Machine Learning Final Assignment

Constantin Pusch - 20304226

Contents

Part One

Introduction

The goal of this assignment is to assess the impact of the covid pandemic on Dublin's city-bike scheme. The first major decision was to define the date range of the pandemic. The dates I chose for the pandemic were 29-02-2020 as the start of the pandemic and 21-01-2022 as the end. The first date correlates to the first covid case that was detected in Ireland and the latter is when the government removed all restrictions and no new ones implemented at a later date. One thing to note with these dates is that the Dublin bike dataset only has data until 01/01/2022 so there is no post pandemic data and, as discussed later, it is difficult to forecast the post pandemic impact.

Data Preprocessing

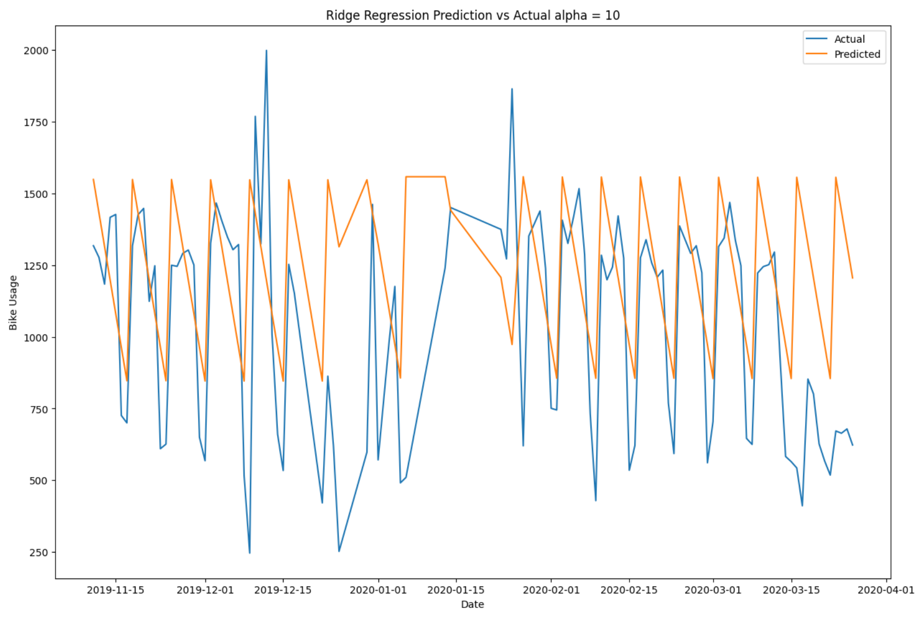
As mentioned in the problem statement, it is crucial to determine a method of measuring bike usage. To measure usage, I decided that each time a bike is removed from a station it counts as one use. This figure is calculated by taking the inverse difference of the Available Bikes per day. Since the bike has to be returned eventually, at an unspecified time, I did not count a return as a use. [1] I decided to keep this a simple measurement since there are multiple unknown variables that could affect bike usage. To prove this point I will give two examples. When I was processing the data, I noticed that the total number of bike stands would fluctuate. This could have an impact on the total number of bikes in the system, thus potentially increasing usage, or have no impact at all. Another point is that bikes are removed from the system for maintenance. It is unclear weather this is recorded in the data or not. Overall, my point is that there are many unknowns and therefore the bike usage figure is simple and will give a general overview not an exact usage figure.

To use the data in the model I combined all the datasets into one and aggregated all the five-minute timestamps of recorded data for all stands into a daily usage figure. Once I compiled this combined data set, I calculated the usage figure for each day as described above. I believe that a daily usage figure is enough granularity for making a long-term forecast.

Model Used

Originally, I intended to use a Ridge regression to forecast the effect that the pandemic had on the bike scheme. After optimizing the alpha value, I was getting a model with not ideal results that did not seem to accurately predict far enough out into the future. I had added additional parameters to attempt to capture the impact of seasonality, such as day of the week and month of the year. These hyperparameters increased the accuracy of the model but I still felt that it was overgeneralizing too much. In order to overcome these short comings, I decided to do some research to find a better model to use.

[1] As a quick note: throughout the report I will interchangeable use 1 'usage' as a rider even though these do not exactly corolate.



This graph shoes the ridge regression model predicting using the training data and not even forecasting. This gave me little hope for accurate forecasting and caused me to search for a better model to use.

The machine learning model I chose was developed by Facebook and is called Prophet. [2] Prophet is specifically designed for time series forecasting which gives it multiple advantages that allow for more accurate long-term predictions. The main advantages that made sense for this project were the following: Prophet accelerates at incorporating seasonality into its forecasts and is designed to work with daily data. This is perfect for the Dublin bikes data set. Prophet is also good at handling longer time horizons which is ideal when forecasting out bike usage for this scenario.

Prophet Technical Aspects

Prophet is based on an additive model where linear/non-linear trends are fit with yearly, weekly, and daily seasonality. The model can automatically detect points in the time-series data where trends change and then it can select the appropriate linear or logistic growth model.

Predictions are generated by splitting the time series into three main components: trend, seasonality, and holidays. For this scenario I have not had the model account for holidays. The trend aspect models non-periodic changes in the data using a piecewise linear or logistic growth curve. This allows for the model to predict different types of growth trends in the data. As mentioned in the lectures, the seasonality captures regular patterns that appear in the data and the model can account for daily, weekly, and yearly seasonality. The daily hyperparameter is used for intraday data so therefore I only selected yearly and weekly hyperparameters. Prophet uses an additive model where these separate components are combined to make the final forecast.

[2] https://facebook.github.io/prophet/

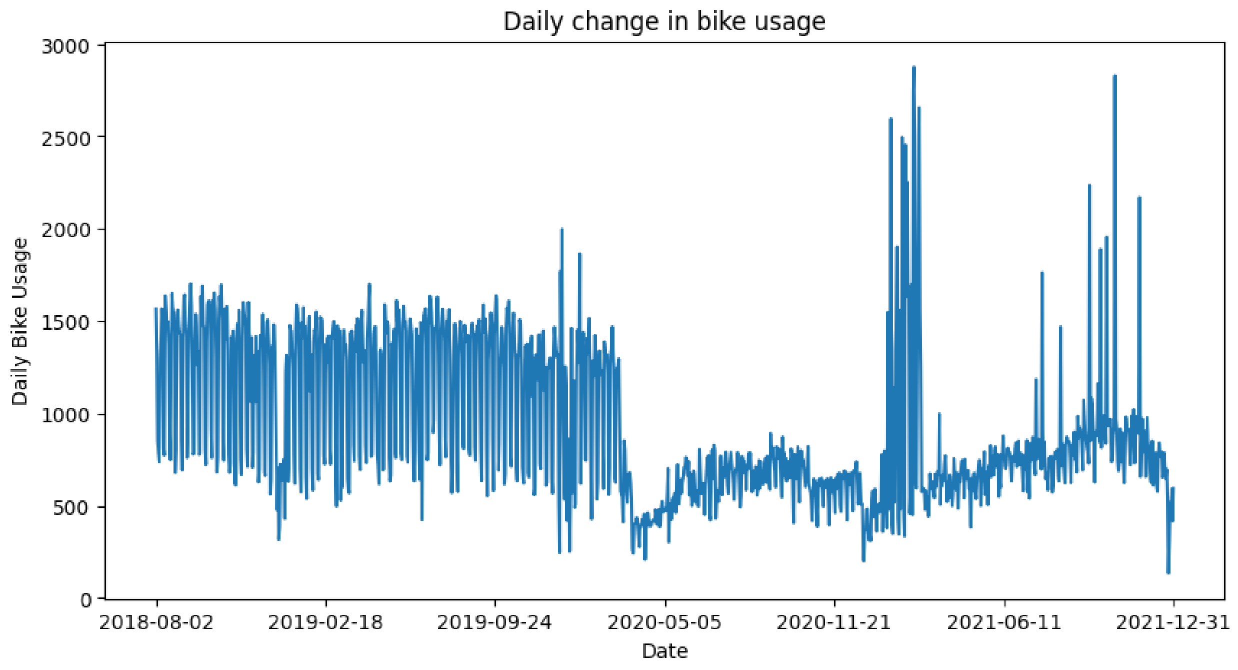
When creating the prophet model the cost function is automatically selected as it performs this step automatically. It typically varies between evaluating the results using mean absolute error or mean squared error. [3]

Results

Below is the raw data that was used to perform the predictions. As mentioned above the dates that I have defined as 'the pandemic' are from 29-02-2020 to 21-01-2022. Due to the few extreme outliers in usage (more explained in the chart caption) I have calculated the descriptive statistics using the median and not mean. Using the mean in this case gives a much more accurate representation of actual usage. The date range of recorded data is 02-08-2018 to 12-31-2021. The first day and last day of the actual dataset are omitted due to there not being a full day of recording on these days.

These are the descriptive statistics results:

Median over the whole period: 766.0, Pre Covid-Median: 1336.0, During Covid-Median: 675.0



This plot shows the original data that I have gotten using the bike usage statistic mentioned from above. There are some weird outliers during the covid period but since I could not pinpoint down why they were so high I decided to keep them in the data set. This definitely had an impact on predictions, but it was not trend breaking so I kept these points in since they well could be real usage. I do know that over this time period there were some updated being made to the system so this could explain these massive jumps. [4]

The graph and statistics clearly show that as soon as lockdown initiated there was a severe reduction in the usage of Dublin city bikes that had not recovered to pre-pandemic levels before the end of the

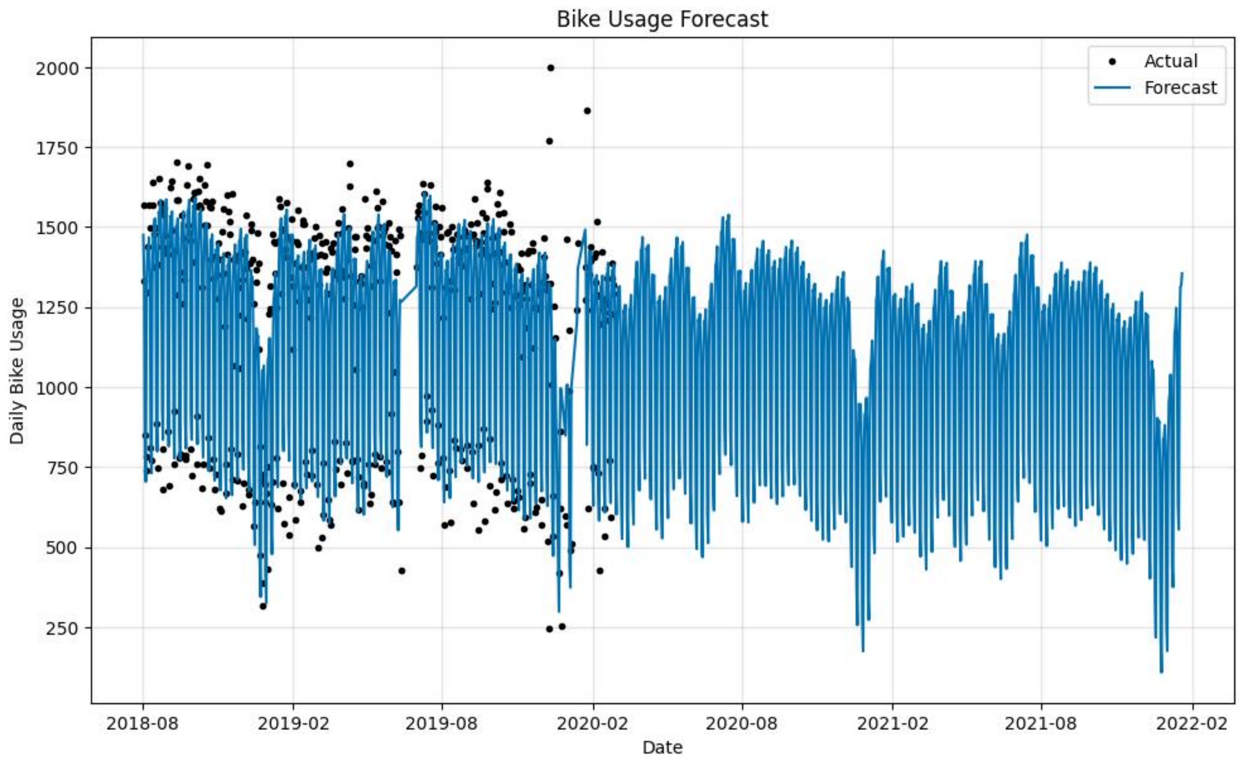
[3] Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2 https://doi.org/10.7287/peerj.preprints.3190v2

[4] https://www.facebook.com/DublinCityCouncil/posts/please-note-that-due-to-a-system-upgrade-dublinbikes-will-not-be-available-from-/10159750050654625/

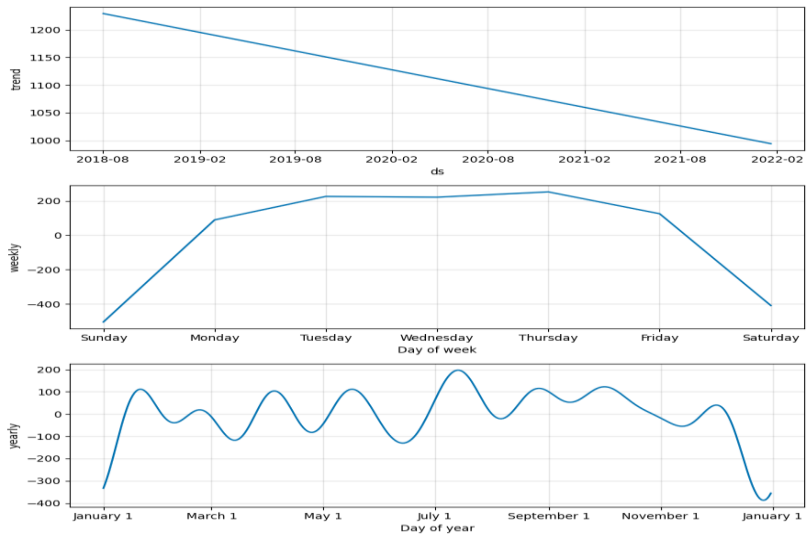
pandemic. The results show that the median daily bike usage dropped by 670 riders from before the pandemic to during. *This is a roughly 50% drop!*

Impact on Usage during the pandemic period

In order to investigate the impact of the pandemic prophet was trained on all datapoints up until the first recorded case of covid which was 29-02-2020. As mentioned, this is the day that the first covid case was detected and before then I found no real news that would correlate to a reduction in city bike usage as everyone was still working and traveling. Using Prophet, I forecasted 692 days until the day that all restrictions were lifted. The first graph represents the forecast using the prophet model and the following three graphs show the overall trend of the pre pandemic data.



This graph represents the forecast made by the Prophet model over the pandemic period.



As can be seen in the three graphs there was clearly a reducing trend of bike usage already before the pandemic. The weekly and yearly seasonality can be seen in the second and third graphs respectively. Clearly most users are using the bikes to commute as can be seen in the weekly seasonality as well as the strong cut off in the Christmas period. The Christmas cut off could be attributed to weather, but this dip really only occurs in December where temperatures are similar to November and January/February.

Based off the model the median daily usage is as following:

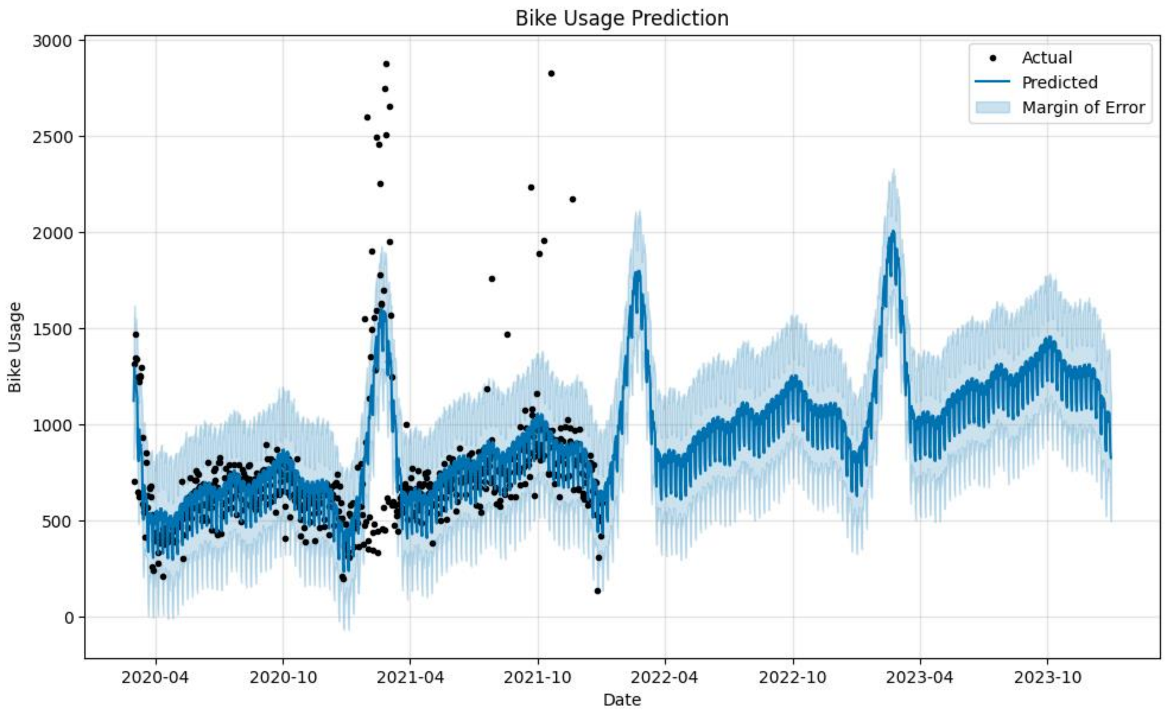
Forecasted median during covid: 1190.54, Pre covid forecast median: 1321.47, Median over the whole forecast period: 1236.84

As mentioned in the caption of the graph. There had been a reducing trend in overall bike usage even before the pandemic. This trend is exemplified in the forecast. Using the pre covid median of the recorded data there is a median daily usage of 1336, but the overall forecasted median is 1237. This is a drop of around 100 daily users and represents a ridership drop of about 7.8%. This is a significant drop that Dublin city council should investigate. When using the mean, as shown in the graph, there is a 200 rider drop from August 2018 to February 2022

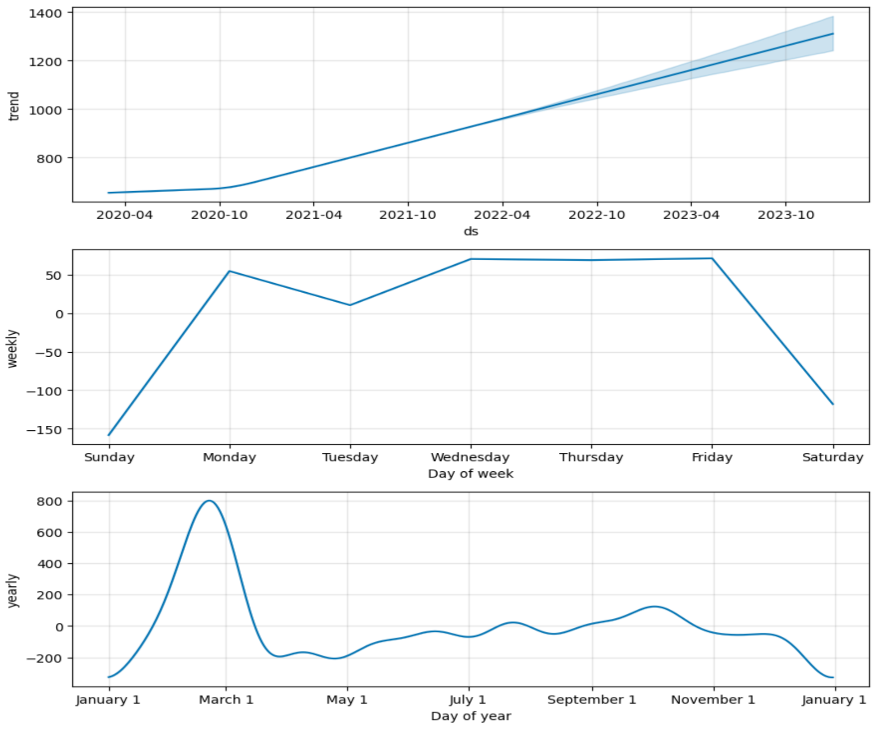
As for the pandemic, despite the predicted declining ridership numbers, it had a massive impact on ridership. Had the pandemic not happened Dublin bikes would have had an additional 515 additional daily riders.

Impact on Usage after the Pandemic Period

As in the previous section the dates of the pandemic remain the same: 29-02-2020 until 21-01-2022. Since there is no data that can be used to model past the end of the pandemic only the trends towards the end of the pandemic where restrictions were being eased can be observed. Using the usage data starting from 29-02-2020 until the end of the dataset, 31-12-2021, Prophet was used to forecast 730 days into the future until 01-01-2024. As before, the first graph represents the forecast until 01-01-2024 and the next three graphs represent the overall trend and weekly/yearly seasonality.



This graph represents the forecast of bike usage after the pandemic. As mentioned earlier there are some extreme outliers that clearly have an effect on the forecast, but I did not want to remove them.



As above these three graphs represent the trend of the forecast. Starting in October of 2020 there is a linear growth of usage. This couples with the fact that there were still people commuting during the pandemic which shows a positive post pandemic recovery for Dublin bike. I have included the yearly graph, but it is somewhat useless due to the skewing of the data around the January/February window.

Based on the forecast these are the median ridership numbers:

Forecast median: 895.78, post covid forecast median: 1090.11, October 2023 forecast median: 1333.91,

Obviously throughout the entire forecast period the median ridership will be below pre-pandemic levels. The reason I chose to show the median ridership for October 2023 is that it can be compared to the levels in October of 2019. The median ridership for October 2019 was 1318. This shows that over the forecasted month of October 2023 there was basically equal ridership as pre pandemic levels. This gives a roughly two-year time frame for ridership to reach normal levels once again. The reason I chose October is that there is a slight peak in the recorded data around then and this is also shown in the forecast post covid.

Model Performance

Cross validation was used to evaluate the performance of the model. The initial training size for both models was 365 days, period was 30 days, and the horizon was 30 days as well. This means that cross-validation was performed on the Prophet model by training the model on the first 365 days of data, then making and evaluating predictions for the next 30 days and repeating this process every 30 days. Looking at the mean absolute error, both models had an error of around 110 which I would consider quite accurate and am happy with. The full table of the cross-validation results is in the appendix.

Part Two

Question I

An ROC (receiver operating characteristic) curve is a visual tool that can be used to evaluate the performance of binary classifiers. What it does is plot the true positive rate, or sensitivity, against the false positive rate and multiple thresholds that can be set. You can use the ROC curve to compare the curve of a baseline to one of the classifier you are testing and then can visually assess how much better the classifier is compared to baseline. The area under the ROC curve can provide a metric for summarizing the performance. AN ROC curve would be used over a accuracy metric because it is less affected by class imbalance and since one can see the classifiers performance across different threshold levels

Question II

As per its name a linear regression should be used in a situation where there is a linear correlation between the data. One example would be when attempting to predict the value of an exponentially growing population. In this case linear regression would fail. In this case using a logistic regression would make sense as it matches the data better. Another example where linear regression can fail is when there are extreme outliers. If one is, for example, attempting to predict the values of homes there might be some mansions or houses in a desirable location that have a higher price. These outliers could disproportionately influence the model and lead to inaccuracies. In this case one could remove the outliers to improve the accuracy of the model.

Question III

In SVMs a kernel is a function used to transform non-linearly separable data into higher dimensional space where it becomes linearly separable where it is then possible to classify them effectively. A few types of kernels for SVMs are linear, polynomial, and radial basis. For CNNs a kernel is a small matrix that is used to perform convolution operations on the input data like an image which is then used to extract features from the input. Different types of CNN kernels can extract different types of features from the input. SVM kernels are used for transforming and classifying data and CNN kernels for feature extraction usually in deep learning.

Question IV

The idea behind resampling the dataset multiple times is to be able to utilize the entire dataset for training and testing which allows for evaluating the model’s general performance better. For example, If I had a dataset of 100 points and I use a 5-fold cross-validation I split this dataset into 5 parts. For each iteration of the cross-validation, one of these parts is used as the test data and the rest for training. These multiple cycles can then be used to derive the model’s generalized accuracy. The cross-validation also can mitigate the risk of bias that can occur if the dataset were to be only split once. Cross-validation can be especially useful when one has a small dataset as one can use the dataset more efficiently then. Somewhere it would not be appropriate to use cross-validation would be in time series data. This is because the temporal order of the datapoints is important and randomly splitting up the dataset would disrupt the order.

Appendix

Cross Validation First Model

horizon	mse	rmse	mae	mape	mdape	smape	coverage
3 days	44841.01	211.757	132.5995	0.158792	0.074711	0.135259	0.849206
4 days	34875.54	186.7499	118.0059	0.153618	0.067082	0.128794	0.849206
5 days	58182.5	241.2105	144.9008	0.151196	0.060531	0.158661	0.796296
6 days	43923.25	209.5787	121.253	0.118355	0.051188	0.139644	0.857143
7 days	64985	254.9216	160.2823	0.170943	0.057458	0.176969	0.769841
8 days	46712.65	216.1311	142.1531	0.153473	0.079269	0.148118	0.833333
9 days	92972.39	304.9137	193.1506	0.362415	0.112661	0.213808	0.801587
10 days	71809.54	267.973	151.6371	0.299234	0.086416	0.160366	0.888889
11 days	69707.76	264.0223	143.1068	0.302993	0.068751	0.16144	0.888889
12 days	48777.57	220.8565	124.4771	0.09551	0.043677	0.101377	0.888889
13 days	38195.67	195.4371	115.6514	0.094635	0.075186	0.096996	0.944444
14 days	33083.33	181.8882	110.898	0.092517	0.072911	0.095776	0.953704
15 days	9355.891	96.72585	82.60332	0.082539	0.053273	0.083387	0.944444
16 days	10155.4	100.774	83.09267	0.081242	0.058966	0.084021	0.888889
17 days	10562.68	102.7749	83.36925	0.085823	0.059801	0.087371	0.896825
18 days	8692.716	93.23474	75.8216	0.078913	0.059894	0.079309	0.936508
19 days	8615.094	92.81753	79.9671	0.080254	0.0704	0.079587	0.981481
20 days	7911.352	88.94578	77.13832	0.070472	0.069876	0.070159	1
21 days	11583.45	107.6264	92.16983	0.083812	0.07831	0.080994	0.944444
22 days	12424.64	111.4659	92.85964	0.093065	0.069876	0.091769	0.944444
23 days	10581.56	102.8667	82.46753	0.088335	0.057887	0.086406	0.944444
24 days	14254.09	119.3905	80.29199	0.092752	0.050373	0.085392	0.944444
25 days	45351.7	212.9594	111.984	0.247417	0.049585	0.135695	0.888889
26 days	111476.8	333.8815	170.2392	0.270859	0.055253	0.175595	0.849206
27 days	131648.4	362.8338	199.9767	0.317738	0.068167	0.205704	0.833333
28 days	128112.6	357.9282	207.8277	0.235238	0.079738	0.191065	0.825397
29 days	66902.66	258.6555	151.2347	0.183021	0.067874	0.14157	0.87963
30 days	31576.31	177.6972	110.8556	0.131502	0.065732	0.106959	0.898148

Cross Validation Second Model

horizon	mse	rmse	mae	mape	mdape	smape	coverage
3 days	107323.2	327.6022	187.1131	0.250744	0.125872	0.208447	0.868966
4 days	75614.8	274.9815	153.1554	0.25355	0.116923	0.183692	0.9
5 days	75240.76	274.3005	159.4186	0.265149	0.115081	0.198846	0.9
6 days	79247.36	281.5091	165.7829	0.280746	0.110658	0.213919	0.9
7 days	60752.93	246.4811	148.6664	0.242801	0.085404	0.195282	0.9
8 days	110529	332.459	181.5503	0.24566	0.085404	0.218665	0.865517
9 days	103277.4	321.3679	177.2623	0.231752	0.102462	0.209383	0.862069
10 days	99497.28	315.4319	178.1417	0.227755	0.120624	0.211277	0.862069
11 days	36248.68	190.3909	132.0787	0.193304	0.112343	0.169402	0.896552
12 days	33932.79	184.2086	129.5213	0.183518	0.120101	0.173094	0.865517
13 days	35692.07	188.9235	130.2537	0.189082	0.112343	0.185502	0.865517
14 days	43342.91	208.1896	136.2209	0.18071	0.120101	0.187453	0.868966
15 days	40675.26	201.6811	145.7107	0.188102	0.166516	0.19716	0.9
16 days	35775.77	189.1448	145.3847	0.184657	0.15546	0.190041	0.934483
17 days	20988.02	144.8724	124.7357	0.170741	0.153162	0.172357	0.968966
18 days	279461.4	528.6411	251.4044	0.1904	0.111311	0.223512	0.896552
19 days	283040.9	532.0159	249.354	0.184718	0.076854	0.227055	0.862069
20 days	284575.5	533.4562	251.0542	0.189535	0.076854	0.239485	0.862069
21 days	26665.22	163.2949	115.6537	0.151711	0.094355	0.175638	0.931034
22 days	52361.4	228.8261	144.1287	0.166713	0.124839	0.191619	0.931034
23 days	51735.77	227.455	149.2709	0.17673	0.148265	0.190963	0.934483
24 days	49258.13	221.9417	145.9852	0.277826	0.155263	0.210385	0.934483
25 days	21151.05	145.434	110.2301	0.267737	0.114466	0.185104	0.965517
26 days	21335.24	146.0659	106.0778	0.258047	0.111279	0.183036	0.968966
27 days	16340.27	127.8291	92.4109	0.15762	0.0844	0.154502	1
28 days	19466.75	139.5233	107.173	0.164857	0.102969	0.167724	1
29 days	22162.2	148.8697	117.7591	0.187933	0.106471	0.184269	0.965517
30 days	19719.08	140.4246	112.3507	0.178483	0.110105	0.167508	0.965517

Code Used

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import os
```

```
In [ ]: files = os.listdir('./Data/')
```

This loop processes each file in the 'files' list. For each file, it reads the data into a DataFrame and performs several transformations:

- Converts the 'TIME' column to datetime format.
- Drops unnecessary columns.
- Groups the DataFrame by 'TIME' and sums the other columns.
- Calculates bike usage as the negative difference in 'AVAILABLE BIKES', replacing negative values with 0 and filling any missing values with 0.
- Resamples the DataFrame to daily frequency, sums the other columns, and appends the result to the 'df_mean' DataFrame. The end result is a DataFrame ('df_mean') that contains daily bike usage data, aggregated from all files in the 'files' list.

```
In [ ]: df_mean = pd.DataFrame(columns = ['TIME','BIKE STANDS','AVAILABLE BIKE STAND
        for file in files:
            df = pd.read_csv(f"./Data/{file}")
            df['TIME'] = pd.to_datetime(df['TIME'])
            df.drop(['LAST UPDATED','NAME','STATUS','STATION ID',
                     'ADDRESS','LATITUDE','LONGITUDE'], axis=1, inplace=True)
            df = df.sort_values(by=['TIME'])

            df = df.groupby('TIME').sum().reset_index()
            df['BIKE USAGE'] =  -df['AVAILABLE BIKES'].diff()
            df['BIKE USAGE'] = df['BIKE USAGE'].apply(lambda x: 0 if x < 0 else x)
            df['BIKE USAGE'].fillna(0, inplace=True)
            df_mean = pd.concat([df_mean,df.resample('D',on='TIME').sum().reset_inde
                                 ,ignore_index=True)
```

```
In [ ]: df_mean.to_csv('UsageByDayClean.csv', index=False,encoding='utf-8')
```

this next cell was used to create different files for daily, weekly, monthly, and quarterly aggregations but only the daily aggregations really ended up being used. I simply changed the resample to get those different files and below it is set to create the daily aggregation file.

```
In [ ]: df = pd.read_csv('./UsageByDayClean.csv')
        df['TIME'] = pd.to_datetime(df['TIME'])
        # here the data is resampled to days and the sum of the bike usage is taken
        df = df.resample('D',on='TIME').sum().reset_index()
        # set all days with no recorded data to NA to then drop from df
        df = df.replace(0,pd.NA)
df = df.dropna()
df.set_index('TIME',inplace=True)
```

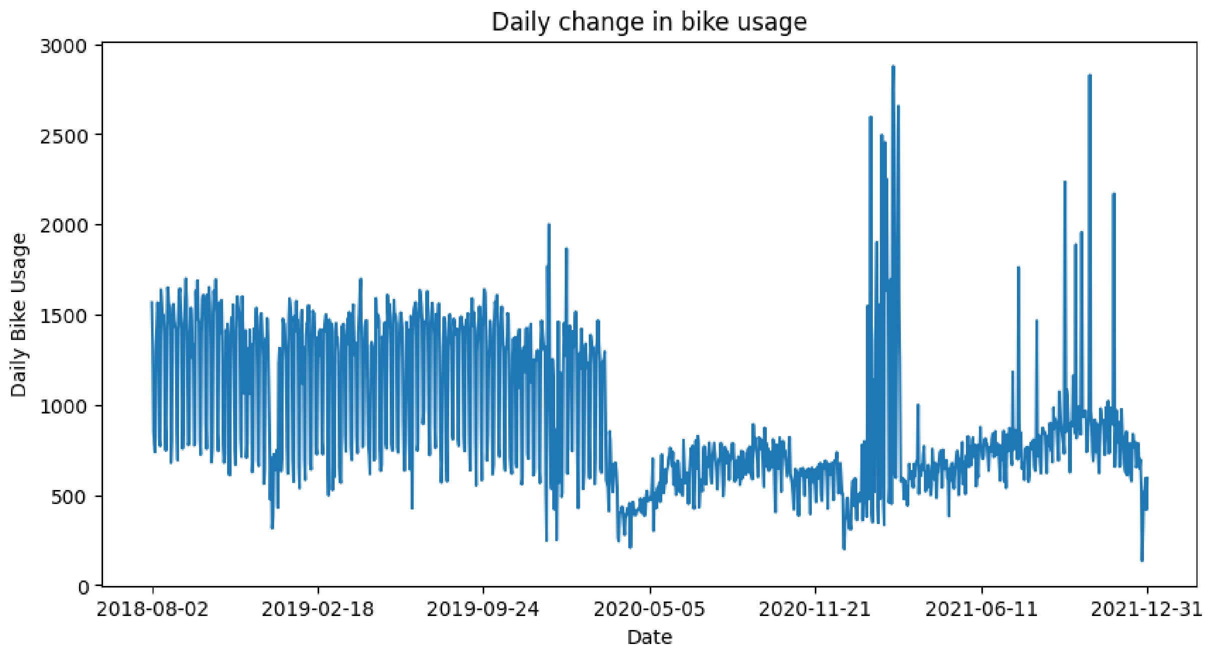
this next block was used to calculate the average bike usage per week/month/quarter this was because there were missing days in the data

```
In [ ]: count_days_recorded = df.resample('D',on='TIME').count()
        count_days_recorded.replace(0,1,inplace=True)
        count_days_recorded.to_csv('dayCount.csv')
        # avg bike usage column was created to have the total usage per period and t
        # this is to normalize the periods with missing days of recorded data
        df['AVG BIKE USAGE'] = df['BIKE USAGE'] / count_days_recorded['BIKE USAGE']
```

```
In [ ]: df.to_csv('monthlyAvg.csv')
```

```
In [2]: df = pd.read_csv('./UsageByDayClean.csv')
```

```
In [10]: # simple plot to show the change in bike usage over time
         df = df.groupby('TIME').sum()
         df['BIKE USAGE'].plot(kind='line')
         # set size of plot 10 x 20
         plt.rcParams["figure.figsize"] = (5,5)
         plt.title('Daily change in bike usage')
         plt.xlabel('Date')
         plt.ylabel('Daily Bike Usage')
         plt.show()
```

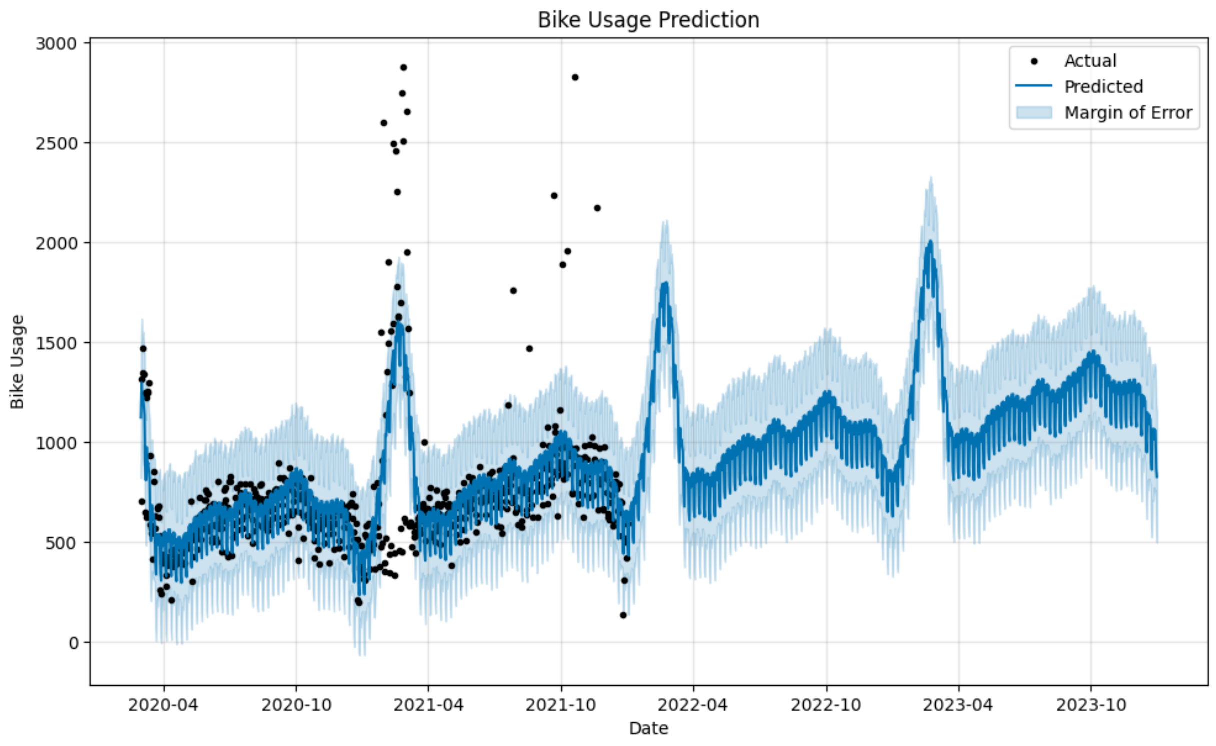


In [18]:
```
from prophet import Prophet
import pandas as pd
from matplotlib import pyplot as plt
from prophet.diagnostics import cross_validation, performance_metrics
```

In [2]:
```
data = pd.read_csv('./UsageByDayClean.csv',usecols=['TIME','BIKE USAGE'])
data['TIME'] = pd.to_datetime(data['TIME'])
data.rename(columns={'TIME':'ds','BIKE USAGE':'y'},inplace=True)
pre_covid_date = pd.to_datetime('2020-02-29')
post_covid_date = pd.to_datetime('2022-01-21')
post_covid_data = data[data['ds'] > pre_covid_date]
```

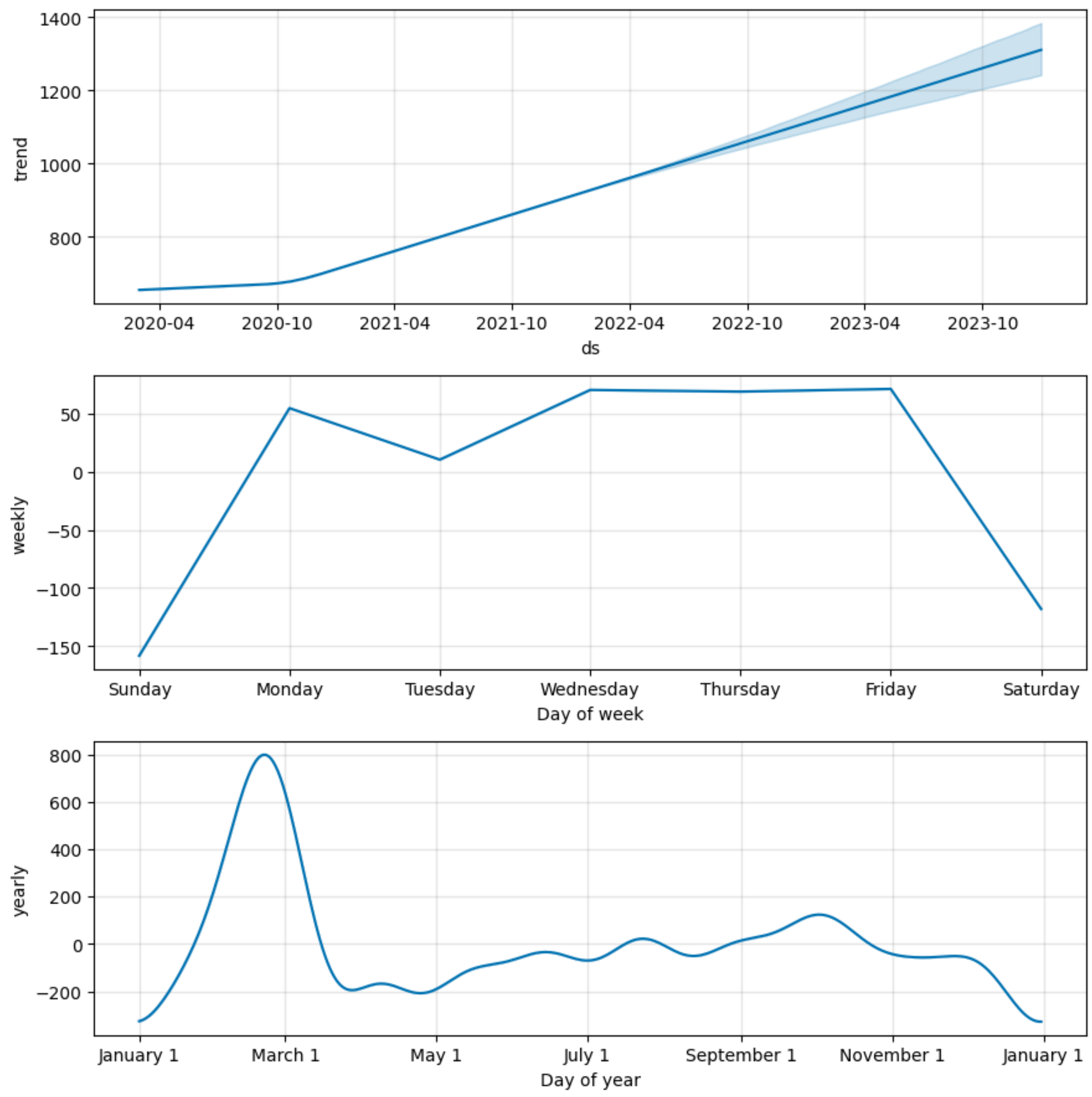
In [23]:
```
model = Prophet(yearly_seasonality=True,weekly_seasonality=True)
model.fit(post_covid_data)
future = model.make_future_dataframe(periods=730)
forecast = model.predict(future)
fig = model.plot(forecast)
plt.title('Bike Usage Prediction')
plt.xlabel('Date')
plt.ylabel('Bike Usage')
plt.legend(['Actual','Predicted','Margin of Error'])
plt.show()
```

```
17:54:54 - cmdstanpy - INFO - Chain [1] start processing
17:54:54 - cmdstanpy - INFO - Chain [1] done processing
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:72: FutureWarning: The behavior of DatetimeProperties.to_pydatet
ime is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:73: FutureWarning: The behavior of DatetimeProperties.to_pydatet
ime is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```



```
In [9]: fig = model.plot_components(forecast)
        plt.show()
```

```
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:228: FutureWarning: The behavior of DatetimeProperties.to_pydate
time is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:351: FutureWarning: The behavior of DatetimeProperties.to_pydate
time is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  df_y['ds'].dt.to_pydatetime(), seas[name], ls='-', c='#0072B2')
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:354: FutureWarning: The behavior of DatetimeProperties.to_pydate
time is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  df_y['ds'].dt.to_pydatetime(), seas[name + '_lower'],
```



```
In [17]: forecast_median = forecast['yhat'].median()
         # median of forecasted data after post covid date
         post_covid_forecast_median = forecast[forecast['ds'] > post_covid_date]['yha
         # median of forecasted data during october 2023
         oct_2023_forecast_median = forecast[(forecast['ds'] > pd.to_datetime('2023-1
         # median of actual data during october 2019
         oct_2019_forecast_median = data[(data['ds'] > pd.to_datetime('2019-10-01'))
         print(f"forecast median: {forecast_median.__round__(2)}")
         print(f"post covid forecast median: {post_covid_forecast_median.__round__(2)
         print(f"october 2023 forecast median: {oct_2023_forecast_median.__round__(2)
         print(f"october 2019 median: {oct_2019_forecast_median.__round__(2)}")
```

```
forecast median: 895.78
post covid forecast median: 1090.11
october 2023 forecast median: 1333.91
october 2019 median: 1318.0
```

```
In [20]: # evaluating the accuracy of the model
         df_cv = cross_validation(model,horizon='30 days',period='30 days',initial='3
df_p = performance_metrics(df_cv)
```

```
Seasonality has period of 365.25 days which is larger than initial window. C
onsider increasing initial.
  0%|          | 0/10 [00:00<?, ?it/s]01:34:21 - cmdstanpy - INFO - Chain
[1] start processing
01:34:21 - cmdstanpy - INFO - Chain [1] done processing
 10%|█         | 1/10 [00:00<00:01,  5.26it/s]01:34:21 - cmdstanpy - INFO -
Chain [1] start processing
01:34:21 - cmdstanpy - INFO - Chain [1] done processing
 20%|██        | 2/10 [00:00<00:01,  5.29it/s]01:34:22 - cmdstanpy - INFO -
Chain [1] start processing
01:34:22 - cmdstanpy - INFO - Chain [1] done processing
 30%|███       | 3/10 [00:00<00:01,  5.06it/s]01:34:22 - cmdstanpy - INFO -
Chain [1] start processing
01:34:22 - cmdstanpy - INFO - Chain [1] done processing
 40%|████      | 4/10 [00:00<00:01,  4.92it/s]01:34:22 - cmdstanpy - INFO -
Chain [1] start processing
01:34:22 - cmdstanpy - INFO - Chain [1] done processing
 50%|█████     | 5/10 [00:01<00:01,  4.84it/s]01:34:22 - cmdstanpy - INFO -
Chain [1] start processing
01:34:22 - cmdstanpy - INFO - Chain [1] done processing
 60%|██████    | 6/10 [00:01<00:00,  4.71it/s]01:34:22 - cmdstanpy - INFO -
Chain [1] start processing
01:34:23 - cmdstanpy - INFO - Chain [1] done processing
 70%|███████   | 7/10 [00:01<00:00,  4.73it/s]01:34:23 - cmdstanpy - INFO -
Chain [1] start processing
01:34:23 - cmdstanpy - INFO - Chain [1] done processing
 80%|████████  | 8/10 [00:01<00:00,  4.63it/s]01:34:23 - cmdstanpy - INFO -
Chain [1] start processing
01:34:23 - cmdstanpy - INFO - Chain [1] done processing
 90%|█████████ | 9/10 [00:01<00:00,  4.60it/s]01:34:23 - cmdstanpy - INFO -
Chain [1] start processing
01:34:23 - cmdstanpy - INFO - Chain [1] done processing
100%|██████████| 10/10 [00:02<00:00,  4.74it/s]
```

In [21]:
```
df_p.iloc[20:]
```

Out[21]:

	horizon	mse	rmse	mae	mape	mdape	smap
20	23 days	51735.767549	227.454979	149.270928	0.176730	0.148265	0.19096
21	24 days	49258.128724	221.941724	145.985229	0.277826	0.155263	0.21038
22	25 days	21151.045107	145.433989	110.230059	0.267737	0.114466	0.18510
23	26 days	21335.239663	146.065874	106.077817	0.258047	0.111279	0.18303
24	27 days	16340.268798	127.829061	92.410899	0.157620	0.084400	0.15450
25	28 days	19466.752193	139.523303	107.172996	0.164857	0.102969	0.16772
26	29 days	22162.199393	148.869740	117.759119	0.187933	0.106471	0.18426
27	30 days	19719.075311	140.424625	112.350653	0.178483	0.110105	0.16750

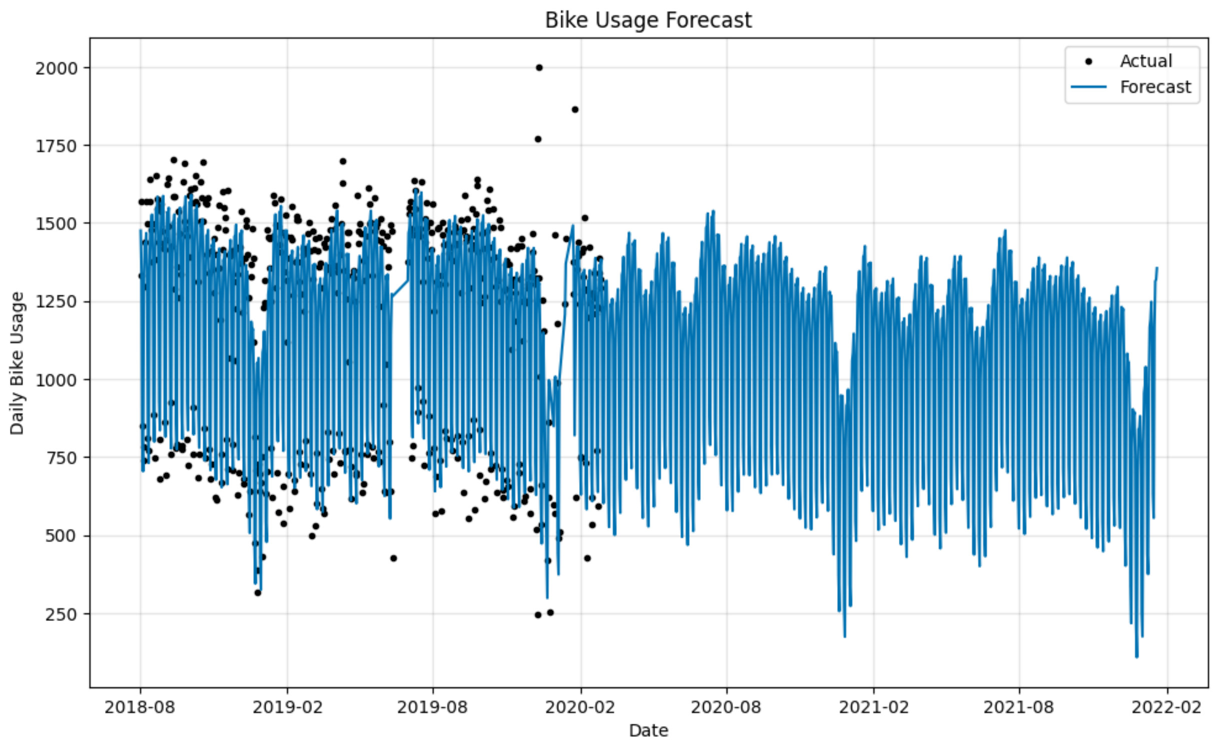
In [22]:
```
df_p.to_csv('./performance post covid.csv',index=False)
```

In [33]:
```python
from prophet import Prophet
import pandas as pd
from matplotlib import pyplot as plt
from prophet.diagnostics import cross_validation, performance_metrics
```

In [20]:
```python
data = pd.read_csv('./UsageByDayClean.csv',usecols=['TIME','BIKE USAGE'])
data['TIME'] = pd.to_datetime(data['TIME'])
data.rename(columns={'TIME':'ds','BIKE USAGE':'y'},inplace=True)

pre_covid_date = pd.to_datetime('2020-02-29')
post_covid_date = pd.to_datetime('2022-01-21')
# remove all data after 2020-03-27
data_pre_covid = data[data['ds'] < pre_covid_date]
data_in_covid = data[data['ds'] >= pre_covid_date]
```

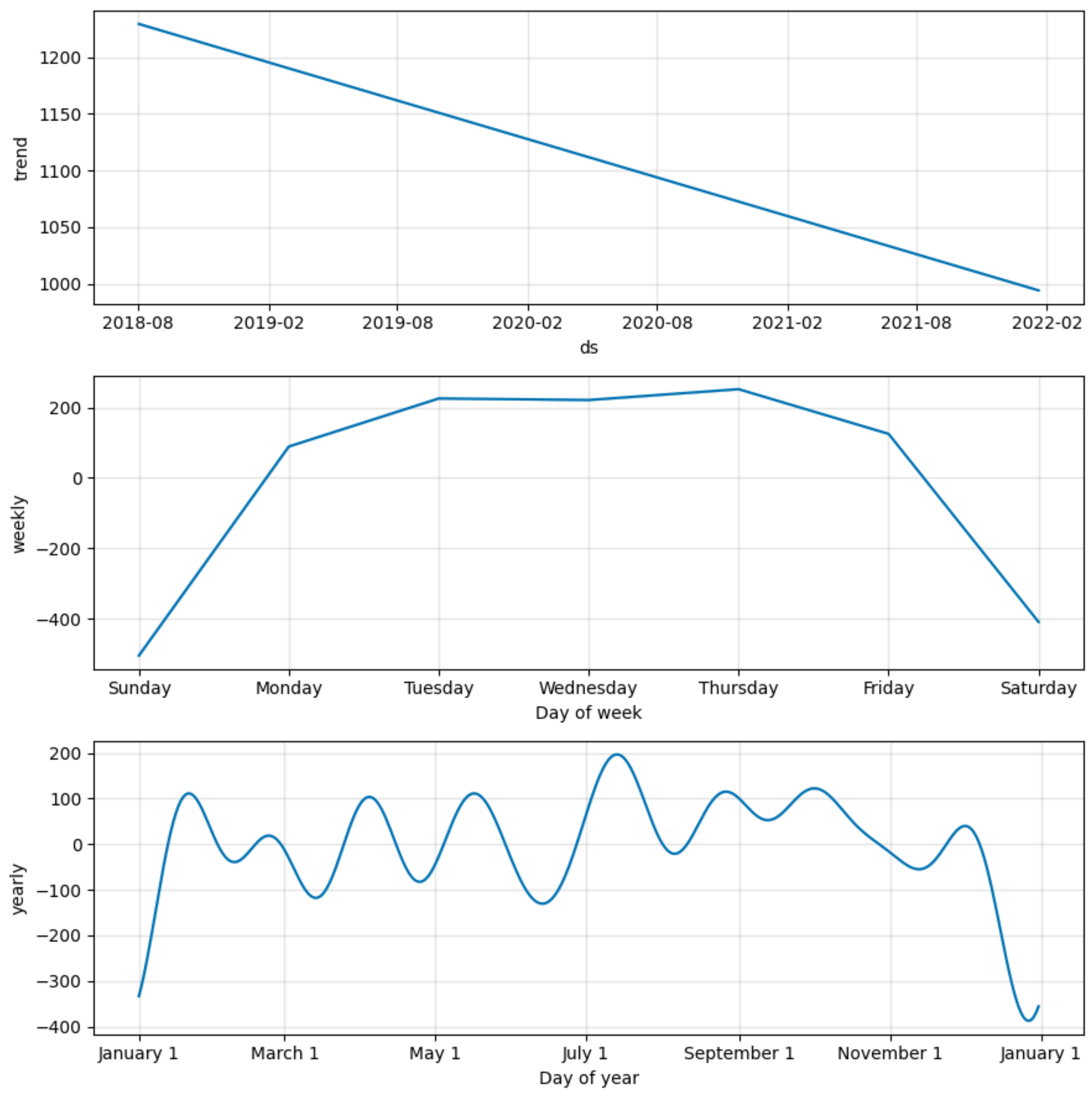
In [27]:
```python
length_of_covid = (post_covid_date - pre_covid_date).days
model = Prophet(yearly_seasonality=True,weekly_seasonality=True,daily_seasor

model.fit(data_pre_covid)
future = model.make_future_dataframe(periods=length_of_covid)
forecast = model.predict(future)
fig = model.plot(forecast,uncertainty=False)
# label the plot
plt.title('Bike Usage Forecast')
plt.xlabel('Date')
plt.ylabel('Daily Bike Usage')
plt.legend(['Actual','Forecast'])
plt.show()
```

```
23:43:08 - cmdstanpy - INFO - Chain [1] start processing
23:43:08 - cmdstanpy - INFO - Chain [1] done processing
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:72: FutureWarning: The behavior of DatetimeProperties.to_pydatet
ime is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:73: FutureWarning: The behavior of DatetimeProperties.to_pydatet
ime is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```



In [24]:
```python
# plot the components of the model
fig = model.plot_components(forecast)
plt.show()
```

```
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:228: FutureWarning: The behavior of DatetimeProperties.to_pydate
time is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:351: FutureWarning: The behavior of DatetimeProperties.to_pydate
time is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  df_y['ds'].dt.to_pydatetime(), seas[name], ls='-', c='#0072B2')
c:\Users\User\Documents\Year 4\MLFinalAssignment\venv\lib\site-packages\prop
het\plot.py:354: FutureWarning: The behavior of DatetimeProperties.to_pydate
time is deprecated, in a future version this will return a Series containing
python datetime objects instead of an ndarray. To retain the old behavior, c
all `np.array` on the result
  df_y['ds'].dt.to_pydatetime(), seas[name + '_lower'],
```



```
In [25]: total_median = data['y'].median()
         pre_covid_median= data_pre_covid['y'].median()
         during_covid_median= data_in_covid['y'].median()
         total_forecast_median= forecast['yhat'].median()
         forecast_in_covid_median = forecast[forecast['ds'] >= pre_covid_date]['yhat'
         pre_covid_forecast_median = forecast[forecast['ds'] < pre_covid_date]['yhat'
         print('total_median: ',total_median)
         print('pre_covid_median: ',pre_covid_median)
         print('during_covid_median: ',during_covid_median)
         print('total forecast_median: ',total_forecast_median.__round__(2))
         print('forecast_in_covid_median: ',forecast_in_covid_median.__round__(2))
         print('Pre covid forecast median: ',pre_covid_forecast_median.__round__(2))
```

```
total_median:  766.0
pre_covid_median:  1336.5
during_covid_median:  675.0
total forecast_median:  1236.84
forecast_in_covid_median:  1190.54
Pre covid forecast median:  1321.47
```

In [43]:
```
# evaluating the accuracy of the model
cv_params = {'initial': '90 days','period': '30 days','horizon': '30 days'}
df_cv = cross_validation(model,horizon='30 days',period='30 days',initial='3
df_p = performance_metrics(df_cv)
```

```
Seasonality has period of 365.25 days which is larger than initial window. C
onsider increasing initial.
  0%|          | 0/7 [00:00<?, ?it/s]01:34:50 - cmdstanpy - INFO - Chain [1]
start processing
01:34:50 - cmdstanpy - INFO - Chain [1] done processing
 14%|█         | 1/7 [00:00<00:01,  5.03it/s]01:34:50 - cmdstanpy - INFO - C
hain [1] start processing
01:34:50 - cmdstanpy - INFO - Chain [1] done processing
 29%|██        | 2/7 [00:00<00:01,  4.88it/s]01:34:50 - cmdstanpy - INFO - C
hain [1] start processing
01:34:50 - cmdstanpy - INFO - Chain [1] done processing
 43%|███       | 3/7 [00:00<00:00,  4.68it/s]01:34:51 - cmdstanpy - INFO - C
hain [1] start processing
01:34:51 - cmdstanpy - INFO - Chain [1] done processing
 57%|████      | 4/7 [00:00<00:00,  4.78it/s]01:34:51 - cmdstanpy - INFO - C
hain [1] start processing
01:34:51 - cmdstanpy - INFO - Chain [1] done processing
 71%|█████     | 5/7 [00:01<00:00,  4.75it/s]01:34:51 - cmdstanpy - INFO - C
hain [1] start processing
01:34:51 - cmdstanpy - INFO - Chain [1] done processing
 86%|██████    | 6/7 [00:01<00:00,  4.67it/s]01:34:51 - cmdstanpy - INFO - C
hain [1] start processing
01:34:51 - cmdstanpy - INFO - Chain [1] done processing
100%|███████| 7/7 [00:01<00:00,  4.73it/s]
```

In [44]:
```
df_p.iloc[20:]
```

Out[44]:

	horizon	mse	rmse	mae	mape	mdape	sma
20	23 days	10581.561144	102.866715	82.467528	0.088335	0.057887	0.0864
21	24 days	14254.091412	119.390500	80.291992	0.092752	0.050373	0.0853
22	25 days	45351.703918	212.959395	111.983958	0.247417	0.049585	0.1356
23	26 days	111476.845703	333.881485	170.239167	0.270859	0.055253	0.1755
24	27 days	131648.381564	362.833821	199.976719	0.317738	0.068167	0.2057
25	28 days	128112.602251	357.928208	207.827711	0.235238	0.079738	0.1910
26	29 days	66902.659231	258.655484	151.234671	0.183021	0.067874	0.1415
27	30 days	31576.309555	177.697241	110.855557	0.131502	0.065732	0.1069

In [45]:
```
df_p.to_csv('./performance no covid.csv',index=False)
```

In [24]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
```

In [25]:
```
data = pd.read_csv('./UsageByDayClean.csv')
data['TIME'] = pd.to_datetime(data['TIME'])
data.sort_values(by=['TIME'], inplace=True)
data.set_index('TIME', inplace=True)
```

In [26]:
```
data.head()
data.drop(['BIKE STANDS','AVAILABLE BIKE STANDS','AVAILABLE BIKES'], axis=1,
```

In [27]:
```
data['MONTH'] = data.index.month
# data['YEAR'] = data.index.year
data['WEEK DAY'] = data.index.weekday
# data['DAY']= data.index.day
```

In [28]:
```
pre_covid_date = pd.to_datetime('2020-03-27')
post_covid_date = pd.to_datetime('2022-01-21')
```

In [29]:
```
data.columns
```

Out[29]:
```
Index(['BIKE USAGE', 'MONTH', 'WEEK DAY'], dtype='object')
```

In [30]:
```
#variable x which is all the data before the covid date excluding the bike u
X = data.loc[data.index < pre_covid_date].drop('BIKE USAGE', axis=1)
X_COVID = data.loc[data.index >= pre_covid_date].drop('BIKE USAGE', axis=1)
#variable y which is the bike usage column before the covid date (target)
# y = data.loc[data.index < pre_covid_date].drop(['MONTH','YEAR','WEEK DAY',
y = data.loc[data.index < pre_covid_date].drop(['MONTH','WEEK DAY'], axis=1)

# split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, shu
```

In [31]:
```
# finding the optimal alpha value for the ridge regression model
ridge = Ridge()
ridge.fit(X_train, Y_train)
alpha_values = {'alpha': np.linspace(.001,1000,100)}
ridge_regressor = GridSearchCV(ridge, alpha_values, scoring='neg_mean_square
ridge_regressor.fit(X_train, Y_train)
print(ridge_regressor.best_estimator_)
# best alpha found is 6.5  25.3
```

```
Ridge(alpha=10.102)
```

In [32]:
```
# training ridge regression model with alpha of 153
model = Ridge(alpha=10)
model.fit(X_train, Y_train)
```

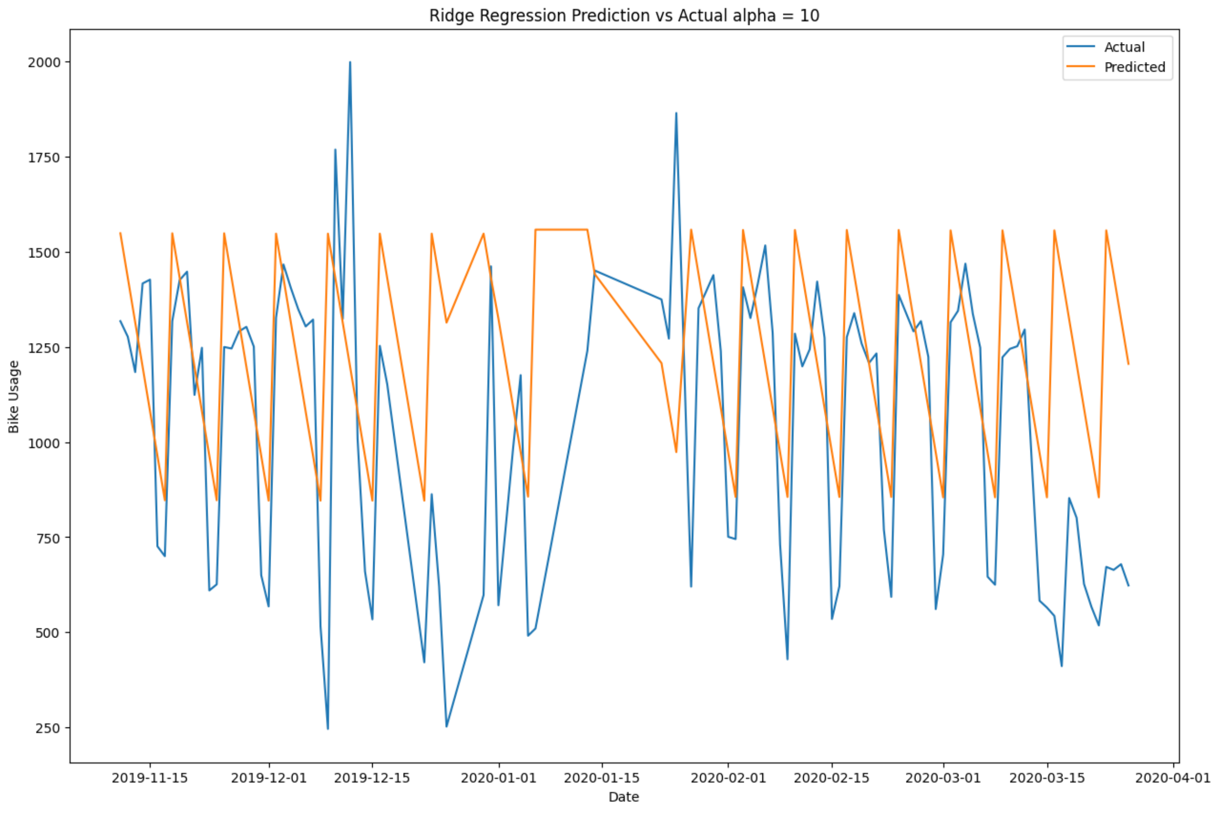
```
Out[32]: ▾    Ridge
         Ridge(alpha=10)
```

```
In [33]: model.coef_
```

```
Out[33]:  array([[  -0.96098653, -116.98244477]])
```

```
In [34]: # evaluation the model using the testing data
         y_pred = model.predict(X_test)
         mse = mean_squared_error(Y_test, y_pred)
         print(f"Mean Squared Error: {mse}")
         mae = mean_absolute_error(Y_test, y_pred)
         print(f"Mean Absolute Error: {mae}")

         # plotting the predicted values against the actual values with the timestamp
         plt.figure(figsize=(15,10))
         plt.plot(Y_test.index, Y_test, label='Actual')
         plt.plot(Y_test.index, y_pred, label='Predicted')
         plt.title('Ridge Regression Prediction vs Actual alpha = 10')
         plt.xlabel('Date')
         plt.ylabel('Bike Usage')
         plt.legend()
         plt.show()
```

```
Mean Squared Error: 163729.22763473366
Mean Absolute Error: 302.6439636849169
```


Ridge Regression Prediction vs Actual alpha = 10
Actual
Predicted
Bike Usage
Date