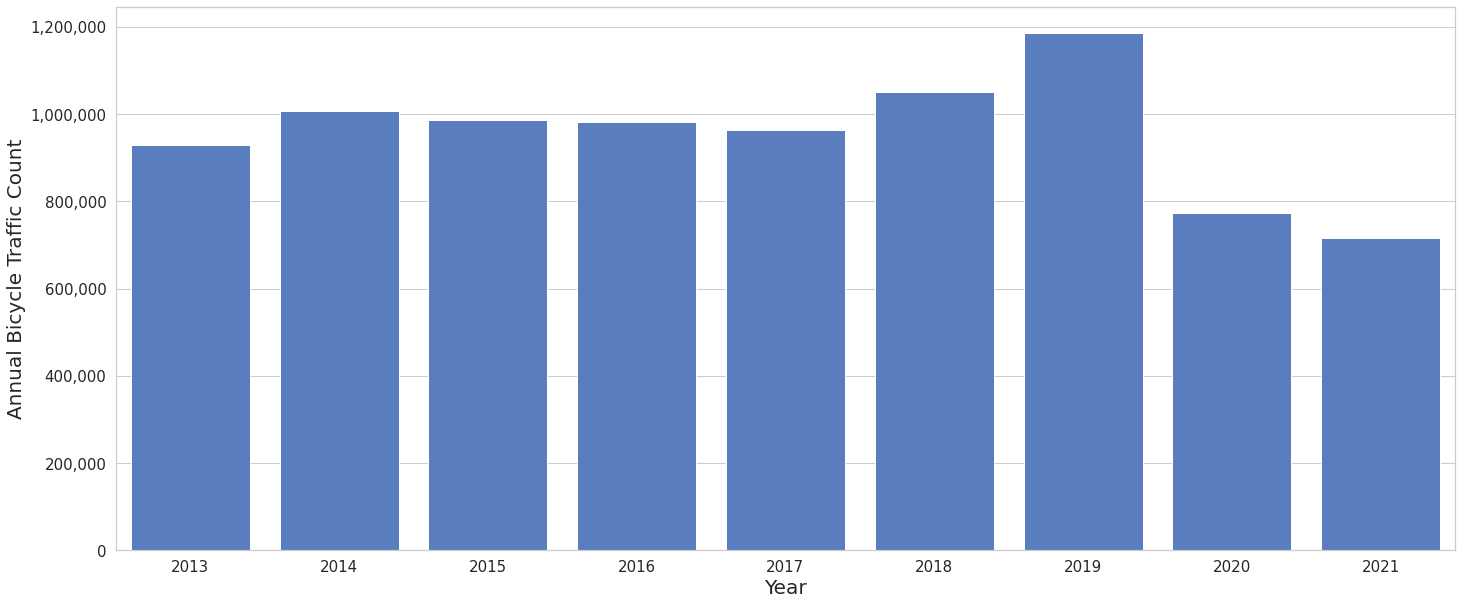
Depending on our desired prediction intervals, we may aggregate the data to specific periods; for example, data may be summarized on an hourly, daily, weekly, monthly, quarterly, or annual basis. See Figure 2, below, where the data was summarized on a weekly basis and plotted for all available data. The code used to generate this time series plot is located [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/8931343c6546ad13f01242277a62a9c3c801b942/time_series_plots.py).



*Figure 2*. Weekly cyclist count for the Seattle Fremont Bridge

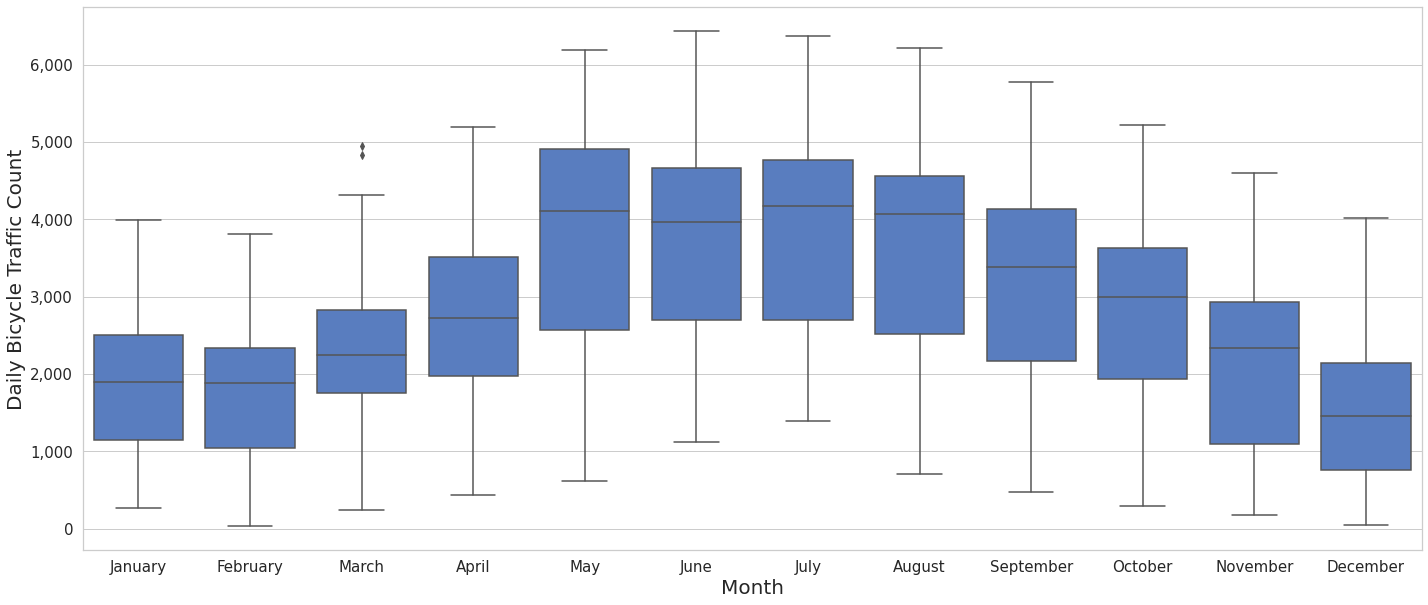
Exploratory Data Analysis

One of the first summaries we can perform on the time-series data is to compare the total cyclists counted in each year (Figure 3). The code used to create this chart may be found on our GitHub repository [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/8931343c6546ad13f01242277a62a9c3c801b942/time_series_plots.py). Because the data includes only partial data for 2012 and 2022, these years will be excluded. Ridership growth appears to be somewhat stagnant in the first several years (2013 to 2017) hovering between 900K and 1M total riders per year. However, in the years 2018 and 2019, the annual ridership is observed to increase by nearly 100K year-on-year reaching a maximum annual ridership in 2019 of just under 1.2M riders. The last two years (2020 and 2021) show a major decrease in ridership, falling to between 700K and 800K annual cyclists. The rapid decrease in ridership starting in 2020 is most likely attributable to the onset of the global COVID-19 pandemic.



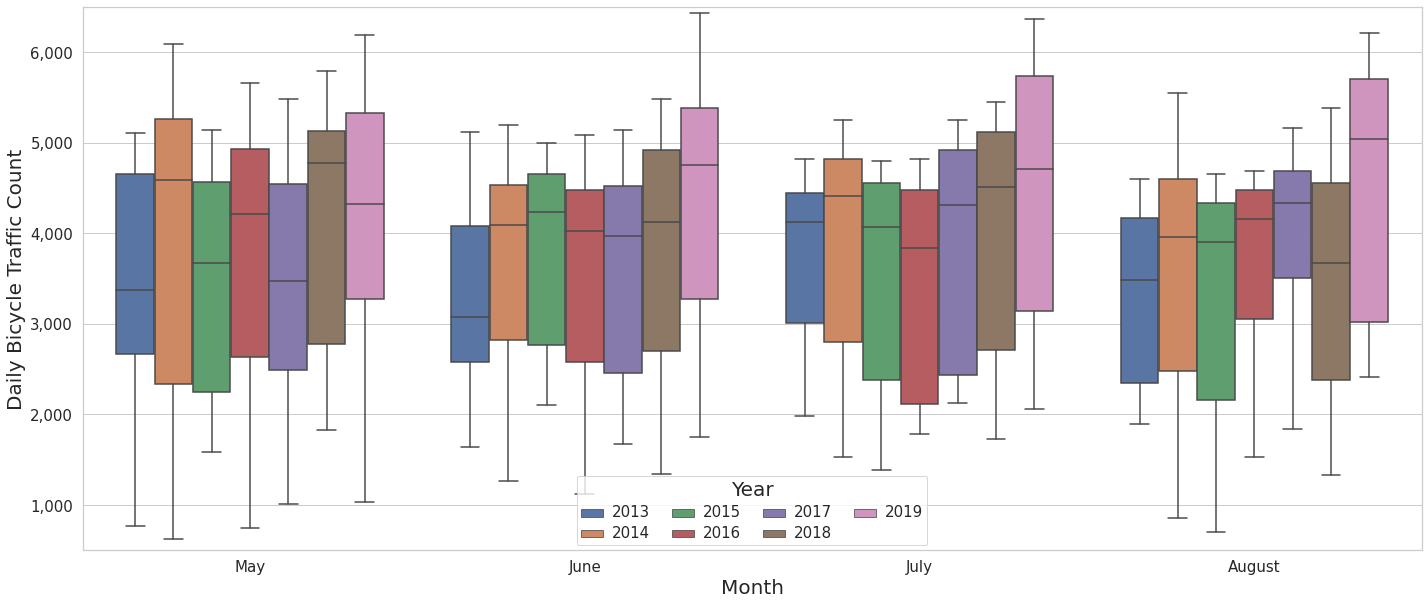
*Figure 3.* Total annual cyclists; 2013 to 2021 comparative

To better understand the seasonality of ridership on the Fremont Bridge, a series of boxplots (Figure 4) was generated to show the general distribution of daily ridership for each month across the pre-pandemic years (2013-2019). Code is [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/35feefae020cdb42340aa3052049a82a19428ea5/boxplots.py). From this plot, we can see that the summer months (May, June, July, and August) experience the highest ridership while the winter months (December, January, and February) experience the lowest ridership. The Fall (September, October, and November) and Spring (March and April) months, between the distinctive Summer and Winter periods, both taper down/up from the previous seasonal extreme.

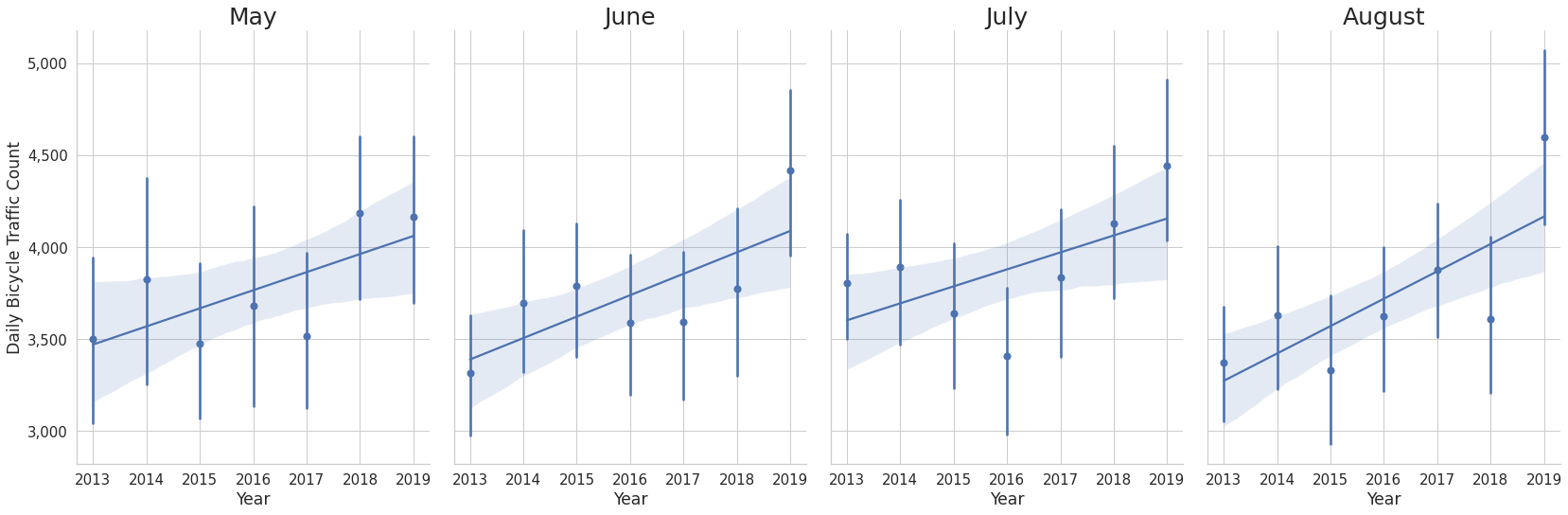


*Figure 4.* Daily distribution of cyclists by Month; 2013 to 2019 inclusive

Ignoring the pandemic affected years, we should expect to see an increase in ridership over the past decade (2013-2019). To narrow the focus of the data under review, only the summer months (May, June, July, and August) will be assessed. See Figure 5 for a plot of the distributions of daily riders for each month and color-coded to the associated year. A best-fit linear regression curve was also fitted to this data (Figure 6) to understand if the overall trend of ridership has been going up for these intervals across the four months observed. It is observed that the ridership does increase across all four months at approximately the same rate. Code for both plots can be found [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/35feefae020cdb42340aa3052049a82a19428ea5/boxplots.py).

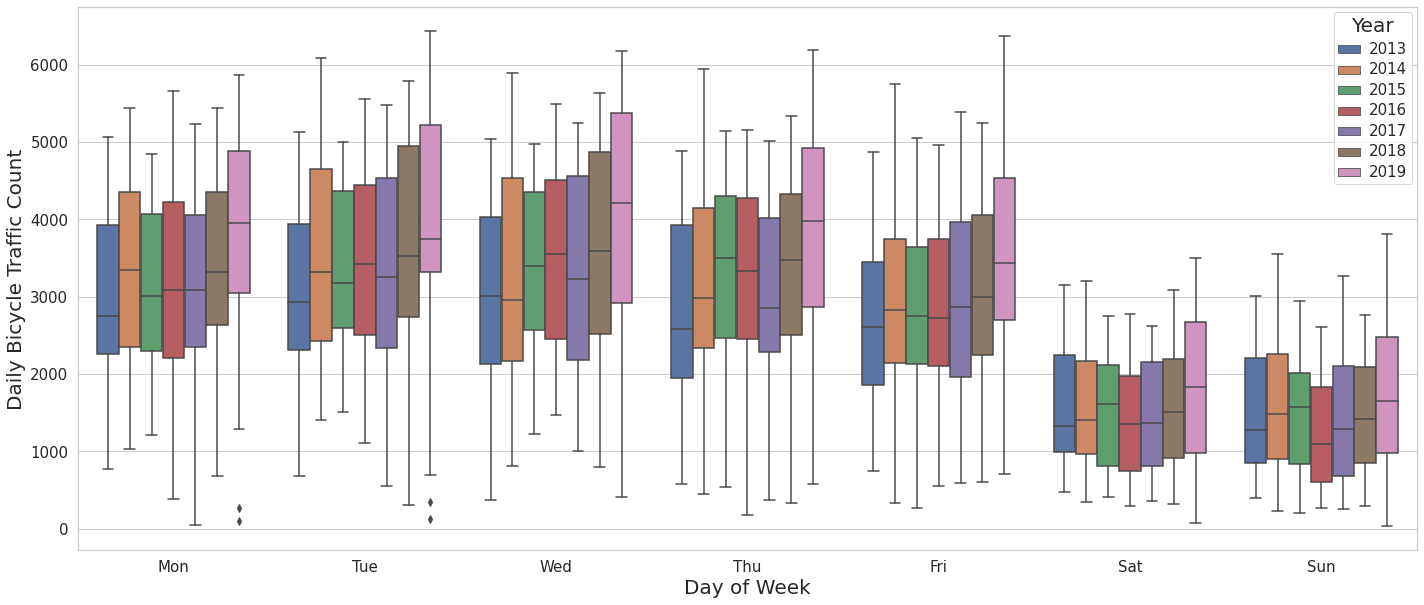


*Figure 5.* Daily distribution of cyclists by Month and year; 2013 to 2019 comparative



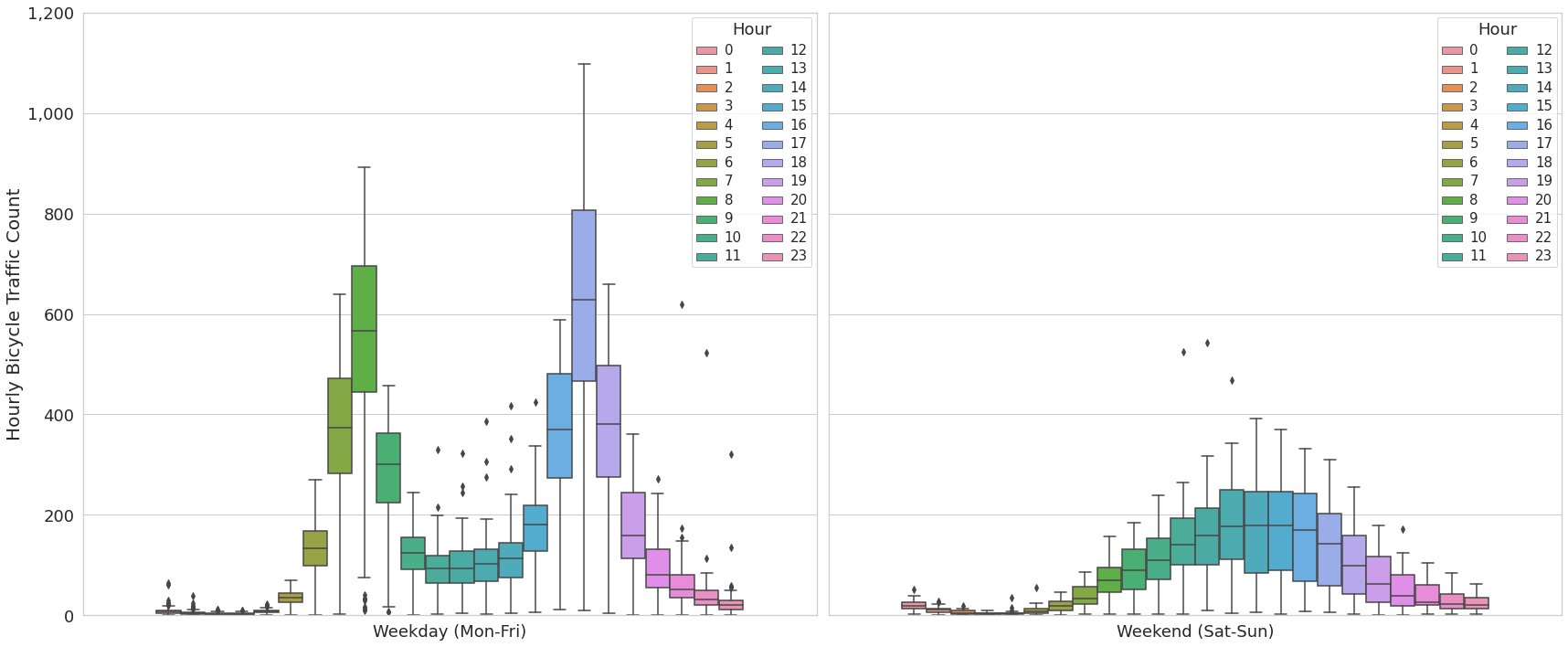
*Figure 6.* Linear regression fit of daily ridership in summer months across years; 2013 to 2019 inclusive

While the season does influence the number of cyclists counted on a daily basis, the next time-series level to review is the day of the week. Figure 7 shows the daily distribution of cyclists by weekday for the entire year, color-coded to the associated pre-pandemic year (2013 to 2019). Code is [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/35feefae020cdb42340aa3052049a82a19428ea5/boxplots.py). From these distributions we see that weekdays (Monday through Friday) experience around double the total traffic compared to weekend days (Saturday and Sunday). This is likely due to commuter traffic on weekdays and less cyclists crossing the bridge on weekends.



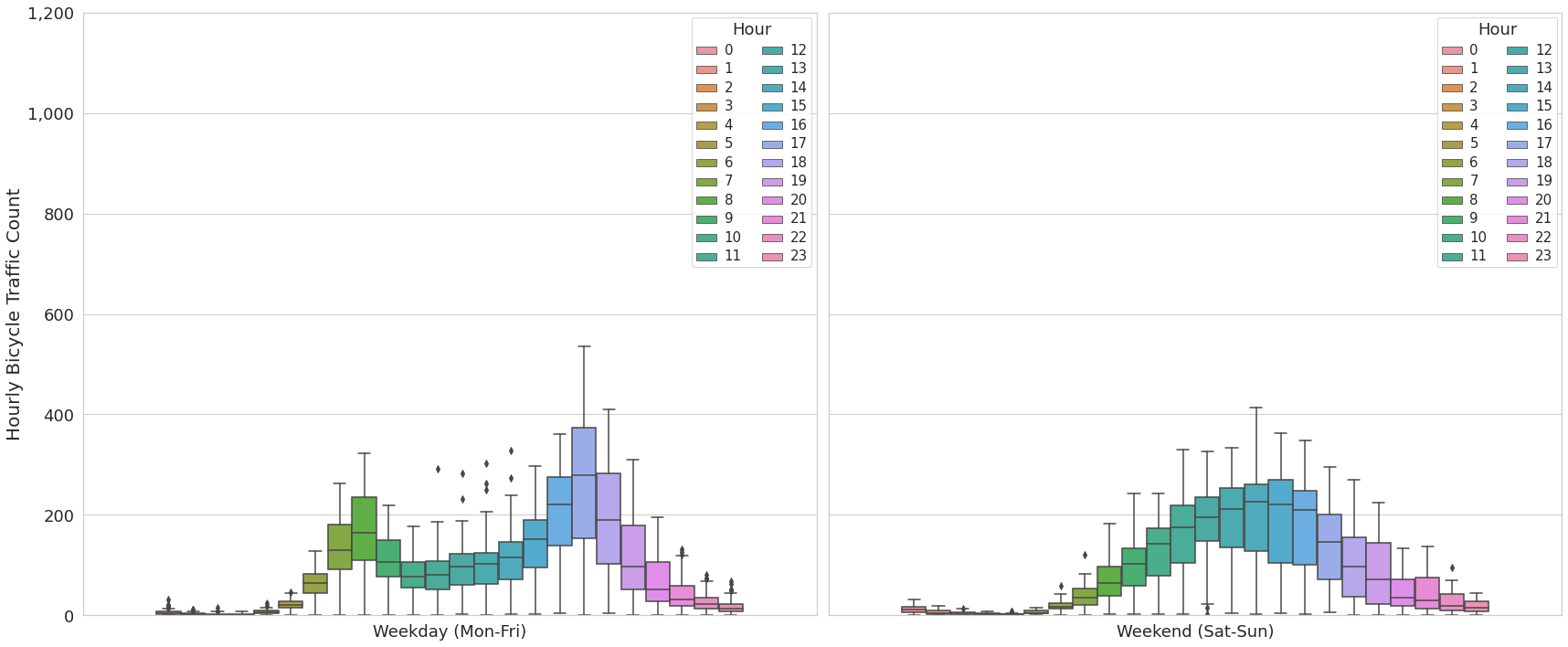
*Figure 7.* Daily distribution of cyclists by weekday over 2013 to 2019

In order to understand the hour-dependency of bicycle traffic on weekdays versus weekends, the full year 2019 data was plotted (Figure 8) as a series of boxplots by hour on both weekdays (Mon-Fri) and weekends (Sat-Sun). Code is [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/35feefae020cdb42340aa3052049a82a19428ea5/boxplots.py). It is apparent that the weekday data is strongly bimodal in its distribution with commuter traffic peaking at 9AM and 6PM at an average of 600 cyclists per hour. The mid-day traffic is much lighter, hovering around 100 cyclists per hour. This trend is complemented by the standard distribution of cyclists on weekend days which shows peak traffic occurring throughout the early to mid-afternoon.



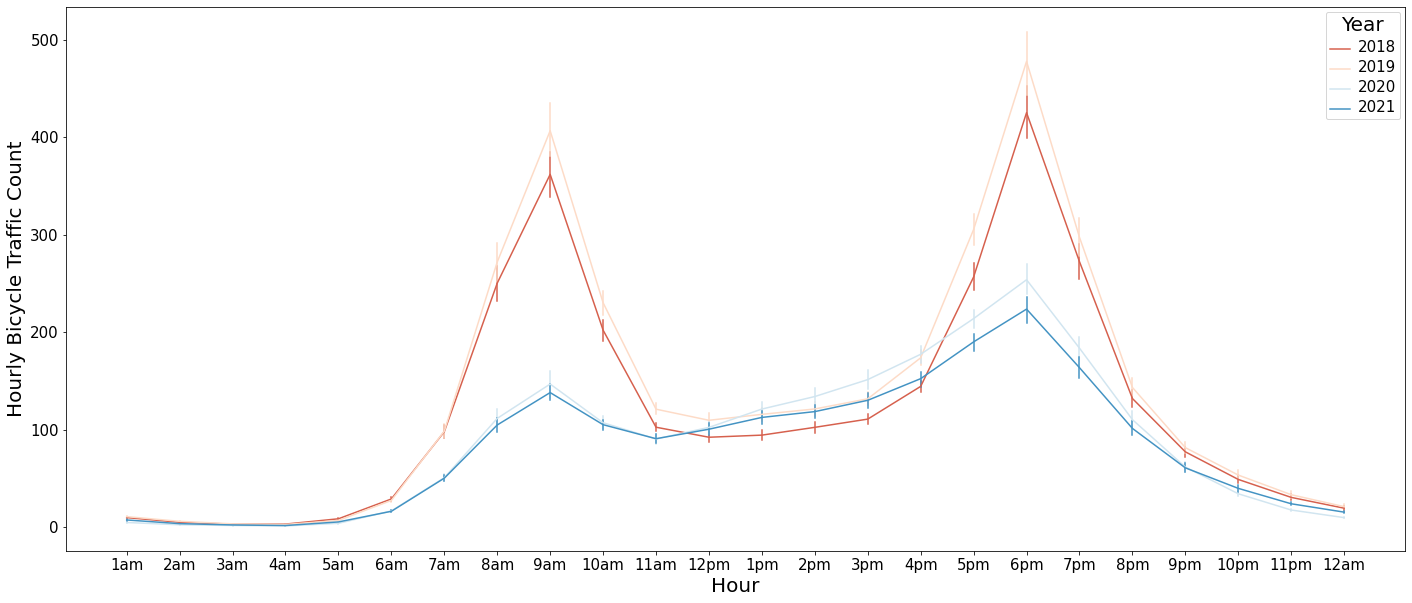
*Figure 8.* 2019 hourly distributions of cyclists by weekday and weekend days

Comparing the full year 2019 data to full year 2021 data gives a good comparison of pre-pandemic and post-pandemic cyclist traffic. Using the 2020 dataset would include a mix of data as the first several months of 2020 reflect pre-pandemic traffic trends. See Figure 9, below, for a comparison of weekday and weekend day hourly traffic distributions. Code used to generate this plot is [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/35feefae020cdb42340aa3052049a82a19428ea5/boxplots.py). These plots are to the same scale as the 2019 data and show that while the weekday data still shows a strong bimodal distribution, the peak commuter traffic numbers pale in comparison to 2019. The 2021 peaks occur around 200-300 cyclists per hour for the same weekday time frame. However, for the weekend days, the traffic appears to have increased through the pandemic. The peak traffic on the distribution is above 200 cyclists per hour for the same afternoon period compared to the 2019 numbers.



*Figure 9.* 2021 hourly distributions of cyclists by weekday and weekend days

The pre- and post-pandemic annual distribution of hourly cyclist traffic is also summarized in Figure 10, below. In this plot, the red lines represent 2018 and 2019 averages for each hour and the blue lines represent 2020 and 2021. Code is [here](https://github.com/arthursetiawan/DATA_5100_Project/blob/35feefae020cdb42340aa3052049a82a19428ea5/boxplots.py). It is apparent here that the commuter traffic previously peaking at 9am and 6pm reduces drastically from 2019 to 2020. However, the average count of cyclists between 12pm and 4pm is higher, on average, in both 2020 and 2021 compared to the previous two years. The main takeaway from this plot is that commuter traffic has reduced, but a greater percentage of people are cycling throughout the day as opposed to only during rush hour as a primary means of transportation.



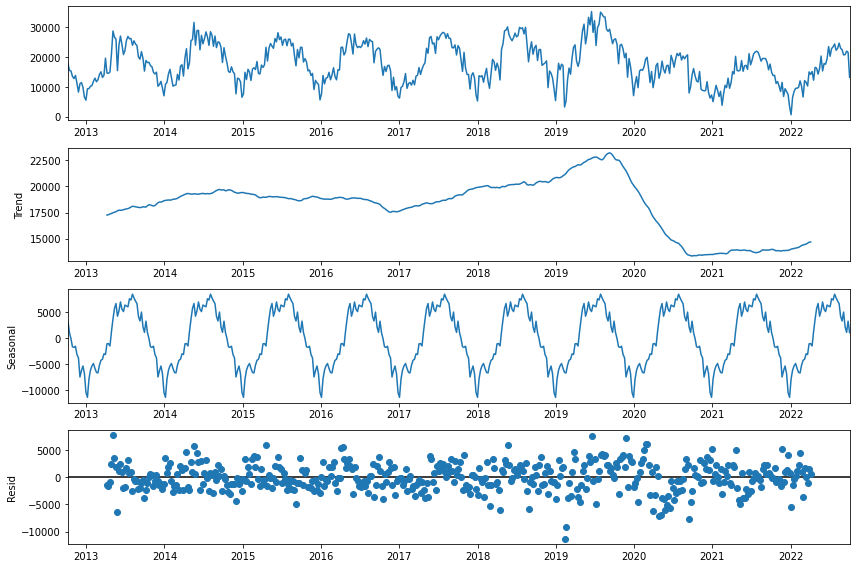
*Figure 10.* Average annual count of cyclists by hour; 2018, 2019, 2020, and 2021 comparative; bars represent 95% confidence interval

Analytical Approach

Upon viewing the Fremont Bridge Bicycle Counter dataset, we can conclude that this is a multivariate time-series dataset. In our forecast modeling, we have simplified the data into weekly summaries of total cyclist counts. The chosen time series interval (weekly) is our independent variable and the totalized bicycle count is our dependent variable in all summaries and models that we have developed.

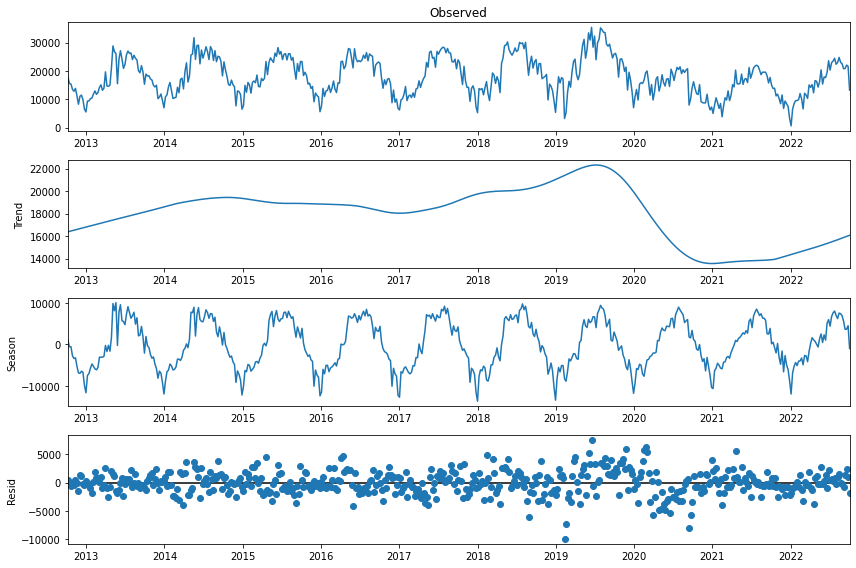
Statsmodels[[1]](#footnote-0) is a Python module that provides functions for the estimation of statistical models, conducting statistical tests, and performing statistical data exploration. Statsmodels contains a subset of time series analysis (statsmodels.tsa[[2]](#footnote-1)) model classes and functions. Within this time series analysis module, there are a handful of seasonal decomposition methods that may provide interesting insight to this dataset.

The first and simplest method, moving average seasonal decomposition, (seasonal\_decompose[[3]](#footnote-2)) utilizes moving averages to produce a seasonal decomposition of the time series data. The approach utilizes a naive decomposition method where the trend is first estimated using a convolution filter. The trend is then removed from the series and the average of the remaining series is returned as the seasonal component. An example of this method applied to the weekly time series data is below in Figure 11.



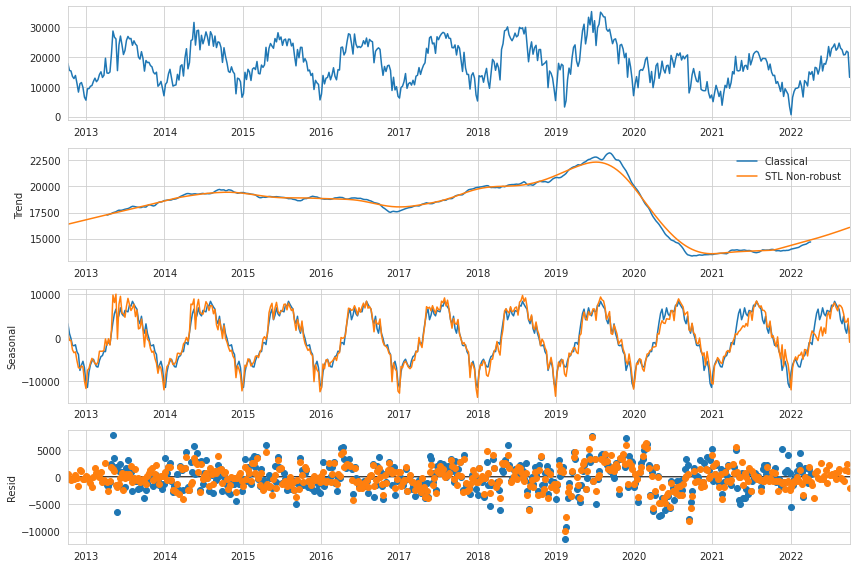
*Figure 11.* Seasonal decomposition of weekly cyclist count on Fremont Bridge using classical moving average approach

While this method does a good job of summarizing the seasonal component of the overall trend, this approach is considered rudimentary and a more sophisticated method is generally preferred. A second time series analysis method available in statsmodels is STL[[4]](#footnote-3) (Season-Trend decomposition using LOESS) where LOESS stands for LOcally Estimated Scatterplot Smoothing. LOESS builds on the simplistic classical moving average method by fitting simple models to localized subsets of the data to build a function that describes the deterministic variation in the data. This allows the user to fit a robust model without having to specify a global function. One trade-off for using this method is the computational intensiveness of its application. An example of this method applied to the weekly time series data is below in Figure 12.



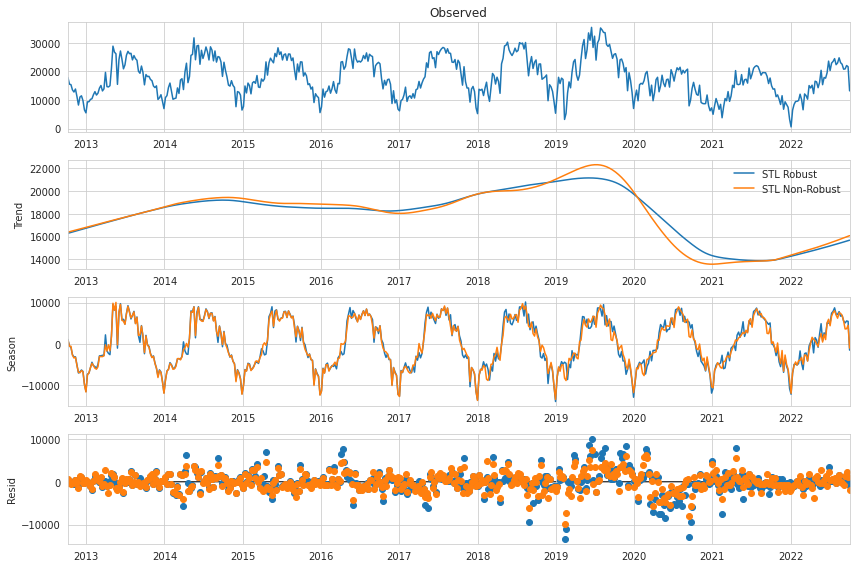
*Figure 12.* Seasonal decomposition of weekly cyclist count on Fremont Bridge using non-robust STL approach

One way to evaluate the fits of the classical and STL/LOESS method is to compare the residual errors of each model. See Figure 13 below for an overlay of the previous two figures. Here we see that the STL model (orange) produces a tighter fit to the data. This is especially true for the pre-pandemic years (2013-2019). The residual error appears to increase for both models in the transition to the pandemic-affected years (2020+). The STL method does generally perform better for all residuals, however.



*Figure 13.* Seasonal decomposition of weekly cyclist count on Fremont Bridge using classical and non-robust STL approaches

Another factor to consider when using the LOESS method is whether the robust estimation option is enabled. The robust option allows for a data-dependent weighting function to re-weight data thus allowing for the model to tolerate larger errors. This difference is highlighted in the residual errors plot at the bottom of Figure 14, below. Here, however, we see that the robust model has visibly larger errors (blue) for several points compared to the original, non-robust LOESS model (orange) from Figure 13, above. However, for a majority of the points, the robust model has smaller residual values. This will be discussed further in the following section.



*Figure 14.* Seasonal decomposition of weekly cyclist count on Fremont Bridge using robust and non-robust STL approaches

Facebook Prophet[[5]](#footnote-4) is a forecasting procedure for time series data based on an additive model where non-linear trends are fit with yearly, weekly, or daily seasonality. The method also allows for the application of holiday effects. This procedure works best with strong seasonally affected time series and readily handles outliers in the dataset. Prophet’s fitting interface is identical to scikit-learn’s (sklearn[[6]](#footnote-5)), where Prophet classes are created and then prediction is achieved by calling the appropriate fit methods. For any instance, a DataFrame of two columns namely and are passed as inputs. The column corresponds to the datestamp in the data while the column is the data to be forecasted. The column data must be numeric data. As a result, the index and values column of the data used in this project were formatted to match this format. The forecasting procedure is then specified by passing it into the constructor. An inbuilt helper method, namely Prophet.make\_future\_DataFrame,generates a new DataFrame based on the specified number of days. The new DataFrame contains both the predicted data and historical data used for prediction and as a result is easier to discern the fitted model. When fitting our model using this approach, we worked into the future by specifying the desired future prediction days, for instance 365 days for 1 year, 30 days for one month and so forth. In addition to the predicted value***,*** the upper limit and lower limit are also returned in the DataFrame as ***yhat\_lower, yhat\_upper*** respectively. A timeseries plot can then be created by calling the Prophet.plot method to visualize the results.

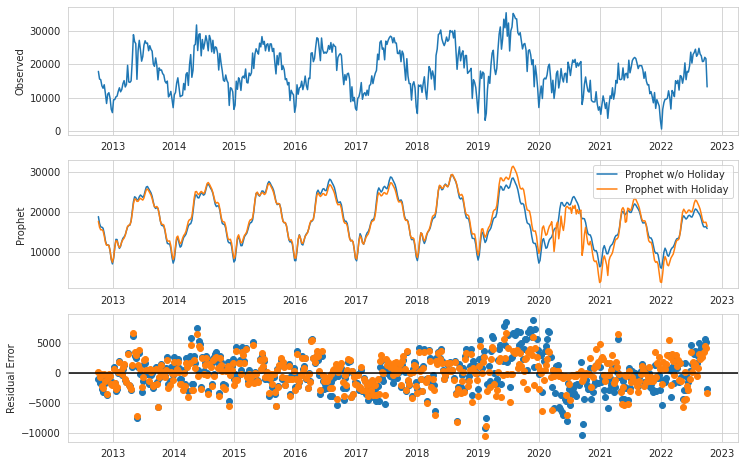
Facebook Prophet has other default functionalities for visualizing the weekly and yearly seasonality, as well as the trend. For more specific insights, ***add\_seasonality*** method can be used to add custom seasonalities. Seasonalities of the Prophet method are of fourier order with certain default orders (Taylor & Letham, 2018). Orders determine the rate of seasonality change and consequently, the accuracy of the fit. For instance, the default seasonality for a year is 10, but the order value can always be adjusted upwards when instantiating the model to match higher frequency changes and vice versa as explained in this [article](https://peerj.com/preprints/3190/). Even though the Prophet method assumes consistency in seasonalities, factors that result in inconsistencies such as weekday and weekends, summer and winter and so forth, can be addressed by introducing conditions.

As discussed previously, there are several metrics for evaluating uncertainty for the Prophet model. In this project, uncertainty was measured by introducing a 95% confidence interval. It is worth noting that the default confidence interval for the Prophet model is 80%.

In addition, in order to estimate how best the model values fits the existing data, the residual errors were also calculated and plotted over the years. These residuals were calculated by simply subtracting the predicted values, yhat,from the actual bike count values . Figure 15 shows a plot of the actual and predicted values as well as the resultant residual errors.

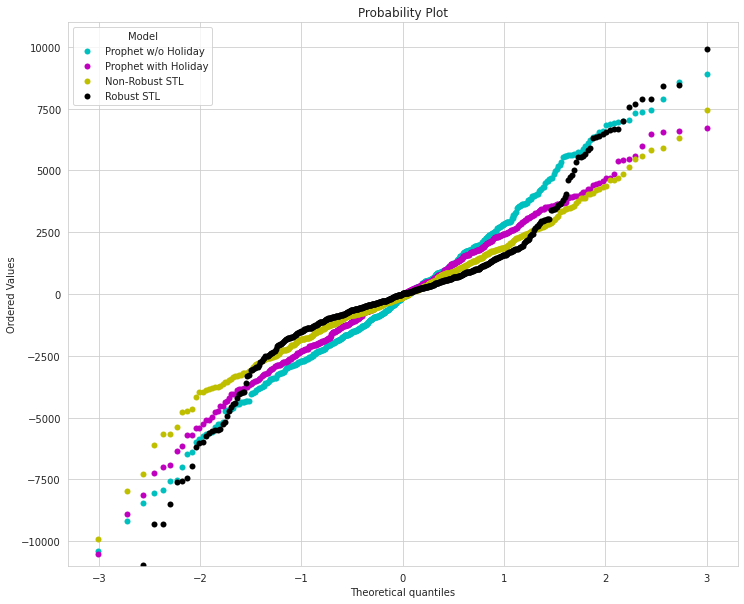
From Figure 15, it appears that the variability of the residual errors seem to be highest between late 2019 and 2021. This is as a result of the significant under estimation of the bike count as evidenced by the plots of predicted and actual values for the 2019 and 2020 season on Figure 15. Contrastingly, the model overestimates the prediction for the period between 2020 and 2021. When reviewing the prediction values for the entire period, we note that the 2021/2021 is the first period where the predicted values are consistently above the actual values. This points to the impact of covid where people who biked to had to work from home.

To investigate the potential impact of covid on the performance of the Prophet model, an alternative modeling was carried out. In covid modeling,all the days falling under the lockdown period were treated as holidays as considered by (Taylor & Letham, 2018).

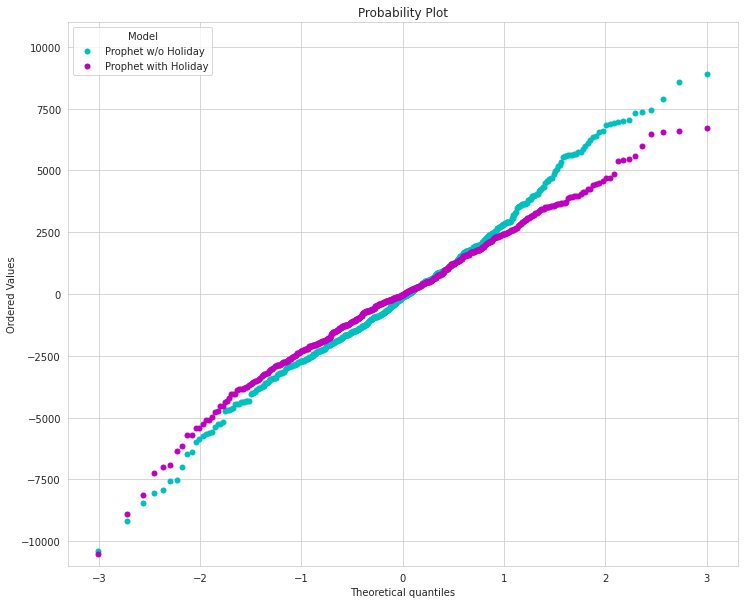


*Figure 15.* A plot of observed and Prophet predicted values with corresponding residual errors for both conditional (with holiday) and unconditional (without holiday)

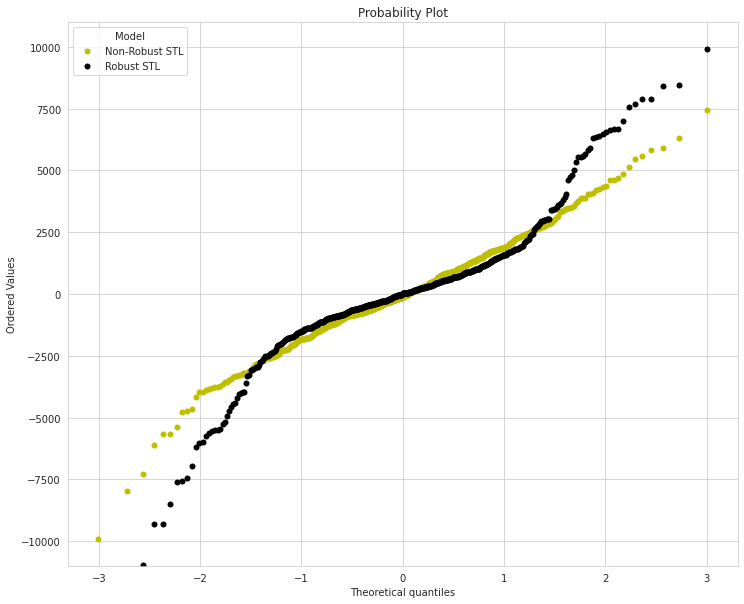
A comparison of the residuals for the four methods considered above may be plotted as a normal Q-Q probability plot (Figure 16). A Q-Q, or quantile-quantile, plot is a graphical tool to assess the normality of a dataset. The normality is related to the linearity of a plot; a curvy or s-shaped line indicates outliers towards the ends of the distribution. In this case, when all four models are compared against one another, we see that the robust STL is the least linear. The non-robust STL model appears to be flatter in comparison. The Prophet models are much more linear in comparison to either STL method. In order to see the detailed comparisons between the two sets of plots, Figure 17 and 18, below, accompany Figure 16.



*Figure 16.* Normal Q-Q probability plot between the robust STL, non-robust STL, holiday conditional Prophet, and holiday un-conditional Prophet model fits

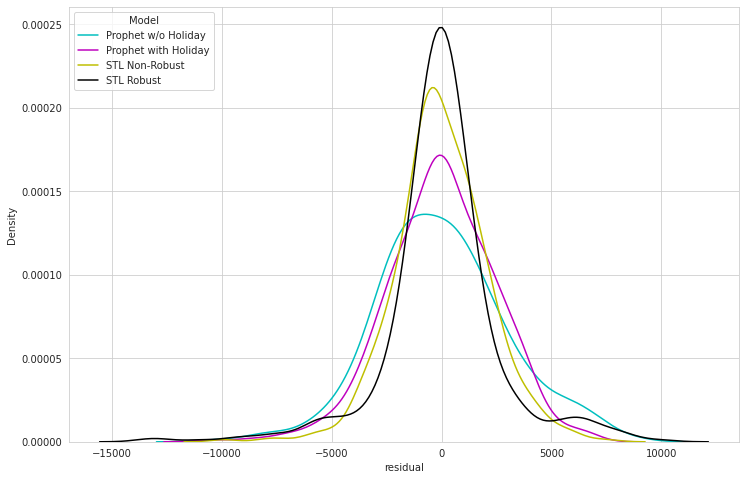


*Figure 17.* Normal Q-Q probability plot between the holiday conditional Prophet and holiday un-conditional Prophet model fits



*Figure 18.* Normal Q-Q probability plot between the robust STL and non-robust STL model fits

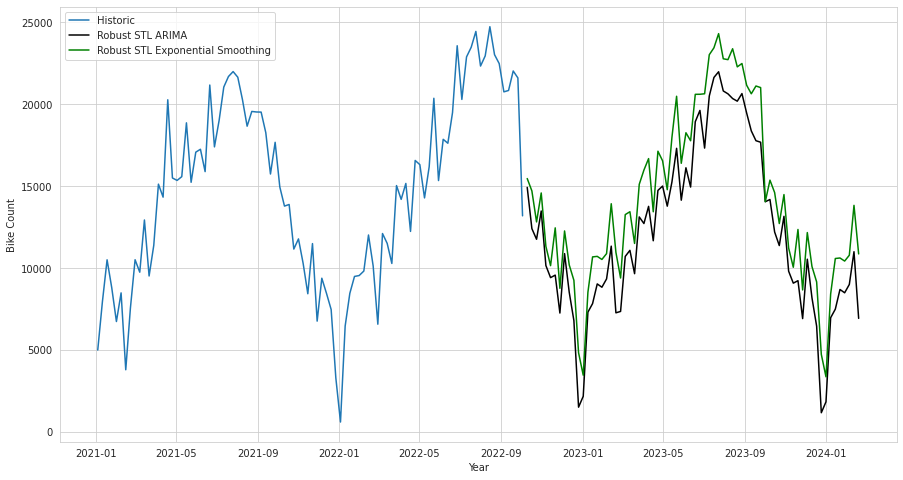
Another way to visualize the normality of these residuals is with a kernel-density estimation (KDE) plot (Fig 19). In this plot, a distribution that is centered away from the y-axis indicates a skewed distribution. A higher probability/density value near the y-axis indicates smaller overall residuals. We can see that the robust STL fit appears to perform best overall as it is centered on the y-axis and has a high density of probability around zero. However, the STL robust fit does show small peaks indicating significant outlier residuals. The second best fit is the holiday-adjusted Prophet model which is also centered on the y-axis. This plot shows fewer outliers when compared to the robust STL model fit. The non-robust STL and unconditional Prophet models are both centered left of the y-axis indicating skewing in the residuals. This is likely due to the models not accounting fully for the disruption of COVID-19 on the time series trend.



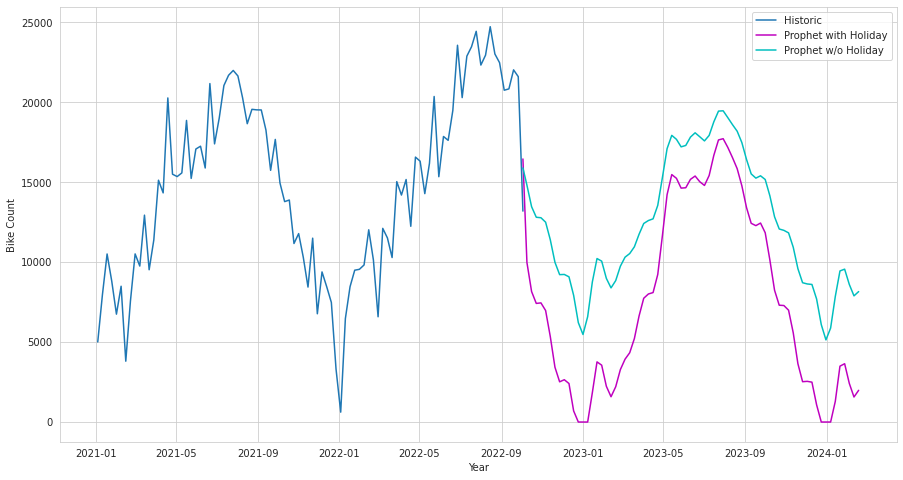
*Figure 19.* Kernel density estimation (KDE) probability plot between the robust STL, non-robust STL, holiday conditional Prophet, and holiday un-conditional Prophet model fits

It is possible to produce a model-based forecast using STL. Forecasts are produced by first decomposing the seasonality estimated by STL and then forecasting this deseasonalized data using a time-series model. These methods are combined using the STLForecast[[7]](#footnote-6) function. An Example of the data forecasted ahead by 72 weeks using ARIMA and Exponential Smoothing[[8]](#footnote-7) is shown below in Figure 20. ARIMA[[9]](#footnote-8) (Auto Regressive Integrated Moving Average) combines both autoregressive (AR) and moving average (MA) models as well as a differencing, pre-processing step called integration (I). It was decided that because the robust STL method performed better on residual distribution than the non-robust method, the robust method is used for forecasting models.

Figure 20.72 shows the predicted trends observed in the two forecasting techniques for STL modelsl. We observe that the trends from both techniques are fairly consistent with the Exponential Smoothing having a slightly higher estimation. If the observed trend and peak values were expected to be identical to the peaks of the historical years, Exponential Smoothing would be considered a better prediction method.



*Figure 20.* 72 week forecast of weekly cyclist counts on Fremont Bridge using robust LOESS seasonal decomposition with ARIMA and with exponential smoothing forecast methods

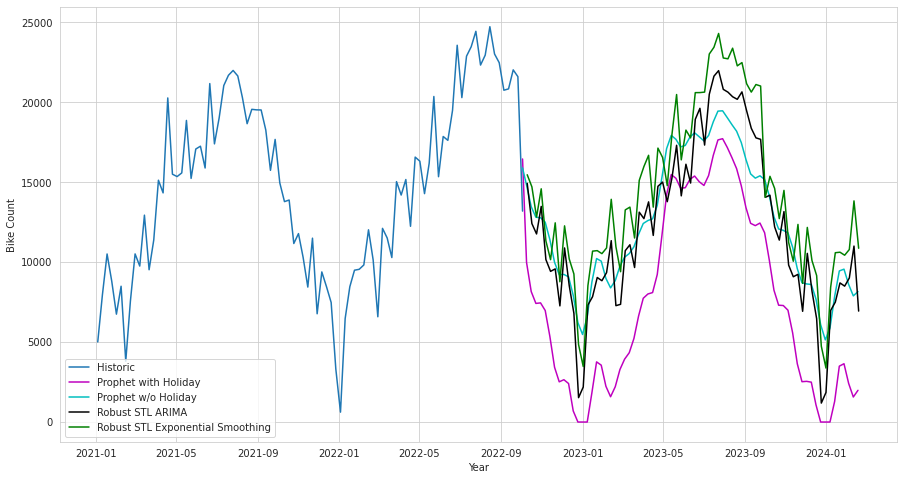
Figure 21.72 shows Prophet predictions while factoring in the impact of covid. With lockdown as holiday condition in force, the model had a minimum bike count prediction of zero and a maximum of around 18,000. The minimum value of zero was not the actual minimum, but rather, the clipped value for our model. It was noted during the prediction process that the model was predicting negative bike count values when the lockdown condition was in place. We therefore had to limit the lowest predicted bike count to zero as the count can not be negative in real life. When comparing the covid model and without covid model, we note the predicted values for the covid model are much lower for the next 72 weeks. The maximum value without covid consideration was around 20,000 while the minimum was around 2,000. The negative values prediction by the covid model can be attributed to the introduction of holidays as holidays. As a result, when the model trys to fit with many holidays occurring together, the trend in seasonality becomes complex resulting in higher variability which in turn causes the predicted values to be out of range. However, in both scenarios, Prophet model suggests that the bike count peaks will be lower with and the downward trend will continue for the next 72 weeks. It is also evident from Figure 21.72 that there is great variability in the predicted values for the two models. Prophet allows diagnostics to determine the outliers , uncertainty and errors that can also be factored in as explained in this [Github page](https://facebook.github.io/Prophet/docs/seasonality,_holiday_effects,_and_regressors.html). Because the seasonalities in Prophet are pegged to Fourier orders, faster changing cycles can be improved by increasing the Fourier order. However, increasing the Fourier order can have a drawback of overfitting the model. 

*Figure 21.* 72 week forecast of weekly cyclist counts on Fremont Bridge using Facebook Prophet forecasting with and without conditions set for COVID-affected data

Figure 22 summarizes the performance of the various algorithms used in this project- python Statsmodels compared to the Facebook Prophet models. We also attempted a third model using TensorFlow library to model for different windows in our dataset. However, because our prediction period was 72 weeks, weekly accumulations of the bicycle counts in TensorFlow’s application was not enough testing data to arrive at consistent results that could be used for comparison with the first two algorithms. As a result, the findings and insights discussed here are derived from statsmodels and the prophet models. The metrics used for comparison were the predictions over the next 72 weeks. The findings discussed here treat the combination of the statsmodels and facebook model as four different models. That is, STL-robust exponential smoothing, robust STL ARIMA, general prophet model (w/o holiday) and prophet with conditionality (holiday model).

When reviewing the prophet model there are interesting findings. Firstly, both prophet models appear to have more consistent residual errors with a more distinct change in trend while transitioning into the pandemic affected period. Specifically, this change in the residual errors trend was more apparent with the conditional prophet model with the consideration of the covid factor impact. Overall, the prophet models had significantly lower but highly variable residual errors compared to STL models shown on KDE plot on Figure 19.

When reviewing the five models fitting into the data and bike count prediction for the next 72 weeks, the results were significantly different. Because of the lower residual errors discussed previously, the prophet models resulted in better fitting to the historical data compared to the STL models. However, these results did not appear to be consistent when predicting 72 weeks into the future. Concernedly, prophet prediction with covid factor consideration had some illogical negative bike count predictions. To circumvent this shortfall, minimum predictions had to be capped at zero. This was not the case with any of the other three models. Another interesting finding is the higher predicted bike count values by the STL models compared to the prophet models. The peak values for the STL models were close to 25,000 while the prophets prediction was well under 20,000. The resulting difference is quite significant and prophet model’s prediction appeared to defy the expected predictions. Even with the covid functionality enforced to prevent the exaggerated impact of covid, the predicted values still appear to be off-because all the predicted values are below all the historical values for the past two years.

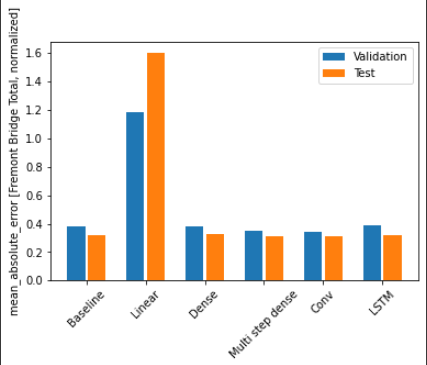


*Figure 22.* Comparison of 72 week forecast of weekly cyclist counts on Fremont Bridge using Facebook Prophet forecasting with and without conditions set for COVID-affected data and robust LOESS seasonal decomposition with ARIMA and with exponential smoothing forecast methods

Having reviewed the four fittings as shown on Figure 22 and factoring in the residuals error trends we can draw the following take aways

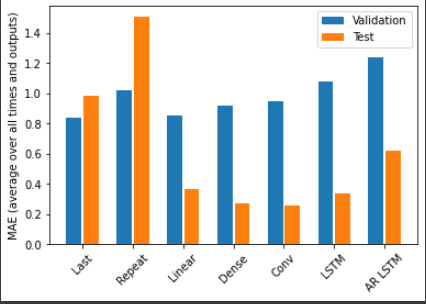
* The Holiday prophet model has the lowest variability when fitting into the historical data. However, the possibility of obtaining negative predictions renders the model to be the least preferred .
* The predictions for Robust STL ARIMA and Robust STL Exponential Smoothing average are close apart from the resulting peak values. The huge difference in the without holiday prophet model can be attributed to larger fittign errors. Since the Robust STL ARIMA is the most consistent with historical data for the past two years, it would the most preferred for decision making. This is consistent with the KDE plots for the residual errors shown in Figure 19 where Robust STL model had the lowest residual errors. The residual errors were very close to zero

We attempted to use the TensorFlow library that builds upon different model styles which includes Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), split into two sections of single time step forecasting and multiple time step forecasting. TensorFlow models create predictions based on specified windows of consecutive samples from our bicycle counter dataset. It then utilizes a split\_window method which allows the multiple inputs to convert them into a window of inputs and window of labels. It then requires transferring these input and label windows into a TensorFlow’s tf.data.Dataset function which converts the bicycle counter time series DataFrame. For the sake of consistency between all group members, we collectively used weekly accumulations of the bicycle counts which in TensorFlow’s application was not enough testing data to arrive at the agreed upon forecast of 72 weeks into the future.

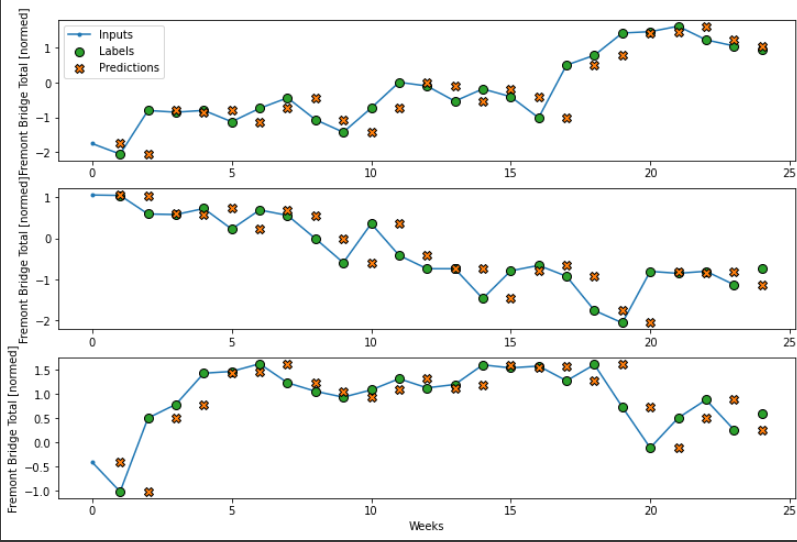


*Figure 23.* Single Step Model Performances Comparison of 24 week input (historical data) and 24 week prediction wide window using TensorFlow

For single step models, we tried out the baseline model which in reality just takes input values and returns the current count which predicts that there is no change or little change in the prediction. Below is a wide\_window example of the baseline predicting 24 weeks into the future with a 24 week historical dataset. Keep in mind that the input and output for a wide window has to be the same.

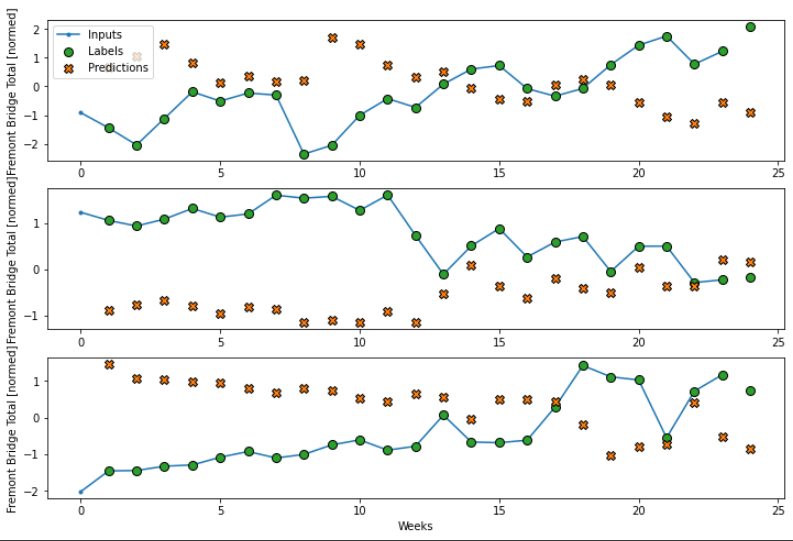


*Figure 24.* Multi Step Model Performances Comparison using TensorFlow



*Figure 25.* Wide window baseline model for 24 week forecast window

We also tried out a linear, dense, multi-step dense, CNN, and Long Short Term Memory (LSTM) which represents the RNN section. Upon comparing the performances of each model by calculating for the mean squared error for validation and test splits respectively, we see that the linear model by far performed the worst out of them all, whereas the other models performed somewhat equally with the convoluted model performing the best. We can see on the figure below that the linear model had very poor predictions.



*Figure 26.* Wide window linear model for 24 week forecast window

For multiple step models, we have a choice of either single-shot models where predictions happen at a single instance, or autoregressive models where prediction happens one at a time and then it feeds back the result into the model. When comparing to performance of the single step models, multi step models conclusively had much higher mean average errors in the validation data, whereas test data mean average errors stayed practically the same except for the linear model, which dropped its MAE from around 1.6 to just below 0.4. The conclusion from further analyses of these models is that they are not a good fit for our dataset with weekly data, as it would be better suited for datasets with larger numbers of points like the hourly version of the data. Furthermore with the restriction of using only the same input and output number of weeks for predictions, predicting 72 weeks in the future with only a split of 365/104/53 week split for the training, validation, and test set resulted in an obstacle to compare this dataset with FBProphet and Statsmodels. Given a larger dataset and a smaller interval to predict on top of the complexity of setting up TensorFlow, it will be a great tool in the future to use for an alternative time series forecast methodology.

Take-aways & Future Developments

The performance of timeseries models is influenced by several factors. These factors include for instance weekday versus weekends, normal days versus holidays and so forth. In addition, other unpredictable factors such as the covid pandemic also influence the seasonality trend and hence the performance of the model as discussed in this study. Future modelling project can be conducted to include additional factors such as weather that affect the seasonality and the resulting trends.

This project focused on a single bridge which helped us to gain specific insights on the seasonality and make prediction for the next 72 weeks. Future projects can be expanded to include other geographical locations. The performance of the model in the different geographical areas can then be compared as a way of quantifying the influence of various factors that influence the model behaviour.

Lastly, a comprehensive study that includes various conditionalities for weather, holidays and extreme elements such as outbreaks pandemics can be conducted to understand indirect elements such as the working habits of people living in different geographical locations.

References

Balk, G. (2019, June 5). *Who bikes in Seattle? Bicycling gender gap one of the biggest in the U.S.* Retrieved from Seattle Times: <https://www.seattletimes.com/seattle-news/data/who-bikes-in-seattle-bicycling-gender-gap-one-of-biggest-in-country/>

Bicycle master plan, BMP. (2021, May). Retrieved October 11, 2022, from <https://www.seattle.gov/transportation/document-library/citywide-plans/modal-plans/bicycle-master-plan>

Bike counters. (n.d.). Retrieved October 11, 2022, from <https://www.seattle.gov/transportation/projects-and-programs/programs/bike-program/bike-counters>

Forecasting Atmospheric CO2 with Python. (2022, Mar.). Retrieved November 1, 2022, from: https://towardsdatascience.com/forecasting-atmospheric-co2-concentration-with-python-c4a99e4cf142

Darts: Time Series Made Easy in Python. (2020, Jun.). Retrieved November 1, 2022, from: https://medium.com/unit8-machine-learning-publication/darts-time-series-made-easy-in-python-5ac2947a8878

https://data-seattlecitygis.opendata.arcgis.com/search?tags=transportation

<https://www.seattle.gov/transportation/projects-and-programs/programs/bike-program>

<https://www.seattle.gov/transportation/projects-and-programs/programs/bike-program/bike-counters/fremont-bike-counters>

Taylor, S.J. and Letham, B., 2018. Forecasting at scale. The American Statistician, 72(1), pp.37-45.

1. Statsmodels: <https://www.statsmodels.org/dev/index.html> [↑](#footnote-ref-0)
2. Statsmodels.tsa: <https://www.statsmodels.org/stable/tsa.html> [↑](#footnote-ref-1)
3. Statsmodels.tsa.seasonal.seasonal\_decompose: <https://www.statsmodels.org/devel/generated/statsmodels.tsa.seasonal.seasonal_decompose.html> [↑](#footnote-ref-2)
4. statsmodels.tsa.seasonal.STL: <https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.STL.html> [↑](#footnote-ref-3)
5. Facebook Prophet: [https://facebook.github.io/Prophet/](https://facebook.github.io/prophet/) [↑](#footnote-ref-4)
6. Sklearn: <https://scikit-learn.org/stable/> [↑](#footnote-ref-5)
7. STLForecast: <https://www.statsmodels.org/dev/generated/statsmodels.tsa.forecasting.stl.STLForecast.html> [↑](#footnote-ref-6)
8. Exponential Smoothing: <https://www.statsmodels.org/dev/examples/notebooks/generated/exponential_smoothing.html> [↑](#footnote-ref-7)
9. ARIMA: <https://www.statsmodels.org/dev/generated/statsmodels.tsa.arima.model.ARIMA.html> [↑](#footnote-ref-8)