Matthew Werner

Canisius College: DAT 512

Project 3: Times Series Forcasting

In [1]:
```
import pandas as pd
import numpy as np
import requests
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

from prophet import Prophet
import matplotlib.pyplot as plt

%matplotlib inline

plt.rcParams['figure.figsize']=(20,10)
plt.style.use('ggplot')
```

In [2]:
```
# Making sure the connection works to 311 dataset
# Checking to see if the data has less than 1000 rows
uri = 'https://data.buffalony.gov/resource/whkc-e5vr.json'
r = requests.get(uri)
print('Status code ',r.status_code)
print('Number of rows returned ',len(r.json()))
print('Endoced URI with params ',r.url)
```

```
Status code  200
Number of rows returned  1000
Endoced URI with params  https://data.buffalony.gov/resource/whkc-e5vr.json
```

In [3]:
```
data311=pd.DataFrame(r.json())
print(data311.shape)
data311.head()
```

```
(1000, 48)
```

Out[3]:

	case_reference	open_date	closed_date	status	subject	reason	type	object_type
0	1001805272	2023-03-09T09:49:00.000	2023-03-09T10:02:00.000	Closed	Dept of Public Works	Engineering - Street Repairs	Paving (Req_Serv)	Stree
1	1001788769	2023-02-07T15:04:00.000	2023-02-14T10:35:00.000	Closed	Dept of Public Works	Sanitation	Totes Replace (Req_Serv)	Property
2	1001805334	2023-03-09T11:11:00.000	2023-03-13T17:33:00.000	Closed	Office of the Mayor	Citizen Services - Quick Response Teams	QRT Other Issue (Req_Serv)	Property
3	1001805712	2023-03-12T16:35:00.000	2023-03-27T07:15:00.000	Closed	DPIS	Housing	Housing Violations (Req_Serv)	Property
4	1001836852	2023-04-28T08:43:00.000	2023-04-28T09:23:00.000	Closed	Dept of Parking	Parking Violations Bureau	Parking Issues (Req_Serv)	Property

5 rows × 48 columns

In [4]:
```
# Pulling in 311 Service Requests API from Open Data Buffalo
# Setting limit parameter in order to extract all of the rows
# Date parameters set to limit data to end on 1/1/2023

params_dict = {
    '$where':'date_extract_y(open_date) >= 2018 and date_extract_y(open_date) < 2023',
    '$limit':1000000
}

uri = 'https://data.buffalony.gov/resource/whkc-e5vr.json'
r = requests.get(uri, params=params_dict)
print('Status code ',r.status_code)
print('Number of rows returned ',len(r.json()))
print('Endoced URI with params ',r.url)
```

```
Status code  200
Number of rows returned  406432
Endoced URI with params  https://data.buffalony.gov/resource/whkc-e5vr.json?%24where=
date_extract_y%28open_date%29+%3E%3D+2018+and+date_extract_y%28open_date%29+%3C+2023
&%24limit=1000000
```

In [5]:
```
# Creating a Pandas dataframe for 311 data
data311=pd.DataFrame(r.json())
print(data311.shape)
data311.head()
```

```
(406432, 33)
```

Out[5]:

	case_reference	open_date	closed_date	status	subject	reason	type	object_ty
0	1001756978	2022-12-27T12:02:00.000	2023-01-06T16:49:00.000	Closed	Dept of Public Works	Streets	Street Snow Plowing (Req_Serv)	Prope
1	1001353407	2021-02-23T10:29:00.000	2023-03-25T11:19:00.000	Closed	Dept of Public Works	Engineering - Traffic	Signal Out or Flashing (Req_Serv)	Prope
2	1001339938	2021-02-09T14:36:00.000	2023-01-02T17:20:00.000	Closed	Dept of Public Works	Engineering - Traffic	Sign Maintenance (Req_Serv)	Prope
3	1001759464	2022-12-29T14:44:00.000	2023-01-13T11:13:00.000	Closed	Dept of Public Works	Animal Shelter	Animals (Req_Serv)	Str
4	1001757568	2022-12-27T16:32:00.000	2023-01-03T11:08:00.000	Closed	Dept of Public Works	Sanitation	Totes Replace (Req_Serv)	Prope

5 rows × 33 columns

Data Cleaning and Preparation

Filtering useful columns into new dataframe

In [6]:
```
# Finding all column names
data311.columns
```

Out[6]:
```
Index(['case_reference', 'open_date', 'closed_date', 'status', 'subject',
       'reason', 'type', 'object_type', 'address_number', 'address_line_1',
       'city', 'state', 'zip_code', 'property_id', 'location', 'latitude',
       'longitude', 'council_district', 'police_district', 'census_tract',
       'census_block_group', 'census_block', 'neighborhood', 'x_coordinate',
       'y_coordinate', 'census_tract_2010', 'census_block_group_2010',
       'census_block_2010', 'tractce20', 'geoid20_tract', 'geoid20_blockgroup',
       'geoid20_block', 'address_line_2'],
      dtype='object')
```

In [7]:
```
# Selcting valuable columns
data311 = pd.DataFrame(data311[['case_reference', 'open_date', 'status', 'type', 'zip_
data311.head()
```

Out[7]:

	case_reference	open_date	status	type	zip_code	reason
0	1001756978	2022-12-27T12:02:00.000	Closed	Street Snow Plowing (Req_Serv)	14212	Streets
1	1001353407	2021-02-23T10:29:00.000	Closed	Signal Out or Flashing (Req_Serv)	14209	Engineering - Traffic
2	1001339938	2021-02-09T14:36:00.000	Closed	Sign Maintenance (Req_Serv)	14208	Engineering - Traffic
3	1001759464	2022-12-29T14:44:00.000	Closed	Animals (Req_Serv)	14220	Animal Shelter
4	1001757568	2022-12-27T16:32:00.000	Closed	Totes Replace (Req_Serv)	14211	Sanitation

In [8]:
```
# Dataframe shape
data311.shape
```

Out[8]:
```
(406432, 6)
```

In [9]:
```
# remove rows where zipcode = unknown
data311 = data311[data311['zip_code'] != 'UNKNOWN']
data311.head()
```

Out[9]:

	case_reference	open_date	status	type	zip_code	reason
0	1001756978	2022-12-27T12:02:00.000	Closed	Street Snow Plowing (Req_Serv)	14212	Streets
1	1001353407	2021-02-23T10:29:00.000	Closed	Signal Out or Flashing (Req_Serv)	14209	Engineering - Traffic
2	1001339938	2021-02-09T14:36:00.000	Closed	Sign Maintenance (Req_Serv)	14208	Engineering - Traffic
3	1001759464	2022-12-29T14:44:00.000	Closed	Animals (Req_Serv)	14220	Animal Shelter
4	1001757568	2022-12-27T16:32:00.000	Closed	Totes Replace (Req_Serv)	14211	Sanitation

In [10]:
```
# Show if there was a change in the number of rows in the filtered 311 dataset
# Number of rows decreased from original dataframe
data311.shape
```

Out[10]:
```
(378686, 6)
```

In [11]:
```
data311.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 378686 entries, 0 to 406431
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype 
---  ------           --------------   ----- 
 0   case_reference   378686 non-null  object
 1   open_date        378686 non-null  object
 2   status           378686 non-null  object
 3   type             378686 non-null  object
 4   zip_code         378686 non-null  object
 5   reason           378686 non-null  object
dtypes: object(6)
memory usage: 20.2+ MB
```

In [12]:
```
# Converting to datetime and extracting only year, month, and day.
data311['open_date'] = pd.to_datetime(data311['open_date']).dt.strftime('%Y-%m-%d')
```

In [13]:
```
# Checking to make sure it worked.
data311['open_date'].head()
```

Out[13]:
```
0    2022-12-27
1    2021-02-23
2    2021-02-09
3    2022-12-29
4    2022-12-27
Name: open_date, dtype: object
```

In [14]:
```
# Building a new dataframe with the days that a case was opened
# Counting the number of cases opened on this day
# Preparing for prophit; Rename count column as "y"
final311 = data311.groupby('open_date')['open_date'].count().reset_index(name="y")
```

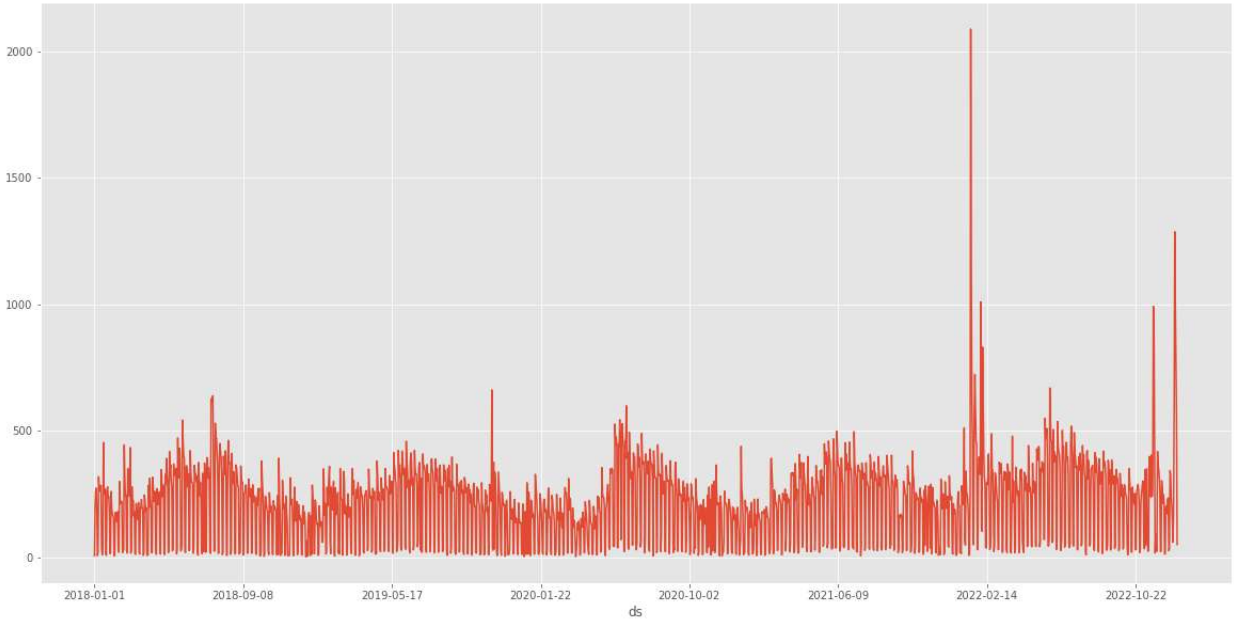
In [15]:
```
# Rename date column as "ds"
final311 = final311.rename(columns={'open_date':'ds'})
```

In [16]:
```
final311.head()
```

Out[16]:

	ds	y
0	2018-01-01	5
1	2018-01-02	206
2	2018-01-03	258
3	2018-01-04	273
4	2018-01-05	188

In [17]:
```
final311.set_index('ds').y.plot();
```



Initial Forecast

In [18]:
```
# Instantiate Model
model = Prophet()

# Fit Model
model.fit(final311)
```

```
13:36:51 - cmdstanpy - INFO - Chain [1] start processing
13:36:51 - cmdstanpy - INFO - Chain [1] done processing
```

Out[18]:
```
<prophet.forecaster.Prophet at 0x19c18416b20>
```

In [19]:
```
# Create future data frame
future = model.make_future_dataframe(periods= 365, freq = 'd')
future.tail()
```

Out[19]:

	ds
2181	2023-12-27
2182	2023-12-28
2183	2023-12-29
2184	2023-12-30
2185	2023-12-31

In [20]:
```
# To forecast this future data, we need to run it through Prophet's model.
# Add predictions to the forecast dataframe
forecast = model.predict(future)
```

In [21]:
```
# The resulting forecast dataframe contains quite a bit of data, but we really only ca
# First, let's look at the full dataframe:
forecast.tail().T
```

Out[21]:

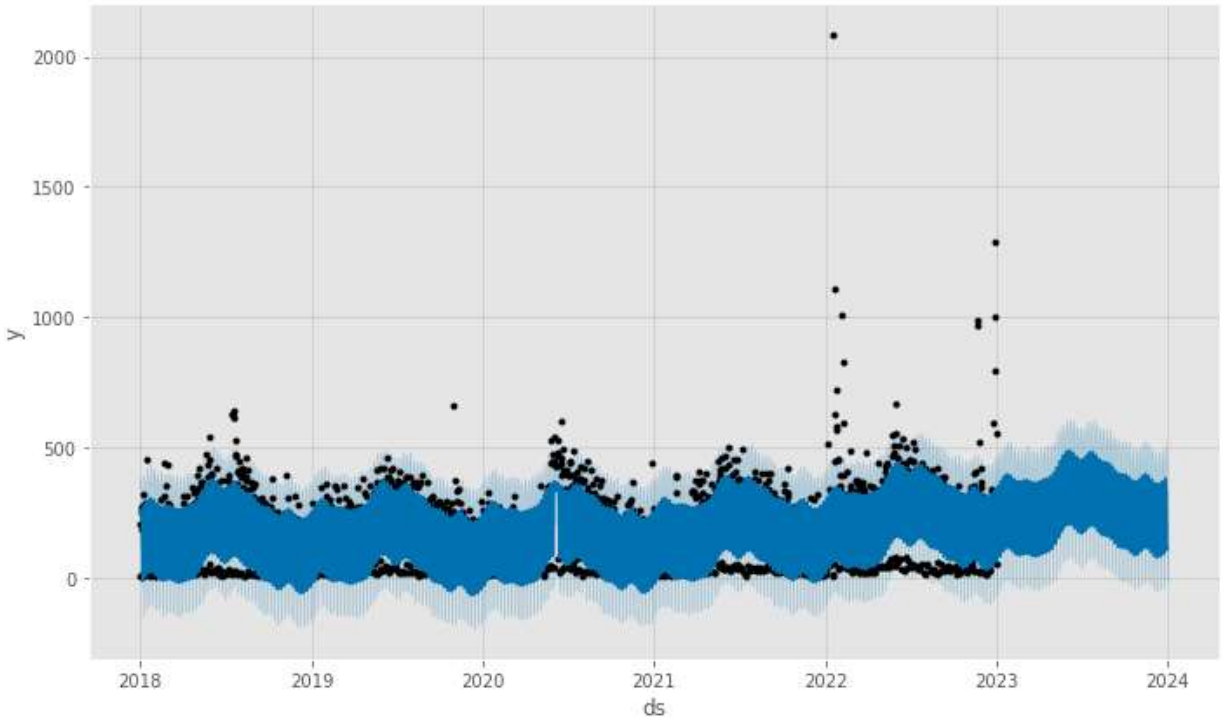
	2181	2182	2183	2184	2185
ds	2023-12-27 00:00:00	2023-12-28 00:00:00	2023-12-29 00:00:00	2023-12-30 00:00:00	2023-12-31 00:00:00
trend	330.804145	330.935655	331.067164	331.198673	331.330183
yhat_lower	244.151521	215.886879	206.03941	-15.194178	-15.393545
yhat_upper	480.909984	475.875981	458.872915	237.465034	243.623497
trend_lower	325.650346	325.755196	325.860047	325.974832	326.098397
trend_upper	336.054076	336.221836	336.386974	336.552001	336.719014
additive_terms	25.443383	14.941501	-3.921192	-226.487888	-218.145732
additive_terms_lower	25.443383	14.941501	-3.921192	-226.487888	-218.145732
additive_terms_upper	25.443383	14.941501	-3.921192	-226.487888	-218.145732
weekly	74.073548	61.586389	40.656923	-184.061692	-177.959592
weekly_lower	74.073548	61.586389	40.656923	-184.061692	-177.959592
weekly_upper	74.073548	61.586389	40.656923	-184.061692	-177.959592
yearly	-48.630165	-46.644888	-44.578115	-42.426196	-40.18614
yearly_lower	-48.630165	-46.644888	-44.578115	-42.426196	-40.18614
yearly_upper	-48.630165	-46.644888	-44.578115	-42.426196	-40.18614
multiplicative_terms	0.0	0.0	0.0	0.0	0.0
multiplicative_terms_lower	0.0	0.0	0.0	0.0	0.0
multiplicative_terms_upper	0.0	0.0	0.0	0.0	0.0
yhat	356.247529	345.877155	327.145972	104.710785	113.184451

In [22]:
```
# Selecting valuable columns.
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[22]:

	ds	yhat	yhat_lower	yhat_upper
2181	2023-12-27	356.247529	244.151521	480.909984
2182	2023-12-28	345.877155	215.886879	475.875981
2183	2023-12-29	327.145972	206.039410	458.872915
2184	2023-12-30	104.710785	-15.194178	237.465034
2185	2023-12-31	113.184451	-15.393545	243.623497

In [23]:
```
# Plot the forecast
model.plot(forecast);
```



Evaluating Initial Forecast

In [24]:
```
#To do this, we have to get the y-hat and original y's from the data
metric_df = pd.concat([forecast[['ds','yhat']],final311['y']], axis=1)
metric_df.head()
```

Out[24]:

	ds	yhat	y
0	2018-01-01	241.793414	5.0
1	2018-01-02	258.899715	206.0
2	2018-01-03	235.360434	258.0
3	2018-01-04	225.511380	273.0
4	2018-01-05	207.292491	188.0

In [25]:
```
# The tail has NaN values, because they're predictions - there was no real Y. Let's dr
metric_df.dropna(inplace = True)
```

In [26]:
```
# check the tail, because we added 12 months of forecast.
metric_df.tail()
```

Out[26]:

	ds	yhat	y
1816	2022-12-27	334.831966	1286.0
1817	2022-12-28	310.872292	1001.0
1818	2022-12-29	300.604377	792.0
1819	2022-12-30	281.980130	553.0
1820	2022-12-31	59.655458	50.0

In [27]:
```
#Let's take a look at the numbers - from sklearn.metrics import mean_squared_error, r2
print("R-squared: ", r2_score(metric_df['y'], metric_df['yhat']))
print("Mean Squared Error: ", mean_squared_error(metric_df['y'], metric_df['yhat']))
print("RMSE: ", np.sqrt(mean_squared_error(metric_df['y'], metric_df['yhat'])))
```

```
R-squared:  0.6240199897390777
Mean Squared Error:  9592.7899708935
RMSE:  97.9427892746245
```

Forcasting with holidays

In [28]:
```
from datetime import date
import holidays
```

In [29]:
```
us_holidays = holidays.UnitedStates(years = [2018,2019,2020,2021,2022])

holidays = pd.DataFrame({
  'holiday': us_holidays.values(),
  'ds': us_holidays.keys(),
  'lower_window': 0,
  'upper_window': 0,
})
holidays
```

Out[29]:

	holiday	ds	lower_window	upper_window
0	New Year's Day	2018-01-01	0	0
1	Martin Luther King Jr. Day	2018-01-15	0	0
2	Washington's Birthday	2018-02-19	0	0
3	Memorial Day	2018-05-28	0	0
4	Independence Day	2018-07-04	0	0
5	Labor Day	2018-09-03	0	0
6	Columbus Day	2018-10-08	0	0
7	Veterans Day	2018-11-11	0	0
8	Veterans Day (Observed)	2018-11-12	0	0
9	Thanksgiving	2018-11-22	0	0
10	Christmas Day	2018-12-25	0	0
11	New Year's Day	2019-01-01	0	0
12	Martin Luther King Jr. Day	2019-01-21	0	0
13	Washington's Birthday	2019-02-18	0	0
14	Memorial Day	2019-05-27	0	0
15	Independence Day	2019-07-04	0	0
16	Labor Day	2019-09-02	0	0
17	Columbus Day	2019-10-14	0	0
18	Veterans Day	2019-11-11	0	0
19	Thanksgiving	2019-11-28	0	0
20	Christmas Day	2019-12-25	0	0
21	New Year's Day	2020-01-01	0	0
22	Martin Luther King Jr. Day	2020-01-20	0	0
23	Washington's Birthday	2020-02-17	0	0
24	Memorial Day	2020-05-25	0	0
25	Independence Day	2020-07-04	0	0
26	Independence Day (Observed)	2020-07-03	0	0
27	Labor Day	2020-09-07	0	0
28	Columbus Day	2020-10-12	0	0
29	Veterans Day	2020-11-11	0	0
30	Thanksgiving	2020-11-26	0	0
31	Christmas Day	2020-12-25	0	0
32	New Year's Day	2021-01-01	0	0
33	New Year's Day (Observed)	2021-12-31	0	0

	holiday	ds	lower_window	upper_window
34	Martin Luther King Jr. Day	2021-01-18	0	0
35	Washington's Birthday	2021-02-15	0	0
36	Memorial Day	2021-05-31	0	0
37	Juneteenth National Independence Day	2021-06-19	0	0
38	Juneteenth National Independence Day (Observed)	2021-06-18	0	0
39	Independence Day	2021-07-04	0	0
40	Independence Day (Observed)	2021-07-05	0	0
41	Labor Day	2021-09-06	0	0
42	Columbus Day	2021-10-11	0	0
43	Veterans Day	2021-11-11	0	0
44	Thanksgiving	2021-11-25	0	0
45	Christmas Day	2021-12-25	0	0
46	Christmas Day (Observed)	2021-12-24	0	0
47	New Year's Day	2022-01-01	0	0
48	Martin Luther King Jr. Day	2022-01-17	0	0
49	Washington's Birthday	2022-02-21	0	0
50	Memorial Day	2022-05-30	0	0
51	Juneteenth National Independence Day	2022-06-19	0	0
52	Juneteenth National Independence Day (Observed)	2022-06-20	0	0
53	Independence Day	2022-07-04	0	0
54	Labor Day	2022-09-05	0	0
55	Columbus Day	2022-10-10	0	0
56	Veterans Day	2022-11-11	0	0
57	Thanksgiving	2022-11-24	0	0
58	Christmas Day	2022-12-25	0	0
59	Christmas Day (Observed)	2022-12-26	0	0

In [30]:
```
#Now let's set up prophet to model our data using holidays - Instantiate and fit the m
model = Prophet(holidays=holidays,
                weekly_seasonality=False)

# model.add_seasonality(name='monthly', period=30.5, fourier_order=5)
model.fit(final311)
```

```
13:36:52 - cmdstanpy - INFO - Chain [1] start processing
13:36:53 - cmdstanpy - INFO - Chain [1] done processing
```

Out[30]:
```
<prophet.forecaster.Prophet at 0x19c64301d30>
```

In [31]:
```
#We've instantiated the model, so now we need to build our future dates to forecast ir
future = model.make_future_dataframe(periods=365, freq = 'd')
future.tail()

#... and then run our future data through prophet's model
forecast = model.predict(future)

forecast.head().T
```

Out[31]:

	0	1	2	3	4
ds	2018-01-01 00:00:00	2018-01-02 00:00:00	2018-01-03 00:00:00	2018-01-04 00:00:00	2018-01-05 00:00:00
trend	201.943129	201.925359	201.907589	201.88982	201.87205
yhat_lower	-181.950191	-6.197508	3.36591	-1.572221	2.840536
yhat_upper	195.869064	368.279945	354.579985	374.957103	377.7092
trend_lower	201.943129	201.925359	201.907589	201.88982	201.87205
...	...	...	...	...	...
yearly_upper	-20.9744	-18.654309	-16.474478	-14.428073	-12.506694
multiplicative_terms	0.0	0.0	0.0	0.0	0.0
multiplicative_terms_lower	0.0	0.0	0.0	0.0	0.0
multiplicative_terms_upper	0.0	0.0	0.0	0.0	0.0
yhat	9.463243	183.27105	185.433111	187.461747	189.365356

67 rows × 5 columns

In [32]:
```
future['ds'] = future['ds'].to_numpy().astype('datetime64[M]')
```

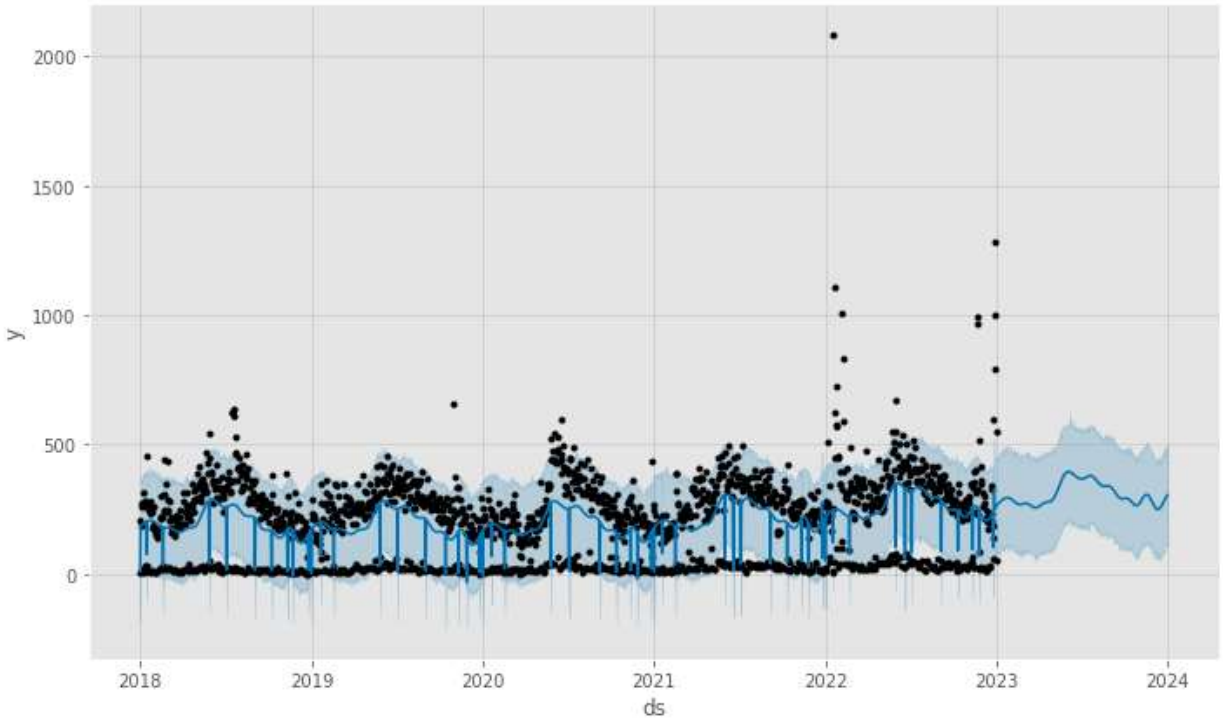
In [33]:
```
#while our new df contains a bit of data, we only care about a few features...
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[33]:

	ds	yhat	yhat_lower	yhat_upper
2181	2023-12-27	293.038640	115.736534	487.021878
2182	2023-12-28	296.243597	104.924234	471.739686
2183	2023-12-29	299.325780	114.506818	496.653253
2184	2023-12-30	302.273007	105.459326	489.464292
2185	2023-12-31	305.077569	114.549100	496.310797

Visualizing with holidays

In [34]:
```
# use Prophet's .plot() method to visualize your timeseries.
model.plot(forecast);
```



In [35]:
```
metric_df = pd.concat([forecast[['ds','yhat']],final311['y']], axis=1)
metric_df.head()
```

Out[35]:

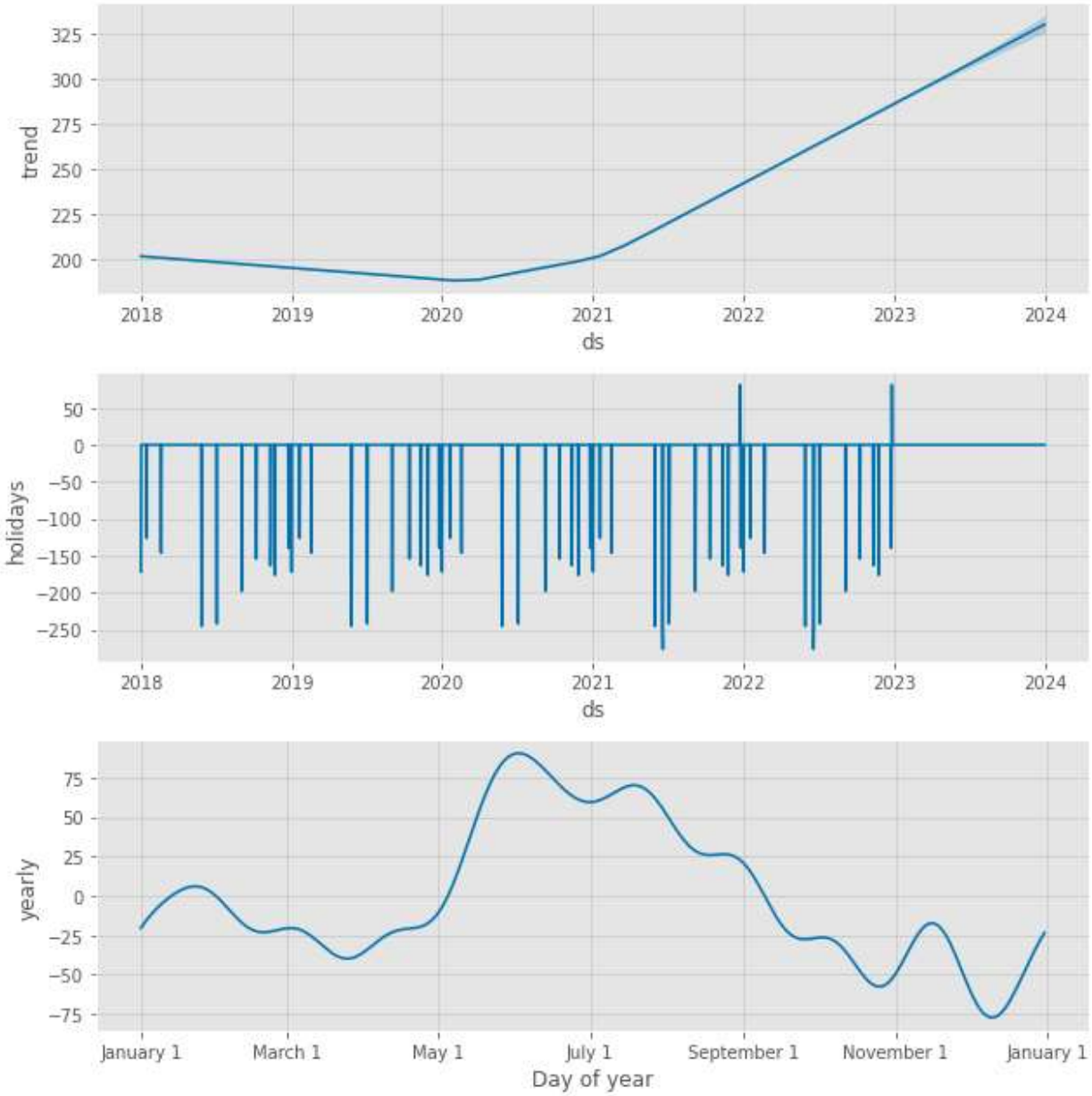
	ds	yhat	y
0	2018-01-01	9.463243	5.0
1	2018-01-02	183.271050	206.0
2	2018-01-03	185.433111	258.0
3	2018-01-04	187.461747	273.0
4	2018-01-05	189.365356	188.0

In [36]:
```
# The tail has NaN values, because they're predictions - there was no real Y. Let's dr
metric_df.dropna(inplace = True)
```

In [37]:
```
print("R-squared: ", r2_score(metric_df['y'], metric_df['yhat']))
print("Mean Squared Error: ", mean_squared_error(metric_df['y'], metric_df['yhat']))
print("RMSE: ", np.sqrt(mean_squared_error(metric_df['y'], metric_df['yhat'])))
```

```
R-squared:  0.14163160836069444
Mean Squared Error:  21900.493307969184
RMSE:  147.98815259327074
```

In [38]:
```
# View the components
model.plot_components(forecast);
```



AMIRA Forcasting

In [39]:
```
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in c:\users\matth\anaconda3\lib\site-packages
(2.0.3)
Requirement already satisfied: pandas>=0.19 in c:\users\matth\anaconda3\lib\site-pack
ages (from pmdarima) (1.4.2)
Requirement already satisfied: statsmodels>=0.13.2 in c:\users\matth\anaconda3\lib\si
te-packages (from pmdarima) (0.13.2)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in c:\users\matth\anaconda
3\lib\site-packages (from pmdarima) (61.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in c:\users\matth\ana
conda3\lib\site-packages (from pmdarima) (0.29.28)
Requirement already satisfied: numpy>=1.21.2 in c:\users\matth\anaconda3\lib\site-pac
kages (from pmdarima) (1.21.5)
Requirement already satisfied: joblib>=0.11 in c:\users\matth\anaconda3\lib\site-pack
ages (from pmdarima) (1.1.0)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\matth\anaconda3\lib\sit
e-packages (from pmdarima) (1.0.2)
Requirement already satisfied: urllib3 in c:\users\matth\anaconda3\lib\site-packages
(from pmdarima) (1.26.9)
Requirement already satisfied: scipy>=1.3.2 in c:\users\matth\anaconda3\lib\site-pack
ages (from pmdarima) (1.7.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\matth\anaconda3\lib\site-pack
ages (from pandas>=0.19->pmdarima) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\matth\anaconda3\lib
\site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\matth\anaconda3\lib\site-packages
(from python-dateutil>=2.8.1->pandas>=0.19->pmdarima) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\matth\anaconda3\lib\s
ite-packages (from scikit-learn>=0.22->pmdarima) (2.2.0)
Requirement already satisfied: patsy>=0.5.2 in c:\users\matth\anaconda3\lib\site-pack
ages (from statsmodels>=0.13.2->pmdarima) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\users\matth\anaconda3\lib\site-p
ackages (from statsmodels>=0.13.2->pmdarima) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\matth\anaconda3\l
ib\site-packages (from packaging>=21.3->statsmodels>=0.13.2->pmdarima) (3.0.4)
```

In [40]:
```
!pip install statsmodels
```

```
Requirement already satisfied: statsmodels in c:\users\matth\anaconda3\lib\site-packa
ges (0.13.2)
Requirement already satisfied: numpy>=1.17 in c:\users\matth\anaconda3\lib\site-packa
ges (from statsmodels) (1.21.5)
Requirement already satisfied: scipy>=1.3 in c:\users\matth\anaconda3\lib\site-packag
es (from statsmodels) (1.7.3)
Requirement already satisfied: pandas>=0.25 in c:\users\matth\anaconda3\lib\site-pack
ages (from statsmodels) (1.4.2)
Requirement already satisfied: patsy>=0.5.2 in c:\users\matth\anaconda3\lib\site-pack
ages (from statsmodels) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\users\matth\anaconda3\lib\site-p
ackages (from statsmodels) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\matth\anaconda3\l
ib\site-packages (from packaging>=21.3->statsmodels) (3.0.4)
Requirement already satisfied: pytz>=2020.1 in c:\users\matth\anaconda3\lib\site-pack
ages (from pandas>=0.25->statsmodels) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\matth\anaconda3\lib
\site-packages (from pandas>=0.25->statsmodels) (2.8.2)
Requirement already satisfied: six in c:\users\matth\anaconda3\lib\site-packages (fro
m patsy>=0.5.2->statsmodels) (1.16.0)
```

In [41]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [42]:
```
from statsmodels.tsa.stattools import adfuller
```

In [43]:
```
def ad_test(dataset):
    dftest = adfuller(dataset, autolag = 'AIC')
    print('1. ADF : ', dftest[0])
    print('2. P-Value : ', dftest[1])
    print('3. Num Of Lags : ', dftest[2])
    print('4. Num Of Observations Used For ADF Regression and Critical Values Calculat
    print('5. Critical Values : ')
    for key, val in dftest[4].items():
        print('\t', key, ':',val)
```

In [44]:
```
ad_test(final311['y'])
```

```
1. ADF :  -4.159648013286395
2. P-Value :  0.0007705551428364633
3. Num Of Lags :  22
4. Num Of Observations Used For ADF Regression and Critical Values Calculation :  179
8
5. Critical Values :
         1% : -3.4339921916016345
         5% : -2.8631488249300574
         10% : -2.5676264862577503
```

In [45]:
```
from pmdarima import auto_arima
# Ignore harmless warnings
import warnings
warnings.filterwarnings('ignore')
```

In [46]:
```
stepwise_fit = auto_arima(final311['y'],
                          trace=True,
                          supress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=22940.456, Time=1.17 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=23905.707, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=23889.920, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=23496.885, Time=0.35 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=23903.707, Time=0.02 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=23109.973, Time=0.83 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=23050.822, Time=0.81 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=1.66 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=22676.543, Time=2.37 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=23035.176, Time=1.56 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=22868.496, Time=2.88 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=22624.289, Time=2.94 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=22991.928, Time=1.83 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=2.70 sec
 ARIMA(2,1,5)(0,0,0)[0] intercept   : AIC=22724.344, Time=3.24 sec
 ARIMA(1,1,5)(0,0,0)[0] intercept   : AIC=22978.711, Time=2.83 sec
 ARIMA(3,1,5)(0,0,0)[0] intercept   : AIC=22718.467, Time=3.68 sec
 ARIMA(2,1,4)(0,0,0)[0]             : AIC=22963.283, Time=2.11 sec

Best model:  ARIMA(2,1,4)(0,0,0)[0] intercept
Total fit time: 31.099 seconds
```

In [47]:
```
stepwise_fit.summary()
```

Out[47]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	1821
Model:	SARIMAX(2, 1, 4)	**Log Likelihood**	-11304.144
Date:	Wed, 10 May 2023	**AIC**	22624.289
Time:	13:37:31	**BIC**	22668.341
Sample:	0	**HQIC**	22640.541
	- 1821		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0085	0.238	0.036	0.972	-0.458	0.474
ar.L1	1.2459	0.002	597.236	0.000	1.242	1.250
ar.L2	-0.9998	0.000	-2759.493	0.000	-1.001	-0.999
ma.L1	-1.9438	0.022	-89.919	0.000	-1.986	-1.901
ma.L2	1.6307	0.050	32.758	0.000	1.533	1.728
ma.L3	-0.4495	0.053	-8.499	0.000	-0.553	-0.346
ma.L4	-0.1856	0.026	-7.255	0.000	-0.236	-0.135
sigma2	1.836e+04	228.720	80.290	0.000	1.79e+04	1.88e+04

Ljung-Box (L1) (Q):	2.48	**Jarque-Bera (JB):**	84320.28
Prob(Q):	0.12	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.11	**Skew:**	2.91
Prob(H) (two-sided):	0.00	**Kurtosis:**	35.83

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [48]:
```
final311 = final311.set_index('ds')
```

In [49]:
```
final311.head()
```

Out[49]:

ds	y
2018-01-01	5
2018-01-02	206
2018-01-03	258
2018-01-04	273
2018-01-05	188

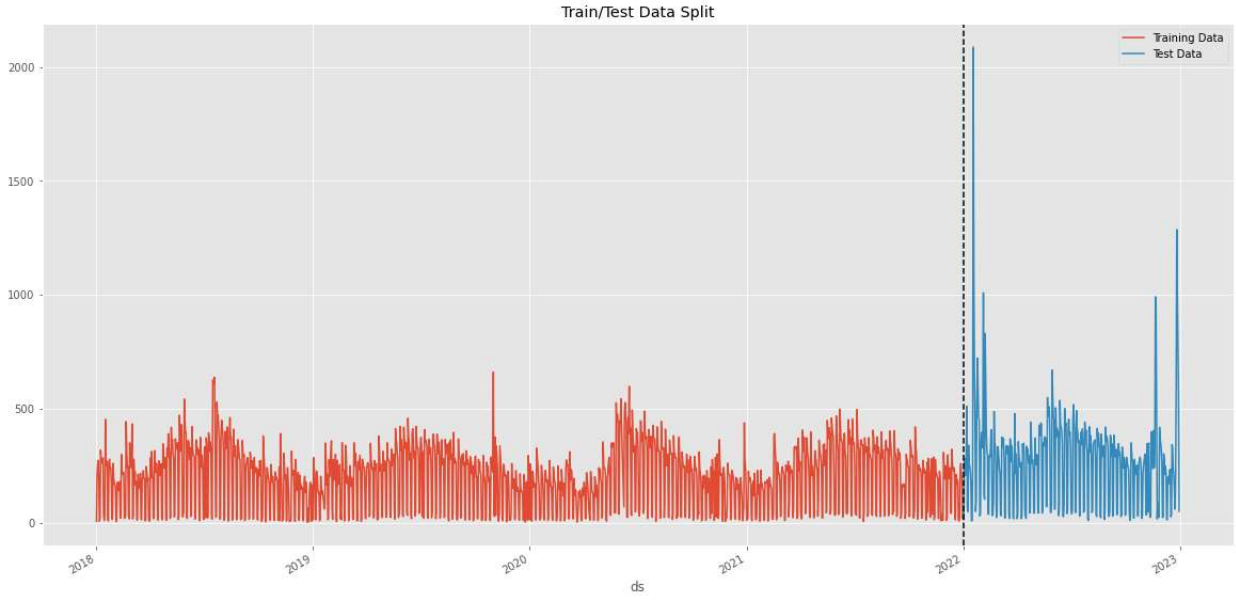
In [50]:
```
final311.index = pd.to_datetime(final311.index)
```

In [51]:
```
final311.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1821 entries, 2018-01-01 to 2022-12-31
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   y       1821 non-null   int64
dtypes: int64(1)
memory usage: 28.5 KB
```

In [52]:
```
split_date = '2022-01-01'
final311_train = final311.loc[final311.index <= split_date].copy()
final311_test = final311.loc[final311.index > split_date].copy()
```

In [53]:
```
# The storm in december of 2022 may skew the test data
fig, ax = plt.subplots(figsize=(20,10))
final311_train.plot(ax=ax, label='Training Data', title='Train/Test Data Split')
final311_test.plot(ax=ax, label='Testing Data')
ax.axvline(split_date, color='black', ls='--')
ax.legend(['Training Data', 'Test Data'])
plt.show()
```



In [54]:
```
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
```

In [55]:
```
model=sm.tsa.ARIMA(final311['y'], order=(5,0,5))
model=model.fit()
model.summary()
```

Out[55]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	1821
Model:	ARIMA(5, 0, 5)	**Log Likelihood**	-11001.851
Date:	Wed, 10 May 2023	**AIC**	22027.702
Time:	13:37:34	**BIC**	22093.788
Sample:	0	**HQIC**	22052.083
	- 1821		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	207.9556	31.613	6.578	0.000	145.995	269.916
ar.L1	1.7698	0.013	133.045	0.000	1.744	1.796
ar.L2	-2.2120	0.015	-144.658	0.000	-2.242	-2.182
ar.L3	2.1887	0.022	99.806	0.000	2.146	2.232
ar.L4	-1.7607	0.018	-98.954	0.000	-1.796	-1.726
ar.L5	0.9616	0.015	65.654	0.000	0.933	0.990
ma.L1	-1.4852	0.019	-78.268	0.000	-1.522	-1.448
ma.L2	1.8473	0.032	58.350	0.000	1.785	1.909
ma.L3	-1.7258	0.044	-38.827	0.000	-1.813	-1.639
ma.L4	1.3294	0.037	36.177	0.000	1.257	1.401
ma.L5	-0.6176	0.029	-21.645	0.000	-0.674	-0.562
sigma2	1.367e+04	149.941	91.195	0.000	1.34e+04	1.4e+04

Ljung-Box (L1) (Q):	4.30	**Jarque-Bera (JB):**	364021.15
Prob(Q):	0.04	**Prob(JB):**	0.00
Heteroskedasticity (H):	3.04	**Skew:**	4.27
Prob(H) (two-sided):	0.00	**Kurtosis:**	71.74

Warnings:

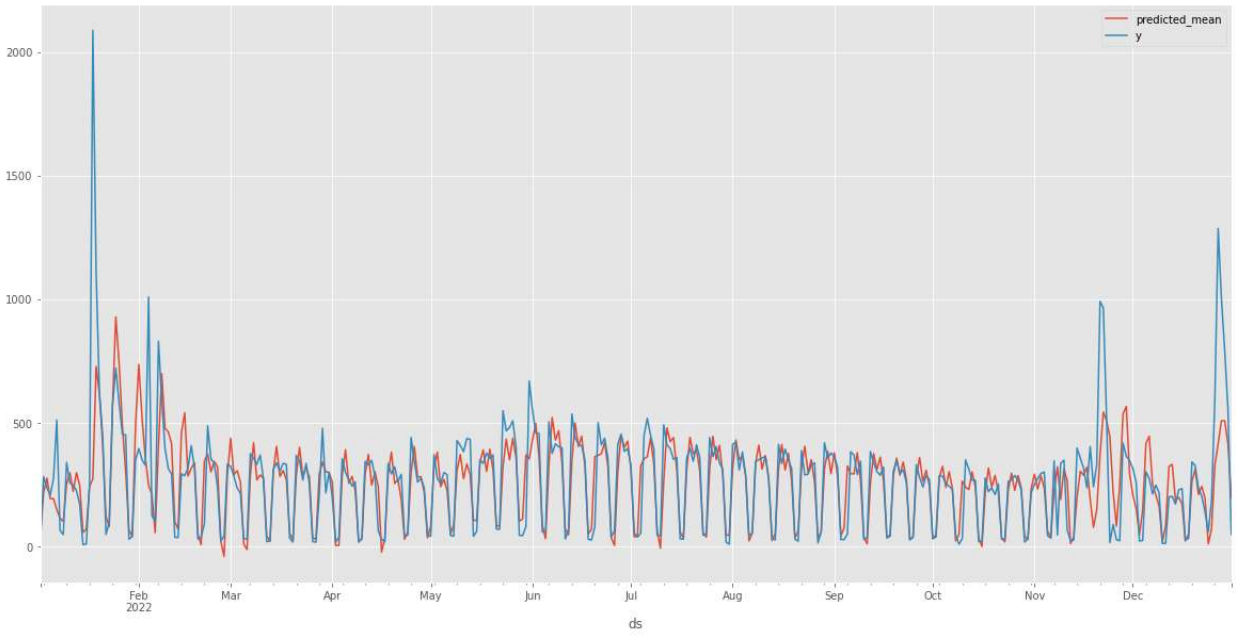
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [56]:
```
start=len(final311_train)
end=len(final311_train)+len(final311_test)-1
pred=model.predict(start=start, end=end, typ='levels')
pred.index=final311.index[start:end+1]
print(pred)
```

```
ds
2022-01-02     43.601562
2022-01-03    218.478687
2022-01-04    276.551172
2022-01-05    193.433580
2022-01-06    193.371463
                  ...
2022-12-27    416.625223
2022-12-28    508.154247
2022-12-29    507.622138
2022-12-30    414.139234
2022-12-31    197.159496
Name: predicted_mean, Length: 364, dtype: float64
```

In [57]:
```
pred.plot(legend=True)
final311_test['y'].plot(legend=True)
```

Out[57]:
```
<AxesSubplot:xlabel='ds'>
```



In [58]:
```
final311['y'].mean()
```

Out[58]:
```
207.95496979681494
```

In [59]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse=sqrt(mean_squared_error(pred, final311_test['y']))
rmse
```

Out[59]:
```
156.0639117751073
```

In [60]:
```
#Fit model on all df5
model2 = sm.tsa.ARIMA(final311['y'], order=(5,0,5))
```

```
model2=model2.fit()
final311.tail() # predict into the future after training
```

Out[60]:

ds	y
2022-12-27	1286
2022-12-28	1001
2022-12-29	792
2022-12-30	553
2022-12-31	50

In [61]:
```
index_future_dates=pd.date_range(start='2023-01-01', end='2023-12-31')
pred=model2.predict(start=len(final311), end=len(final311)+(364), type='levels').renar
pred.index=index_future_dates
print(pred)
```

```
2023-01-01    186.356597
2023-01-02    570.758293
2023-01-03    769.166249
2023-01-04    679.945459
2023-01-05    572.970127
                 ...
2023-12-27    305.960515
2023-12-28    249.493834
2023-12-29    148.526088
2023-12-30     45.133561
2023-12-31     99.255170
Freq: D, Name: ARIMA Predictions, Length: 365, dtype: float64
```

2023 Arima Predictions

In [62]:
```
pred.plot(figsize=(12,5),legend=True)
```

Out[62]:
```
<AxesSubplot:>
```