Analyzing Patterns in Chicago Motor Vehicle Crashes Using Time-Series Analysis and Classification Techniques

Christina Trotta

# “Analyzing Patterns in Chicago Motor Vehicle Crashes Using Time-Series and Classification Techniques”

By: Christina Trotta Faculty Mentor: Dr. Andrew Ross

### Table of Contents:

#### Section 1 - Introduction

#### Section 2 - Exploratory Data Analysis

#### Section 3 - Prophet Forcasting and Time-Series Analysis

#### Section 4 - Holt Winters Smoothing

#### Section 5 - Conclusion

# Section 1 - Introduction

Understanding and forecasting traffic crashes is essential for improving public safety, resource allocation, and infrastructure planning. With Chicago’s vast and detailed traffic data available at the daily level, this project aimed to explore how past crash patterns — along with external variables like weather and roadway conditions — could be used to predict future crash rates. The central question driving this work was: “How much does yesterday’s crash rate help predict today’s?”

To answer this, we used time series modeling techniques including Holt-Winters exponential smoothing, Prophet forecasting, and SARIMAX regression. We engineered a clean, daily-level dataset of crashes from 2018 to 2024, incorporating binary flags for crash conditions and merging with weather data from both O’Hare and Midway airports. The project explored the predictive power of lagged crash data alone, as well as how factors like precipitation, lighting, and road defects might help improve forecasting accuracy.

This data comes from the City of Chicago’s open data portal, which provides a wealth of information on traffic incidents. The dataset includes details such as the date, time, location, and severity of each crash, as well as contributing factors like weather conditions and road defects. By analyzing this data, we aimed to identify patterns and trends that could inform future traffic safety initiatives. It is recorded by the city government, not citizens, which should ensure a complete record of significant incidents. However, minor fender benders or unreported incidents may not be included in the dataset.

# Section 2 - Exploratory Data Analysis

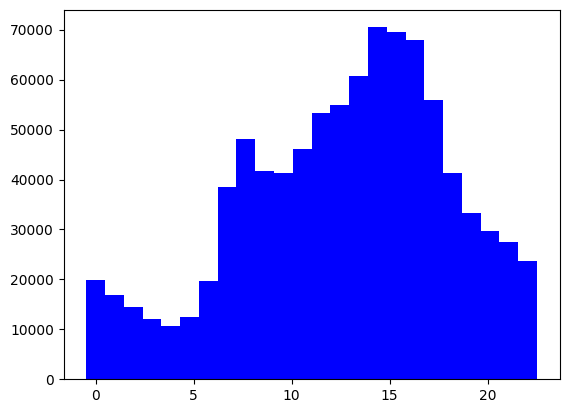
In this section we will take a first glance at the data. Are there any obvious trends in the data? What should we expect from this data set? We need to pul summary statistics. Additionally, are there any obvious flaws with the data? It’s important that we take into account any missing data, duplicate data, and handle it.

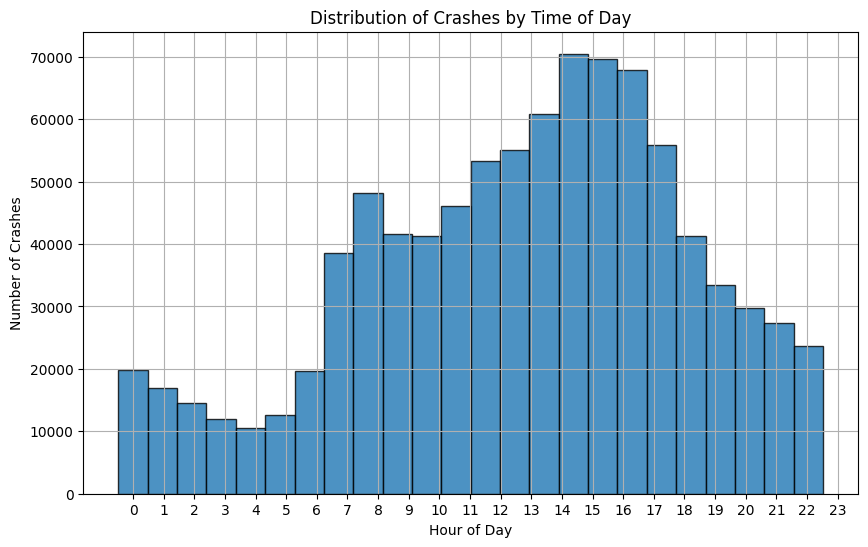
### General Data Trends and Descriptive Statistics

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 910387 entries, 0 to 910386  
Data columns (total 48 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 CRASH\_RECORD\_ID 910387 non-null object   
 1 CRASH\_DATE\_EST\_I 67175 non-null object   
 2 CRASH\_DATE 910387 non-null object   
 3 POSTED\_SPEED\_LIMIT 910387 non-null int64   
 4 TRAFFIC\_CONTROL\_DEVICE 910387 non-null object   
 5 DEVICE\_CONDITION 910387 non-null object   
 6 WEATHER\_CONDITION 910387 non-null object   
 7 LIGHTING\_CONDITION 910387 non-null object   
 8 FIRST\_CRASH\_TYPE 910387 non-null object   
 9 TRAFFICWAY\_TYPE 910387 non-null object   
 10 LANE\_CNT 199023 non-null float64  
 11 ALIGNMENT 910387 non-null object   
 12 ROADWAY\_SURFACE\_COND 910387 non-null object   
 13 ROAD\_DEFECT 910387 non-null object   
 14 REPORT\_TYPE 881887 non-null object   
 15 CRASH\_TYPE 910387 non-null object   
 16 INTERSECTION\_RELATED\_I 209237 non-null object   
 17 NOT\_RIGHT\_OF\_WAY\_I 41399 non-null object   
 18 HIT\_AND\_RUN\_I 285524 non-null object   
 19 DAMAGE 910387 non-null object   
 20 DATE\_POLICE\_NOTIFIED 910387 non-null object   
 21 PRIM\_CONTRIBUTORY\_CAUSE 910387 non-null object   
 22 SEC\_CONTRIBUTORY\_CAUSE 910387 non-null object   
 23 STREET\_NO 910387 non-null int64   
 24 STREET\_DIRECTION 910383 non-null object   
 25 STREET\_NAME 910386 non-null object   
 26 BEAT\_OF\_OCCURRENCE 910382 non-null float64  
 27 PHOTOS\_TAKEN\_I 12493 non-null object   
 28 STATEMENTS\_TAKEN\_I 21063 non-null object   
 29 DOORING\_I 2869 non-null object   
 30 WORK\_ZONE\_I 5055 non-null object   
 31 WORK\_ZONE\_TYPE 3899 non-null object   
 32 WORKERS\_PRESENT\_I 1301 non-null object   
 33 NUM\_UNITS 910387 non-null int64   
 34 MOST\_SEVERE\_INJURY 908379 non-null object   
 35 INJURIES\_TOTAL 908393 non-null float64  
 36 INJURIES\_FATAL 908393 non-null float64  
 37 INJURIES\_INCAPACITATING 908393 non-null float64  
 38 INJURIES\_NON\_INCAPACITATING 908393 non-null float64  
 39 INJURIES\_REPORTED\_NOT\_EVIDENT 908393 non-null float64  
 40 INJURIES\_NO\_INDICATION 908393 non-null float64  
 41 INJURIES\_UNKNOWN 908393 non-null float64  
 42 CRASH\_HOUR 910387 non-null int64   
 43 CRASH\_DAY\_OF\_WEEK 910387 non-null int64   
 44 CRASH\_MONTH 910387 non-null int64   
 45 LATITUDE 903755 non-null float64  
 46 LONGITUDE 903755 non-null float64  
 47 LOCATION 903755 non-null object   
dtypes: float64(11), int64(6), object(31)  
memory usage: 333.4+ MB

|  | POSTED\_SPEED\_LIMIT | LANE\_CNT | STREET\_NO | BEAT\_OF\_OCCURRENCE | NUM\_UNITS | INJURIES\_TOTAL | INJURIES\_FATAL | INJURIES\_INCAPACITATING | INJURIES\_NON\_INCAPACITATING | INJURIES\_REPORTED\_NOT\_EVIDENT | INJURIES\_NO\_INDICATION | INJURIES\_UNKNOWN | CRASH\_HOUR | CRASH\_DAY\_OF\_WEEK | CRASH\_MONTH | LATITUDE | LONGITUDE |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 910387.000000 | 1.990230e+05 | 910387.000000 | 910382.000000 | 910387.000000 | 908393.000000 | 908393.000000 | 908393.000000 | 908393.000000 | 908393.000000 | 908393.000000 | 908393.0 | 910387.000000 | 910387.000000 | 910387.000000 | 903755.000000 | 903755.000000 |
| mean | 28.422284 | 1.332937e+01 | 3689.169599 | 1245.759196 | 2.035159 | 0.194431 | 0.001183 | 0.019645 | 0.108830 | 0.064774 | 2.000673 | 0.0 | 13.199414 | 4.119371 | 6.745317 | 41.855095 | -87.673537 |
| std | 6.100338 | 2.961497e+03 | 2879.309252 | 704.806174 | 0.451768 | 0.572585 | 0.037298 | 0.164182 | 0.425255 | 0.327036 | 1.156244 | 0.0 | 5.573139 | 1.980622 | 3.410060 | 0.340557 | 0.692681 |
| min | 0.000000 | 0.000000e+00 | 0.000000 | 111.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | -87.936193 |
| 25% | 30.000000 | 2.000000e+00 | 1255.000000 | 715.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.0 | 9.000000 | 2.000000 | 4.000000 | 41.783320 | -87.721841 |
| 50% | 30.000000 | 2.000000e+00 | 3201.000000 | 1212.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.000000 | 0.0 | 14.000000 | 4.000000 | 7.000000 | 41.874975 | -87.674313 |
| 75% | 30.000000 | 4.000000e+00 | 5562.000000 | 1822.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.000000 | 0.0 | 17.000000 | 6.000000 | 10.000000 | 41.924589 | -87.633634 |
| max | 99.000000 | 1.191625e+06 | 451100.000000 | 6100.000000 | 18.000000 | 21.000000 | 4.000000 | 10.000000 | 21.000000 | 15.000000 | 61.000000 | 0.0 | 23.000000 | 7.000000 | 12.000000 | 42.022780 | 0.000000 |

This is a preliminary test to see the general shape of the data. We can see the raw data has 910387 entries. There are 47 different fields in this data set.



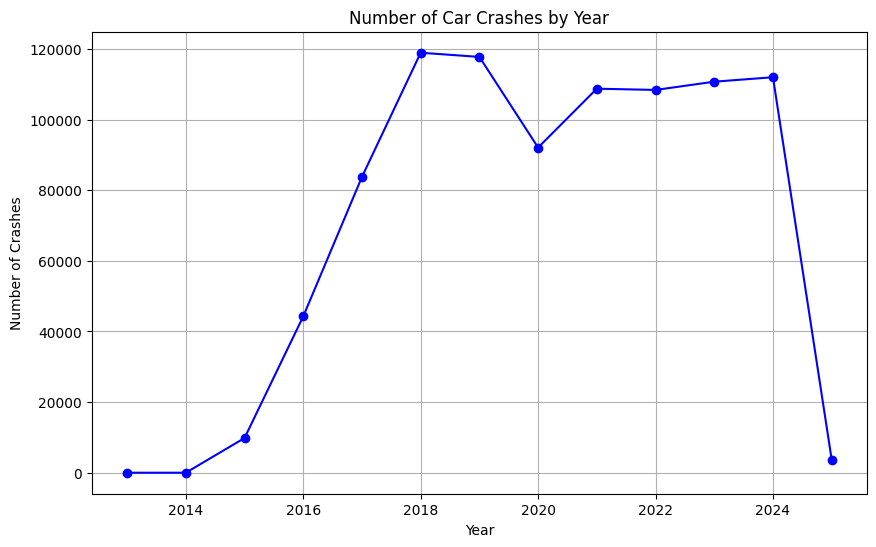


The histogram shows the frequency of crashes by hour of day. We see a large increase in the number of crashes around “rush hour”, around the hours 2-6pm. Many people are commuting home from work during this time period. We also see an upward trend in crashes in the morning, around hours 7-10am. This increase compared to the early hours of the day again makes sense, due to the heavy increase in traffic for commuters on their way to work. The number of crashes during this timeframe 7-10am is similar to that during the 6-8pm timeframe. We can see a low number of crashes in the early hours of the day, from 12-5am.

Questions from this initial histogram: - How well does time/hour of day predict the probability/the number of car crashes?

1. How many crashes occurred in total, and how is this distributed by year or month? (Examine trends over time to identify whether crashes are increasing, decreasing, or remaining constant.)

C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2656098333.py:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.  
 df\_raw['CRASH\_DATE'] = pd.to\_datetime(df\_raw['CRASH\_DATE'])



The above graph shows the number of car crashes by year. We can see a large increase in the number of crashes after 2015. This might not be extremely accurate however, since the graph is showing that the number of crashes in both 2013 and 2014 were both 0? Or a very small number. Let’s dig a little further into these years to see what the data during this 2-year time span looks like.

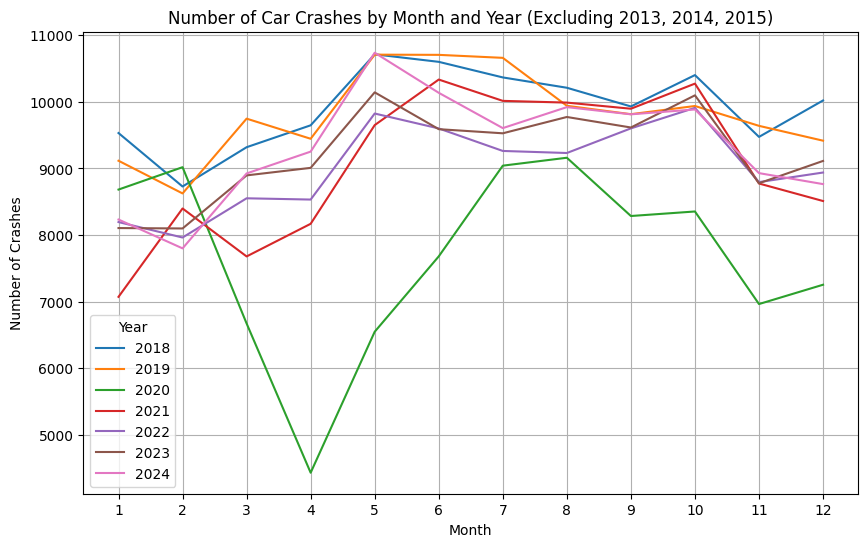
CRASH\_YEAR CRASH\_COUNT  
0 2013 2  
1 2014 6  
2 2015 9830  
3 2016 44297  
4 2017 83786  
5 2018 118950  
6 2019 117762  
7 2020 92094  
8 2021 108765  
9 2022 108410  
10 2023 110747  
11 2024 112006  
12 2025 3732

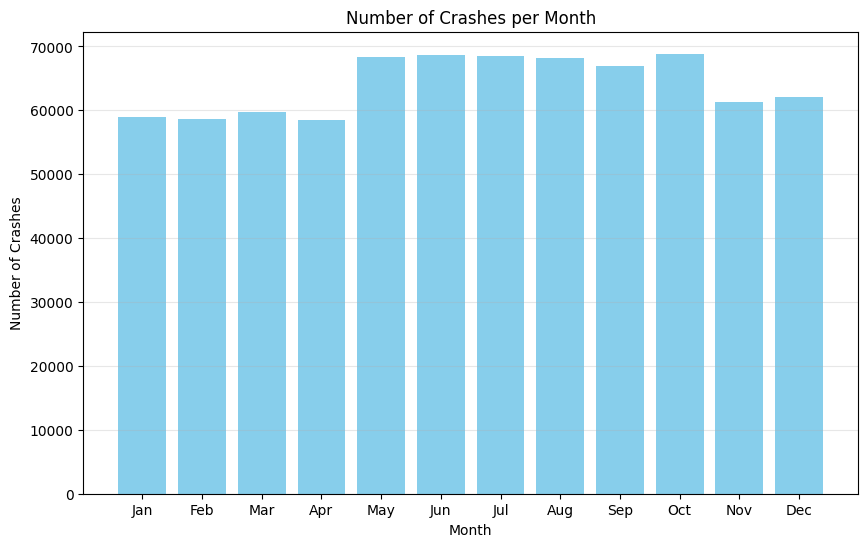
It seems like there is a discrepancy in the data reporting for years 2013 and 2014. This could be a reporting error, computer error, or perhaps the recording system was slowly being phased in throughout the city during this time.

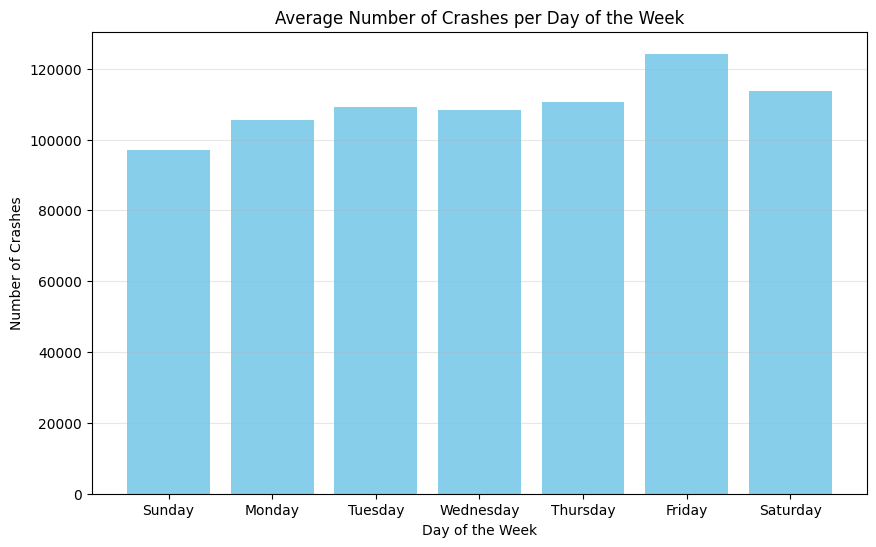
Since 2015, 2016, and 2017 are also a little bit low compared to years 2018-2024, we should remove these from further analysis as well.

Additionally, since we don’t have a full year’s worth of data for 2025, we will also explude this data from our analysis.

1. What is the average number of crashes per day, week, or month?







We can begin to see some trends in the monthly data, such as higher crash rates in the summer than in the winter, but the trends aren’t as obvious or strong as we might expect them to be. Similarly, we can see some trends in the weekly bar chart, indicating that as the week goes on, there are more and more crashes. Again, this trend is not as string as we thought. We will continue to explore this with our time-series modeling.

# Section 3: Prophet Forecasting and Time Series Analysis

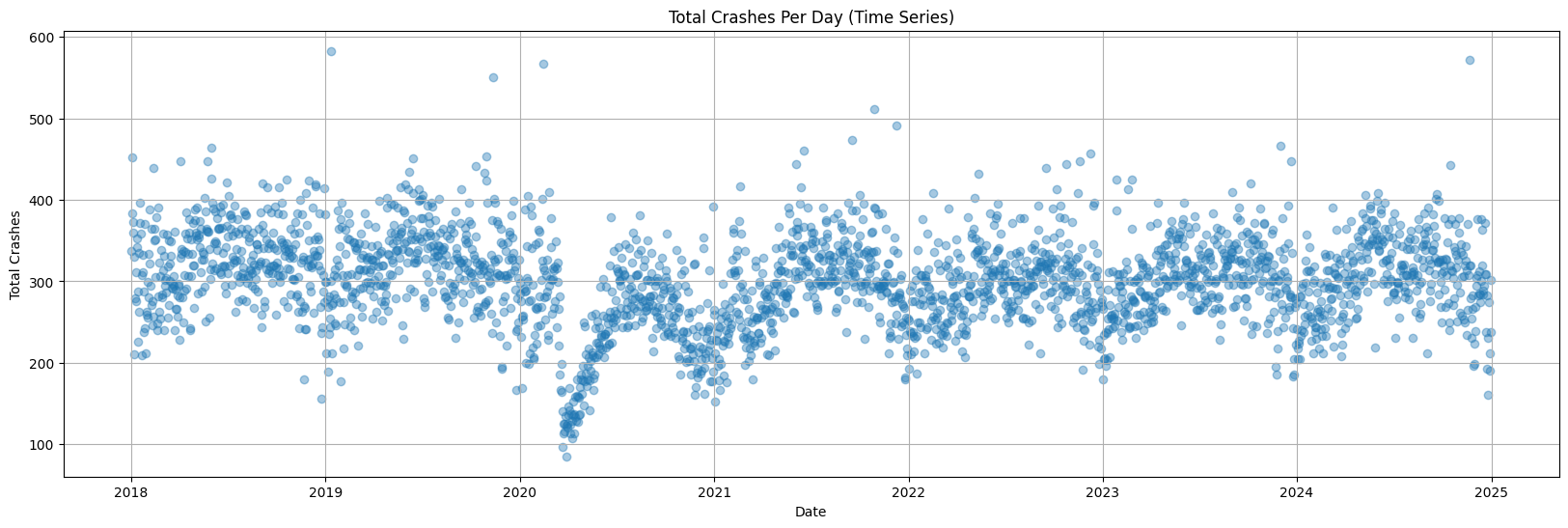
Before we begin our analysis, is there a significant amount of missing data? Are there days where there are no crashes? We should check to see if there is data every day during this timeframe.

287935 277.0  
424268 244.0  
24037 231.0  
251132 221.0  
564161 219.0  
123246 215.0  
263872 214.0  
211318 204.0  
564710 202.0  
532823 201.0  
264935 200.0  
420593 196.0  
157571 193.0  
265103 191.0  
444921 191.0  
464600 190.0  
484548 190.0  
728558 188.0  
224531 185.0  
500424 185.0  
551930 184.0  
237619 180.0  
434522 180.0  
435476 180.0  
437667 180.0  
546227 180.0  
262232 178.0  
417920 175.0  
261954 173.0  
142385 172.0  
237078 170.0  
583514 167.0  
646320 167.0  
246490 165.0  
754845 163.0  
260822 162.0  
638455 162.0  
49347 160.0  
261196 160.0  
315584 158.0  
479901 158.0  
314875 157.0  
148645 156.0  
334675 156.0  
122384 155.0  
267287 155.0  
316532 155.0  
327887 155.0  
171812 154.0  
683537 153.0  
Name: Time\_Difference, dtype: float64

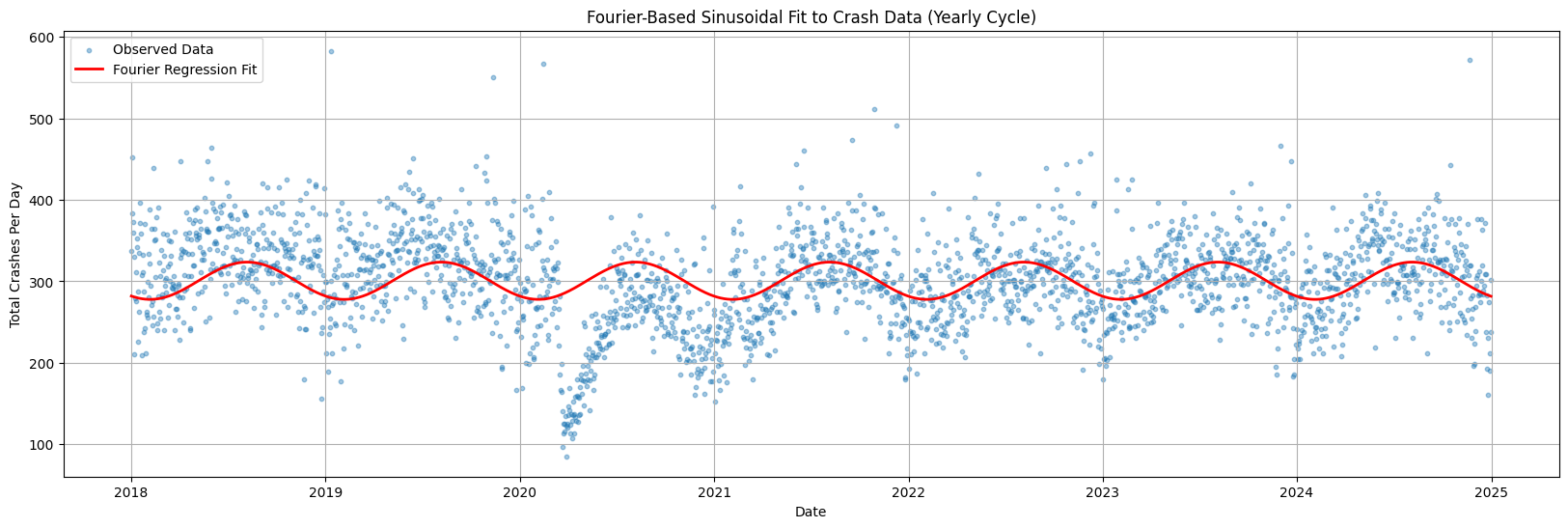
The largest time gap in this dataset is only a couple of hours. We can assume that there are no significant issues with this data and proceeed with our time-series analysis.

Begin time-series analysis on this data

CRASH\_DATE Total\_Crashes  
0 2018-01-01 337  
1 2018-01-02 452  
2 2018-01-03 383  
3 2018-01-04 360  
4 2018-01-05 373



Here we can see the aggregated Total Crashes Per Day graph, and we can see some trends starting to show themselves. Each year, there appears to be an upside-down U shaped pattern - meaning decreased crash rates in the winter and higher in the summer. We can also see a huge dip in the crash rates in early 2020. Let’s fit a Fourier Regression curve to this dataset, as it looks approximately sinusoidal.



What does this graph show us?

This graph helps to predict crash rates based on time of year. It is only accounting for monthly cycles, and not weekly cycles. For example, the curve is evaluating each day as if they were the same, not evaluating based on weekly seasonality, such as a Sunday in January.

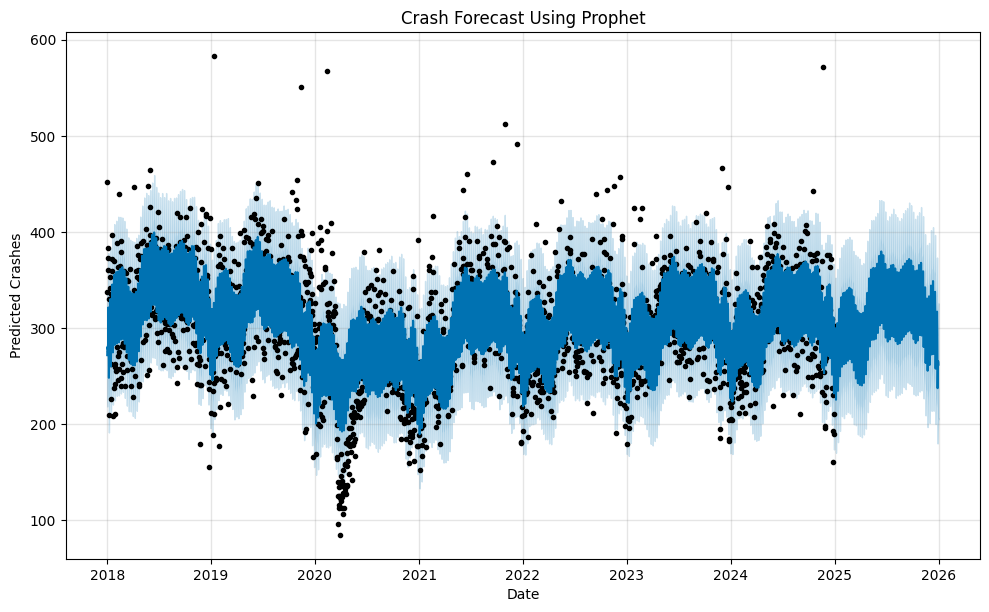
Now we will begin forecasting on this data.

Prophet is a package developed by Meta to forecast time series data. It is designed to handle missing data and outliers, and it can also incorporate seasonality and holidays into the forecasting model. Prophet is particularly useful for time series data that exhibits strong seasonal patterns and several seasons of historical data.

Prophet is a good choice for this data because it can handle the missing data and outliers that are present in the dataset. It can also incorporate seasonality and holidays into the forecasting model, which is important for this data.

ds y Days\_Since\_Start Sin\_Term Cos\_Term Fourier\_Prediction  
0 2018-01-01 337 0 0.000000 1.000000 281.828518  
1 2018-01-02 452 1 0.017213 0.999852 281.606744  
2 2018-01-03 383 2 0.034422 0.999407 281.390614  
3 2018-01-04 360 3 0.051620 0.998667 281.180192  
4 2018-01-05 373 4 0.068802 0.997630 280.975541

23:31:24 - cmdstanpy - INFO - Chain [1] start processing  
23:31:25 - cmdstanpy - INFO - Chain [1] done processing



Crash Forecast results:

This graph shows the predicted number of crashes per day 2018 through 2026, and was modeled on the data from 2018-2024.

Key Components of the Graph - Black Dots = Observed Data (Actual recorded crashes per day). - Dark Blue Line = Model’s Predicted Trend (yhat), showing the expected number of crashes. - Light Blue Shaded Area = Uncertainty Interval (confidence range for predictions). – Darker blue = Higher confidence in the forecast. – Lighter blue (outer edges) = Higher uncertainty in future predictions.

What This Forecast Tells Us

1. The Model Captures Seasonality Well

* The trend clearly shows repeated dips and rises within each year, suggesting strong weekly and yearly seasonal patterns.
* Notice the dip in 2020—this aligns with COVID-19 lockdowns, when fewer people were on the road.

1. Model Confidence Decreases in the Future

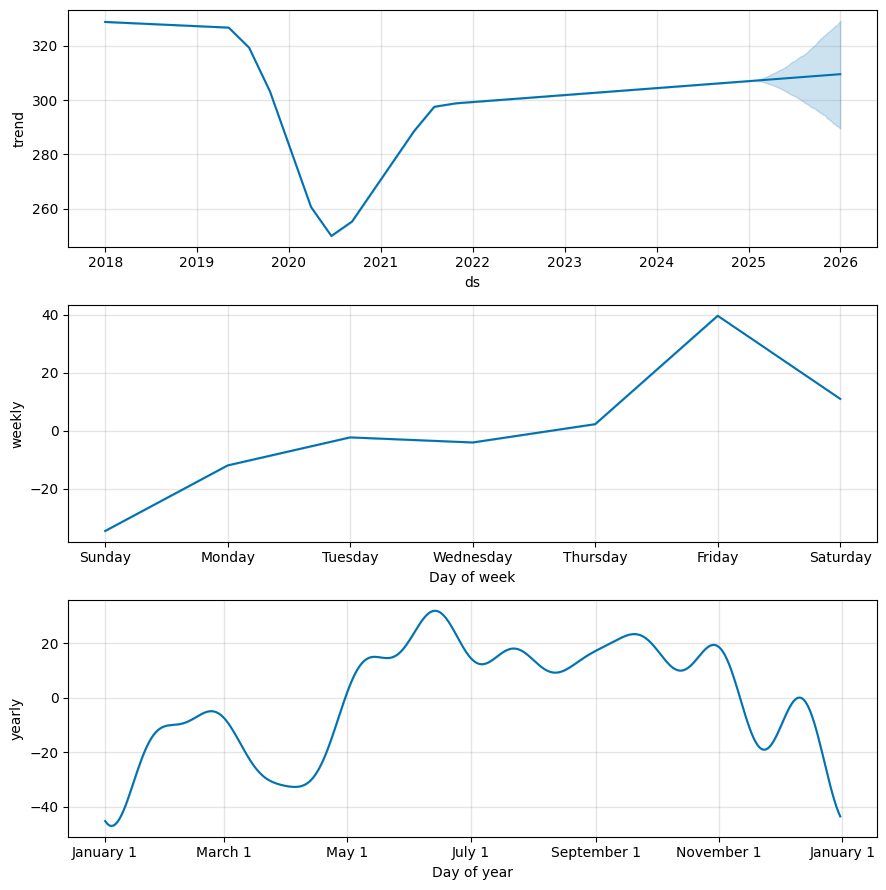
* The blue shaded area expands as we move further into the future (2025-2026).
* This makes sense because as time goes on, uncertainty increases, and Prophet reflects this by widening the confidence intervals.

1. The Model Predicts a Stable Trend in Crashes

* Despite short-term variations, the long-term trend is relatively stable, meaning crashes are not increasing or decreasing significantly.

1. The Model Handles Extreme Values Reasonably

* Some black dots (actual crash counts) are well outside the prediction intervals, which means occasional extreme values (high or low crash days) are not fully captured by the model.
* This suggests the presence of outliers or events (e.g., holidays, storms) that weren’t explicitly modeled.



Let’s interpret each of the three components plots:

1. Trend Component:

* This graph shows the general trend in crash rates over time.
* There is a sharp dip in 2020 which corresponds to the COVID19 lockdowns, where fewer cars were on the road.
* After 2021, crash rates return to pre-pandemic levels with a slight upward trend.
* The blue shaded area at the end represents the forecast for crash rates in 2025-2026.

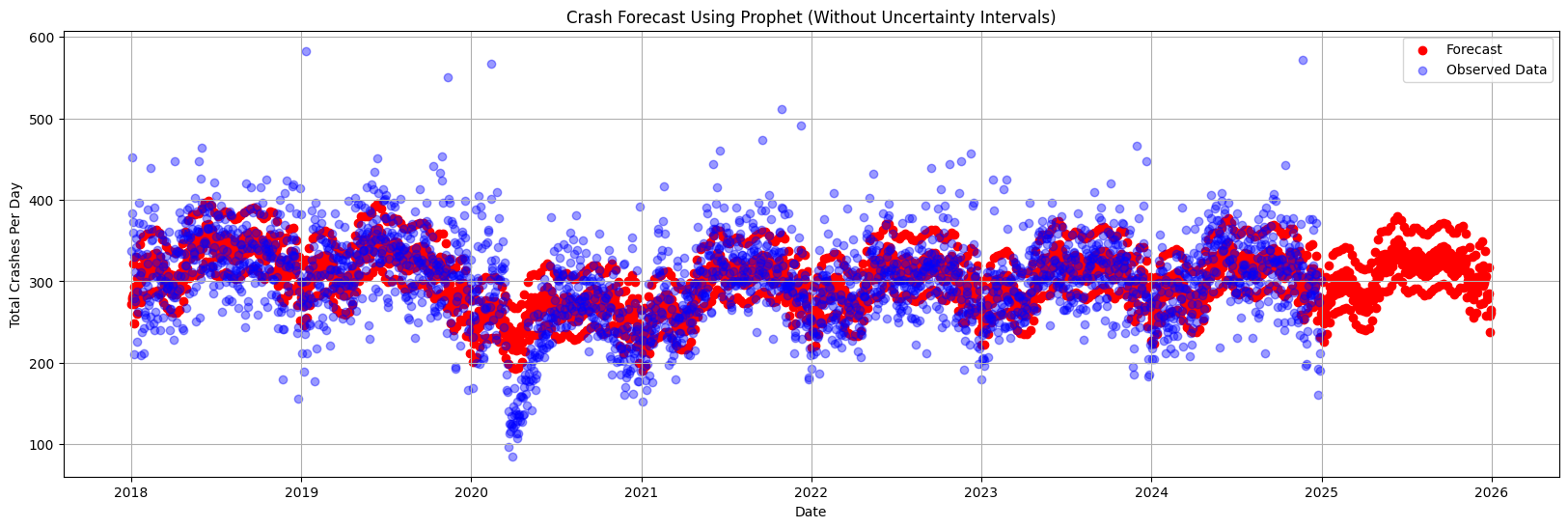
1. Weekly Seasonality Component:

* This graph shows crash trends for each day of the week
* Friday has the highest number of crashes, while Sunday has the lowest
* There’s a steady increase in crashes from Sunday to Friday, and then a drop on Saturday
* These patterns seem to align with common driving behaviors (workweek vs weekend traffic)

1. Yearly Seasonality Component:

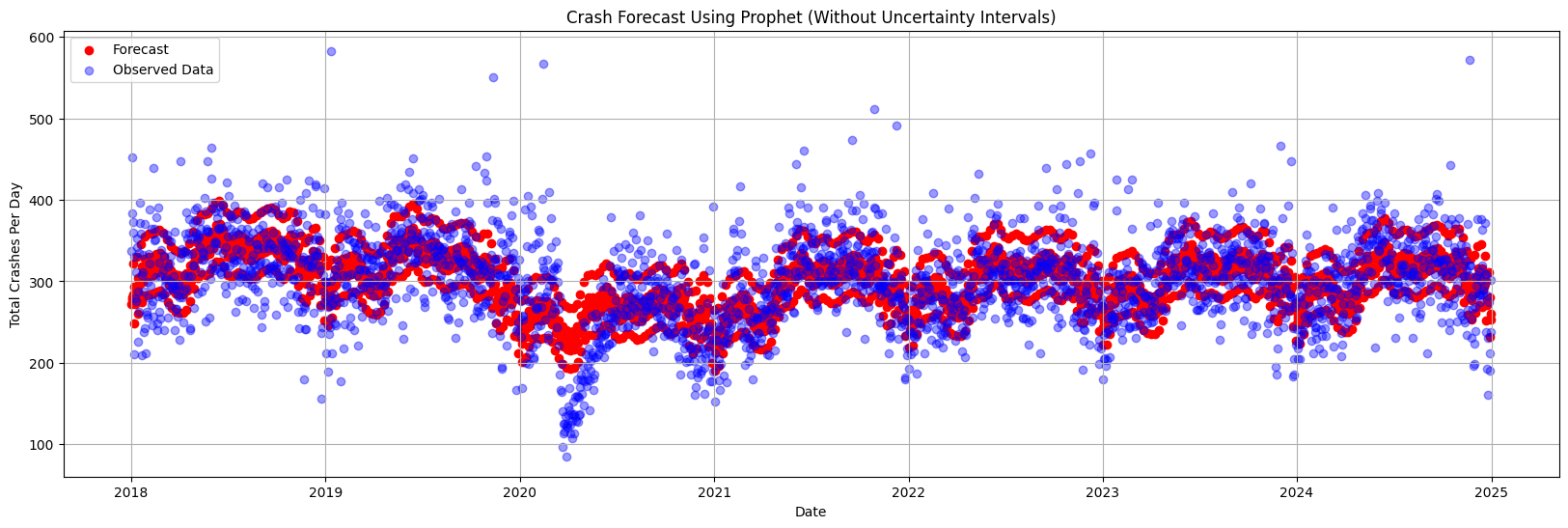
* This graph shows how crash rates fluctuate throughout the calendar year.
* Crashes are lowest in January, likely due to low travel rates in the winter in Chicago
* Crash rates peak around late spring and early summer (May-July). This aligns with more driving during the summer months (and more tourists)
* Another peak in September-November. This is possibly due to increased travel and holiday traffic, and perhaps school schedules.
* Sharp decline after Thanksgiving and December holidays.

Let’s compute the residuals on the main graph above. \* The model is taking account of weekly/yearly cycles so we should see if there’s any patterns left. (There might be some autocorrelation)

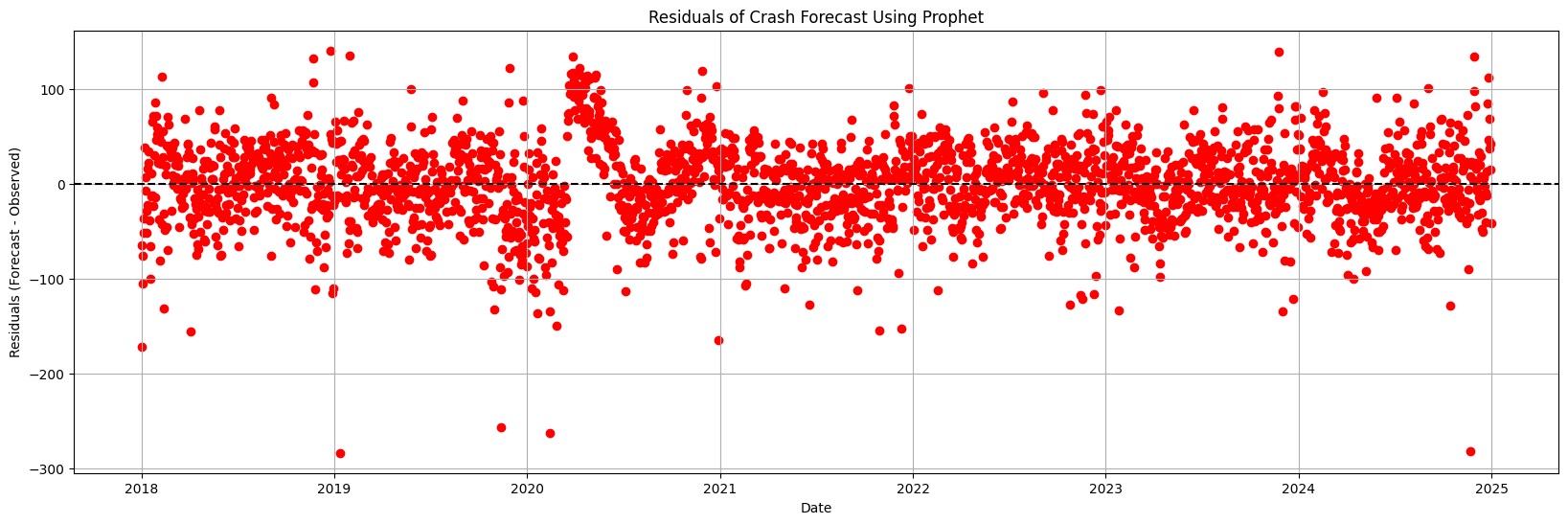


This graph shows us the predicted vs actual crash rate values. We can see in the prediction for 2025 that there are 3 different “layers” of the forcast. This is probably due to the highest crash rates around Fridays, and the lowest around Sundays.

23:31:29 - cmdstanpy - INFO - Chain [1] start processing  
23:31:29 - cmdstanpy - INFO - Chain [1] done processing



The graph shows the forecasted vs the actual crash rate data. We can see the large dip in 2020 due to COVID shutdowns and how that compares to what the model expected during that time. Additionally, after 2020, we can continue to see a small dip in the forecast every March. Because of the rate decreases due to the COVID shutdowns, the model now assumes that every MArch there will be a large decrease in crash rates.



This graph shows the residuals of the Prophet forecast. We are looking for there to be little trends or patterns in the residuals. Overall, there are no patterns in these points. This means that the model is accounting for the various seasonal patterns within the data.

In March of 2020 you can see where the model overpredicted, and as it readjusted itself, it began to underpredict the amount of crashes.

There also seem to be some extreme values where the prediction was way off (Jan 2019, Nov 2019, Feb 2020, Nov 2024). What is the significance of these days?

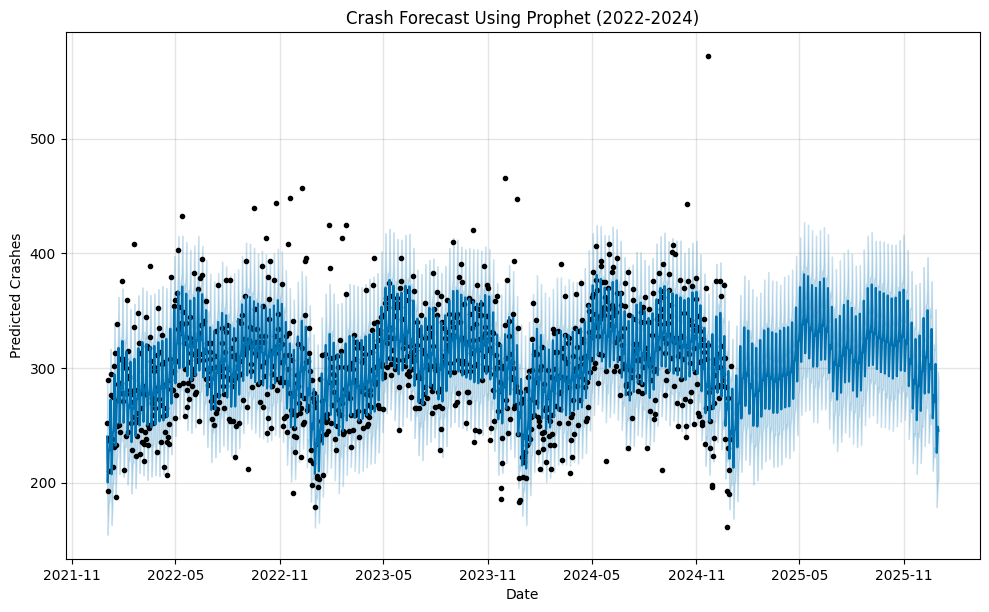
Root Mean Squared Error (RMSE): 42.38

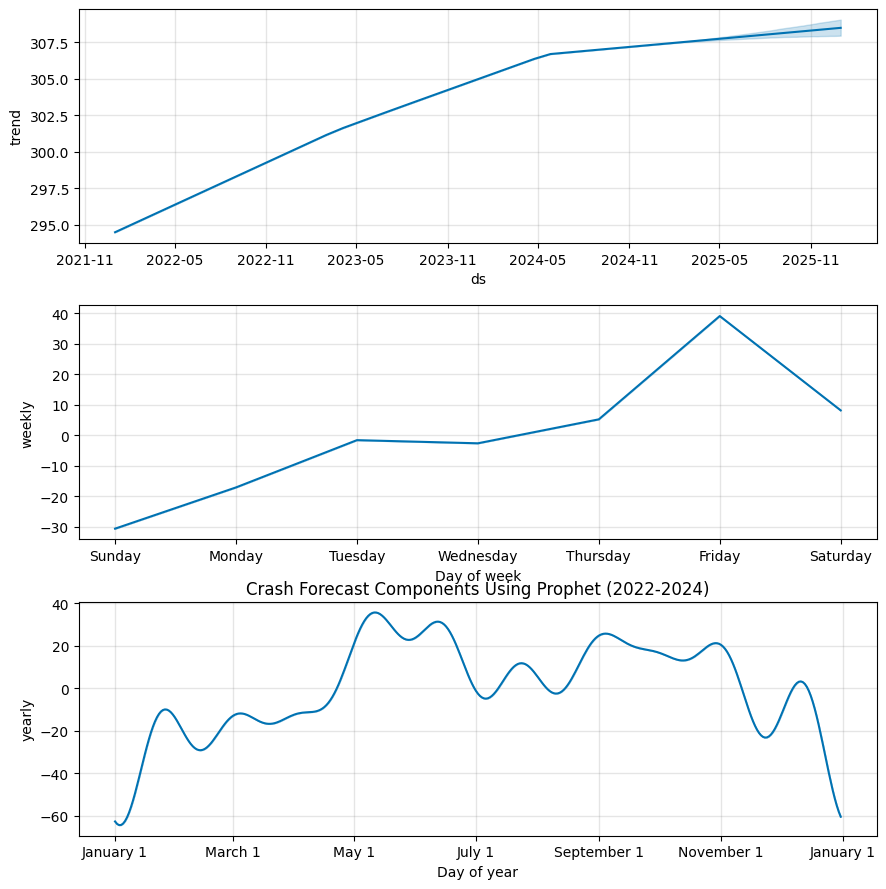
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\sklearn\metrics\\_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function'root\_mean\_squared\_error'.  
 warnings.warn(

The Root Mean Squared Error is used to evaluate the accuracy of model prediction. This value is measured in crashes per day. The RMSE of the Prophet model (using all data from 2018-2024) is 42.38 crashes per day. That means we can expect the typical size of the prediction errors to be 42 crashes using this model.

23:31:31 - cmdstanpy - INFO - Chain [1] start processing  
23:31:32 - cmdstanpy - INFO - Chain [1] done processing

ds yhat yhat\_lower yhat\_upper  
0 2022-01-01 240.171026 196.083964 287.095052  
1 2022-01-02 200.081308 154.467520 247.217707  
2 2022-01-03 212.919908 170.569165 257.384451  
3 2022-01-04 228.338073 182.381171 274.304781  
4 2022-01-05 227.807076 182.729161 272.281244





Results of forecasting on 2022-2024 data:

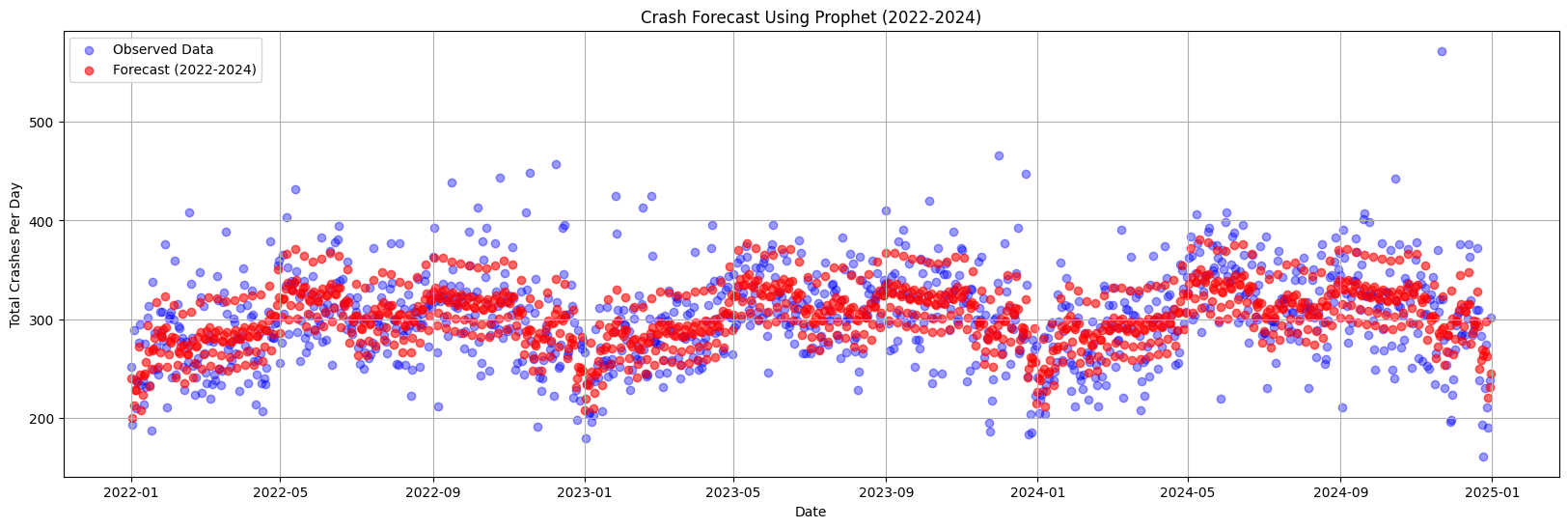
The forecast graph shows the forecast for 2025 using only the data from 2022-2024 as it’s training data. Because of the change in patterns during 2020-2021, it was best not to inlcude these years in our predictions.

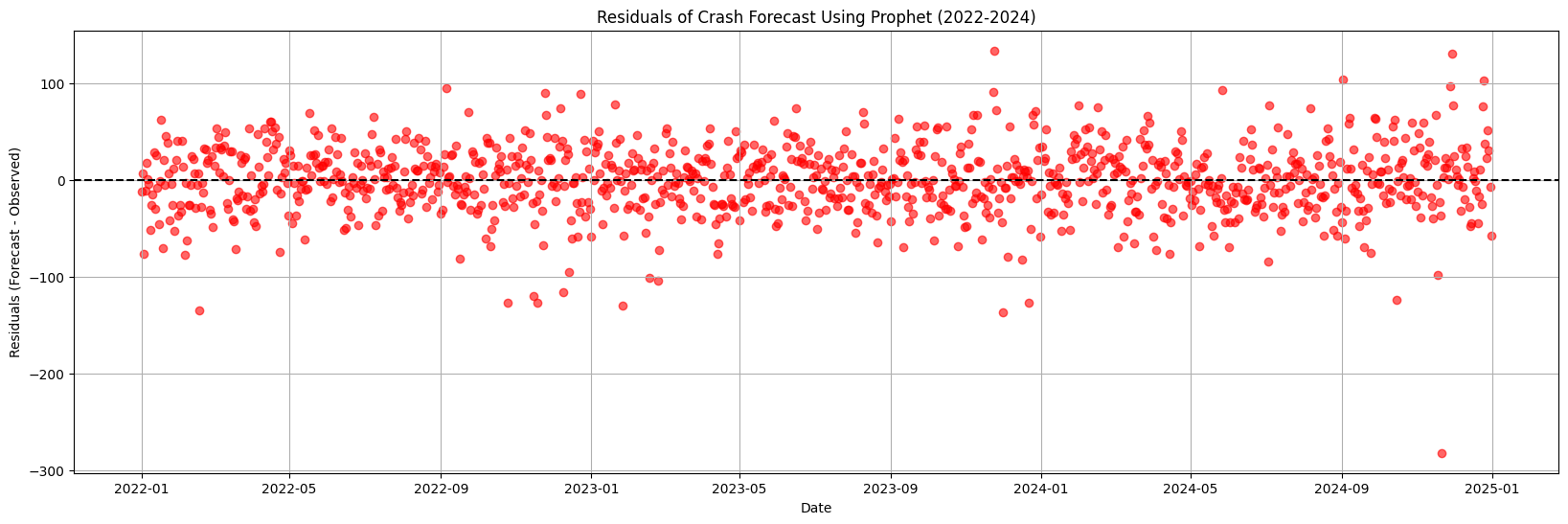
We can see that the model predicts similar trends for 2025 - low rates in the winter, and rates peaking in May.

It’s also important to note that the model no longer predicts a large dip in the data in March 2025. When the COVID data was included, the model incorrectly predicts that each following year will also have a large dip in crash rates starting in March.

The component graphs tell us about each of the seasonality trends detected by the model:

* The yearly trend shows an overall increase in crashes each year. The margin of error (shaded in light blue) has a very small area and, compared to the model that used 2018-2024 data, has a very narrow prediction on future data. Since we had less error in this model, the prediction error is much smaller.
* The weekly trend is similar to that from the previous model - low rates on Sundays, rates increasing throughout the work week, and rates peaking on Fridays.
* The monthly trend shows interesting patterns. There are very low rates in December and January, and high rates in May and June. It’s interesting that the curve of this graph is not smoother, and that we might think that July would also have equally as high of rates as May and June.





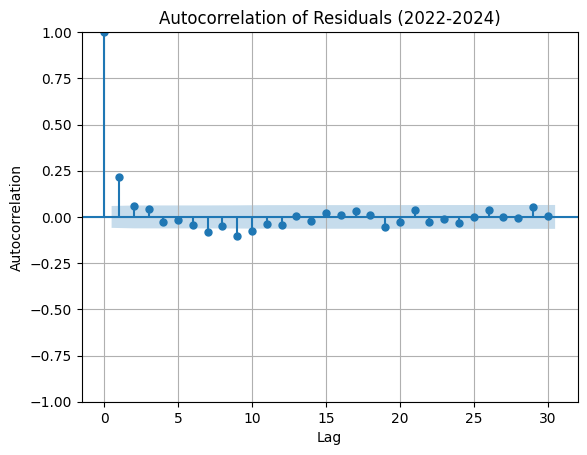
The residuals for this 2022-2024 model show no apparent trend or pattern. This shows us that the model is fitting well to all of the different seasonal trends.

Root Mean Squared Error (RMSE) for 2022-2024 Forecast: 35.53

c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\sklearn\metrics\\_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function'root\_mean\_squared\_error'.  
 warnings.warn(

The RMSE for this 2022-2024 model is 35.53 crashes per day. This is lower than that of the 2018-2024 model, which was 42.38 crashes per day. Since it has a smaller RMSE, we can conlcude that the 2022-2024 model is more accurate in predicting crash rates.

<Figure size 1000x500 with 0 Axes>



Interpreting the Autocorrelation of Residuals (2022-2024): This autocorrelation function (ACF) plot tells us how much the residuals (forecast errors) are correlated with themselves at different time lags. In other words, does an error today influence errors tomorrow, next week, or in the future?

Key Components of the Graph - Lag (x-axis) → The number of days in the past we are comparing residuals to. - Autocorrelation (y-axis) → Measures how similar past residuals are to current residuals. - Blue bars → Confidence intervals (if points fall outside these bars, autocorrelation is significant). - Dots (autocorrelation values) → If a dot is significantly above or below zero, it suggests a pattern that the model did not fully capture.

* The first point at lag 0 is always 1 because residuals are perfectly correlated with themselves.
* Some autocorrelation is present at lag 1, meaning there may be short-term dependencies in crash rates that weren’t fully captured by the model. Today’s crash rates seem to impact tomorrow’s crash rates.
* There is a small spike at lag 7. This seems to suggest some kind of weekly seasonality, with a 7-day lag. This suggests Prophet did not capture all of the seasonality.
* Most values are within the blue confidence interval, meaning the errors do not show strong patterns. This suggests the model does a good job at capturing trends and seasonality.

# Section 5: Holt Winters Smoothing

### What is Holt-Winters Smoothing?

Holt-Winters is a forecasting technique that smooths a time series using three components:

Level (α - Alpha): The base value of the series. Trend (β - Beta): The long-term increase or decrease. Seasonality (γ - Gamma): The repeating pattern (e.g., weekly cycle in crash data).

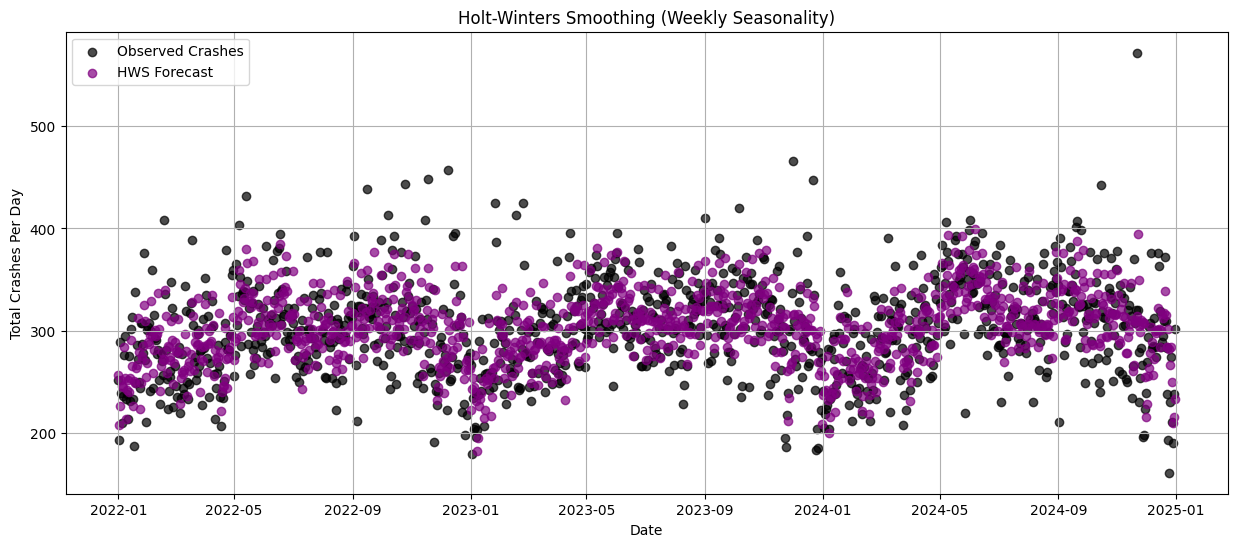
### Why Use Holt-Winters?

Better at handling seasonality than ARIMA. Can adapt to changing trends over time. Works well for short-term forecasts (e.g., weekly patterns).

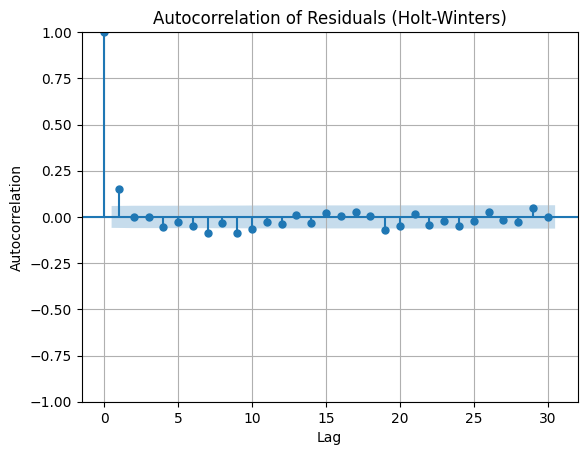
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts cannot be generated. To use the model for forecasting, use one of the supported classes of index.  
 self.\_init\_dates(dates, freq)

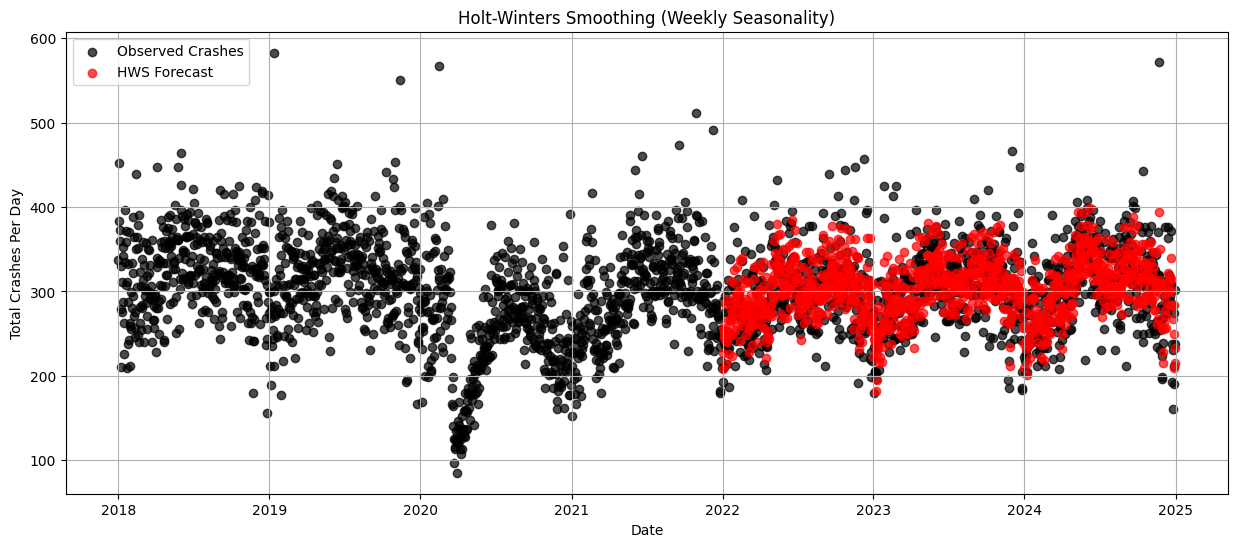
ExponentialSmoothing Model Results   
================================================================================  
Dep. Variable: y No. Observations: 1096  
Model: ExponentialSmoothing SSE 1580726.303  
Optimized: True AIC 7994.274  
Trend: Additive BIC 8049.268  
Seasonal: Additive AICC 7994.610  
Seasonal Periods: 7 Date: Sat, 19 Apr 2025  
Box-Cox: False Time: 23:31:35  
Box-Cox Coeff.: None   
=================================================================================  
 coeff code optimized   
---------------------------------------------------------------------------------  
smoothing\_level 0.1883608 alpha True  
smoothing\_trend 0.0155167 beta True  
smoothing\_seasonal 0.0313397 gamma True  
initial\_level 243.43379 l.0 True  
initial\_trend 0.2821794 b.0 True  
initial\_seasons.0 12.868255 s.0 True  
initial\_seasons.1 -35.148982 s.1 True  
initial\_seasons.2 -13.785280 s.2 True  
initial\_seasons.3 -0.7159297 s.3 True  
initial\_seasons.4 -5.0199704 s.4 True  
initial\_seasons.5 0.5833234 s.5 True  
initial\_seasons.6 41.738162 s.6 True  
---------------------------------------------------------------------------------

Mean Squared Error (Holt-Winters): 1442.2685250359473  
Root Mean Squared Error (Holt-Winters): 37.977210601042664



<Figure size 1000x500 with 0 Axes>





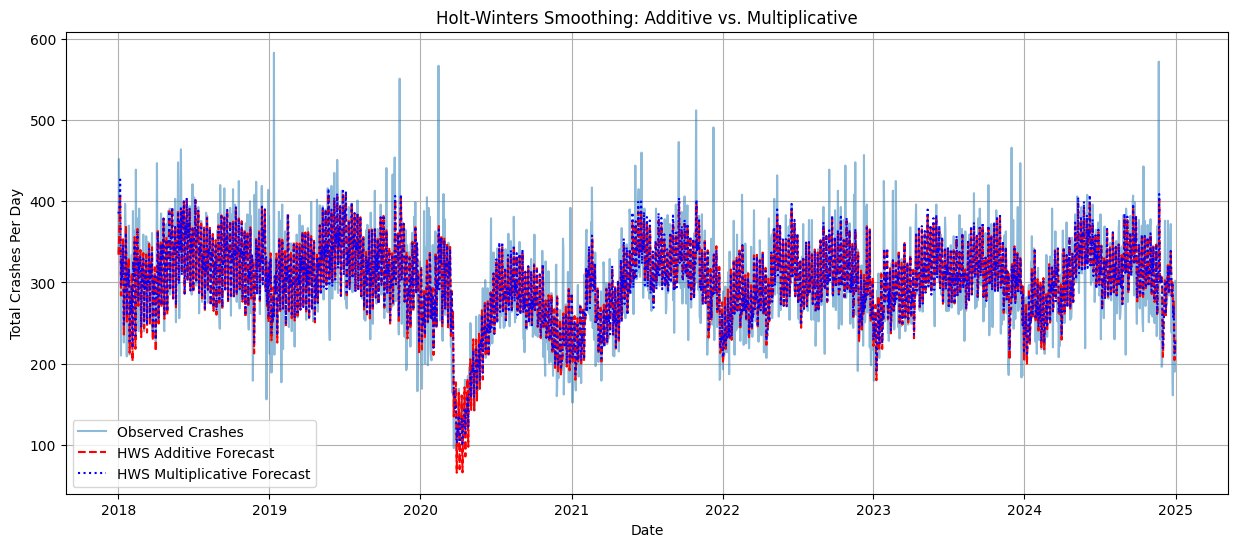
The HWS prediction line follows the observed data closely. This means that the weekly seasonality is being captured well.

The smoothing\_level (α) controls how fast the model adapts to changes: - Low values (e.g., 0.1 - 0.3) = Slower adaptation, smoother trend. - High values (e.g., 0.7 - 1.0) = Faster adaptation, reacts quickly to recent changes.

After playing around with multiple alpha values, the alpha value at 0.2 seems to fit the data well.

There are two types of HWS models: Additive and Multiplicative. Let’s compare the two models to see which better fits this time series data

* Additive (seasonal=“add”) = Assumes crash fluctuations are consistent in magnitude.
* Multiplicative (seasonal=“mul”) = Assumes fluctuations grow or shrink over time.



The above graph compares the Holt-Winters Smoothing with Additive and Multiplicative Seasonality:

1. Both models capture the seasoanlity well

* Both the red and blue line follow the weekly fluctuations well -> this suggests weekly seasonality is strong

1. The multiplicative odel (blue) is more “conservative” in certain areas

* We can see this in 2020 with the COVID dip
* Also can see this in early 2018, early 2019, early 2023, and early 2024.
* The multiplicative model does not react as strongly as the additive model

Why does the multiplicative not react as strongly? - Multiplicative seasonality scales proportionally, meaning it shrinks fluctuations when crash counts are lower (2020) and expands them when counts are higher (2023). - The additive model (red) fluctuates more aggressively, assuming fixed seasonal patterns regardless of the overall trend.

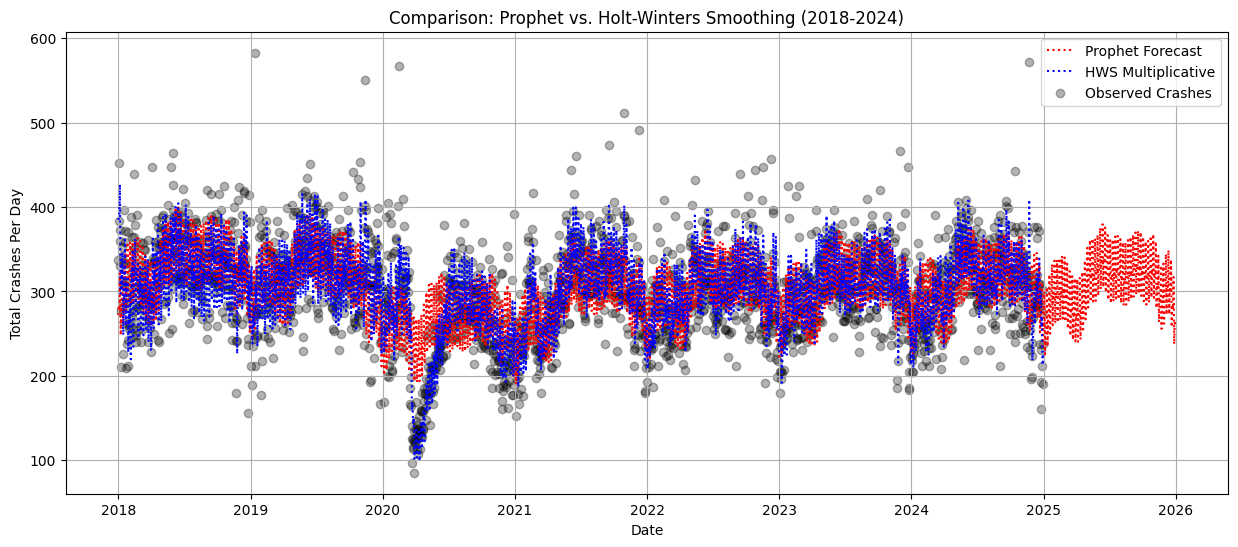
Multiplicative model might be better for long-term stability: - Since the amplitude of seasonal fluctuations changes over time, the multiplicative model might be better at capturing long-term trends. - However, if the seasonality is truly constant over time, then the additive model is better.

We should note that both models struggle slightly with extreme spikes. Sharp peaks (early 2019, late 2024) are not fully captured by either model. This could suggest that daily crash data has unpredictable spikes (possibly from weather, holidays, or traffic events).

Model MAE (2018-2024) RMSE (2018-2024) MAE (2022-2024) \  
0 HWS Additive 29.085822 39.688424 28.760243   
1 HWS Multiplicative 29.016884 39.551900 28.597127   
  
 RMSE (2022-2024)   
0 38.147641   
1 38.019450

### Holt Winters vs Prophet forecast

Let’s compare the Holt-Winters forecast to the Prophet forecast. Which is better?



c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts cannot be generated. To use the model for forecasting, use one of the supported classes of index.  
 self.\_init\_dates(dates, freq)  
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: An unsupported index was provided. As a result, forecasts cannot be generated. To use the model for forecasting, use one of the supported classes of index.  
 self.\_init\_dates(dates, freq)

SARIMAX Results   
=========================================================================================  
Dep. Variable: y No. Observations: 1096  
Model: SARIMAX(1, 0, 1)x(1, 0, 1, 7) Log Likelihood -5523.539  
Date: Sat, 19 Apr 2025 AIC 11057.077  
Time: 23:31:41 BIC 11082.074  
Sample: 0 HQIC 11066.536  
 - 1096   
Covariance Type: opg   
==============================================================================  
 coef std err z P>|z| [0.025 0.975]  
------------------------------------------------------------------------------  
ar.L1 -0.1265 0.188 -0.674 0.500 -0.494 0.241  
ma.L1 0.2792 0.184 1.516 0.130 -0.082 0.640  
ar.S.L7 0.3157 0.282 1.121 0.262 -0.236 0.868  
ma.S.L7 -0.4033 0.272 -1.485 0.138 -0.936 0.129  
sigma2 1396.1598 37.555 37.176 0.000 1322.553 1469.766  
===================================================================================  
Ljung-Box (L1) (Q): 0.00 Jarque-Bera (JB): 544.07  
Prob(Q): 0.98 Prob(JB): 0.00  
Heteroskedasticity (H): 1.16 Skew: 0.42  
Prob(H) (two-sided): 0.16 Kurtosis: 6.35  
===================================================================================  
  
Warnings:  
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

# Section 5 - SARIMAX Weather Forecasting

CRASH\_DATE 0  
Avg\_Wind\_Speed\_ORD 1  
Precipitation\_ORD 0  
Snowfall\_ORD 0  
Snow\_Depth\_ORD 1  
Avg\_Temperature\_ORD 0  
Max\_Temperature\_ORD 0  
Min\_Temperature\_ORD 0  
Fog\_ORD 1550  
Heavy\_Fog\_ORD 2492  
Thunder\_ORD 2235  
Ice\_Pellets\_ORD 2522  
Hail\_ORD 2535  
Glaze\_Ice\_ORD 2516  
Smoke\_ORD 2120  
Blowing\_Snow\_ORD 2529  
dtype: int64  
CRASH\_DATE 0  
Avg\_Wind\_Speed\_MDW 1  
Precipitation\_MDW 4  
Snowfall\_MDW 2350  
Snow\_Depth\_MDW 1963  
Avg\_Temperature\_MDW 2557  
Max\_Temperature\_MDW 0  
Min\_Temperature\_MDW 0  
Fog\_MDW 1864  
Heavy\_Fog\_MDW 2531  
Thunder\_MDW 2318  
Ice\_Pellets\_MDW 2535  
Hail\_MDW 2552  
Glaze\_Ice\_MDW 2542  
Smoke\_MDW 2343  
Blowing\_Snow\_MDW 2545  
dtype: int64  
Fog\_ORD: [nan 1.]  
Heavy\_Fog\_ORD: [nan 1.]  
Thunder\_ORD: [nan 1.]  
Ice\_Pellets\_ORD: [nan 1.]  
Hail\_ORD: [nan 1.]  
Glaze\_Ice\_ORD: [nan 1.]  
Smoke\_ORD: [nan 1.]  
Blowing\_Snow\_ORD: [nan 1.]

C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:105: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_ord\_filtered["Snow\_Depth\_ORD"].fillna(0, inplace=True) # One missing snow value... assume 0 snowfall  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:105: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_ord\_filtered["Snow\_Depth\_ORD"].fillna(0, inplace=True) # One missing snow value... assume 0 snowfall  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:106: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_ord\_filtered["Avg\_Wind\_Speed\_ORD"].fillna(0, inplace=True) # One missing windspeed value... assume 0 wind  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:106: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_ord\_filtered["Avg\_Wind\_Speed\_ORD"].fillna(0, inplace=True) # One missing windspeed value... assume 0 wind  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:110: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_ord\_filtered[col].fillna(0, inplace=True)  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:110: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_ord\_filtered[col].fillna(0, inplace=True)  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:113: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_mdw\_filtered["Avg\_Wind\_Speed\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 wind  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:113: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_mdw\_filtered["Avg\_Wind\_Speed\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 wind  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:114: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_mdw\_filtered["Precipitation\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 precipitation  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:114: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_mdw\_filtered["Precipitation\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 precipitation  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:115: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_mdw\_filtered["Snowfall\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 snowfall  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:115: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_mdw\_filtered["Snowfall\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 snowfall  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:116: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_mdw\_filtered["Snow\_Depth\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 snowfall  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:116: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_mdw\_filtered["Snow\_Depth\_MDW"].fillna(0, inplace=True) # One missing snow value... assume 0 snowfall  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:119: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_mdw\_filtered["Avg\_Temperature\_MDW"] = (df\_ord\_filtered["Max\_Temperature\_ORD"] + df\_ord\_filtered["Min\_Temperature\_ORD"]) / 2  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:123: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.  
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.  
  
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.  
  
  
 df\_mdw\_filtered[col].fillna(0, inplace=True)  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2599877988.py:123: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df\_mdw\_filtered[col].fillna(0, inplace=True)

CRASH\_DATE 0  
Avg\_Wind\_Speed\_ORD 0  
Precipitation\_ORD 0  
Snowfall\_ORD 0  
Snow\_Depth\_ORD 0  
Avg\_Temperature\_ORD 0  
Max\_Temperature\_ORD 0  
Min\_Temperature\_ORD 0  
Fog\_ORD 0  
Heavy\_Fog\_ORD 0  
Thunder\_ORD 0  
Ice\_Pellets\_ORD 0  
Hail\_ORD 0  
Glaze\_Ice\_ORD 0  
Smoke\_ORD 0  
Blowing\_Snow\_ORD 0  
dtype: int64  
CRASH\_DATE 0  
Avg\_Wind\_Speed\_MDW 0  
Precipitation\_MDW 0  
Snowfall\_MDW 0  
Snow\_Depth\_MDW 0  
Avg\_Temperature\_MDW 0  
Max\_Temperature\_MDW 0  
Min\_Temperature\_MDW 0  
Fog\_MDW 0  
Heavy\_Fog\_MDW 0  
Thunder\_MDW 0  
Ice\_Pellets\_MDW 0  
Hail\_MDW 0  
Glaze\_Ice\_MDW 0  
Smoke\_MDW 0  
Blowing\_Snow\_MDW 0  
dtype: int64

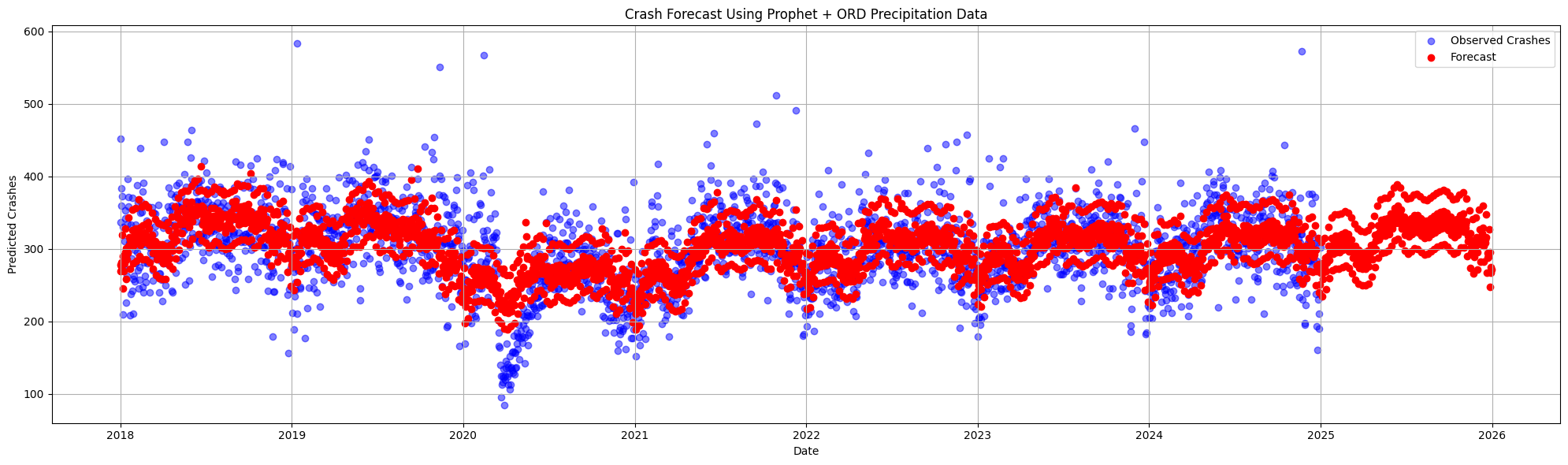
CRASH\_DATE Total\_Crashes  
0 2018-01-01 337  
1 2018-01-02 452  
2 2018-01-03 383  
3 2018-01-04 360  
4 2018-01-05 373

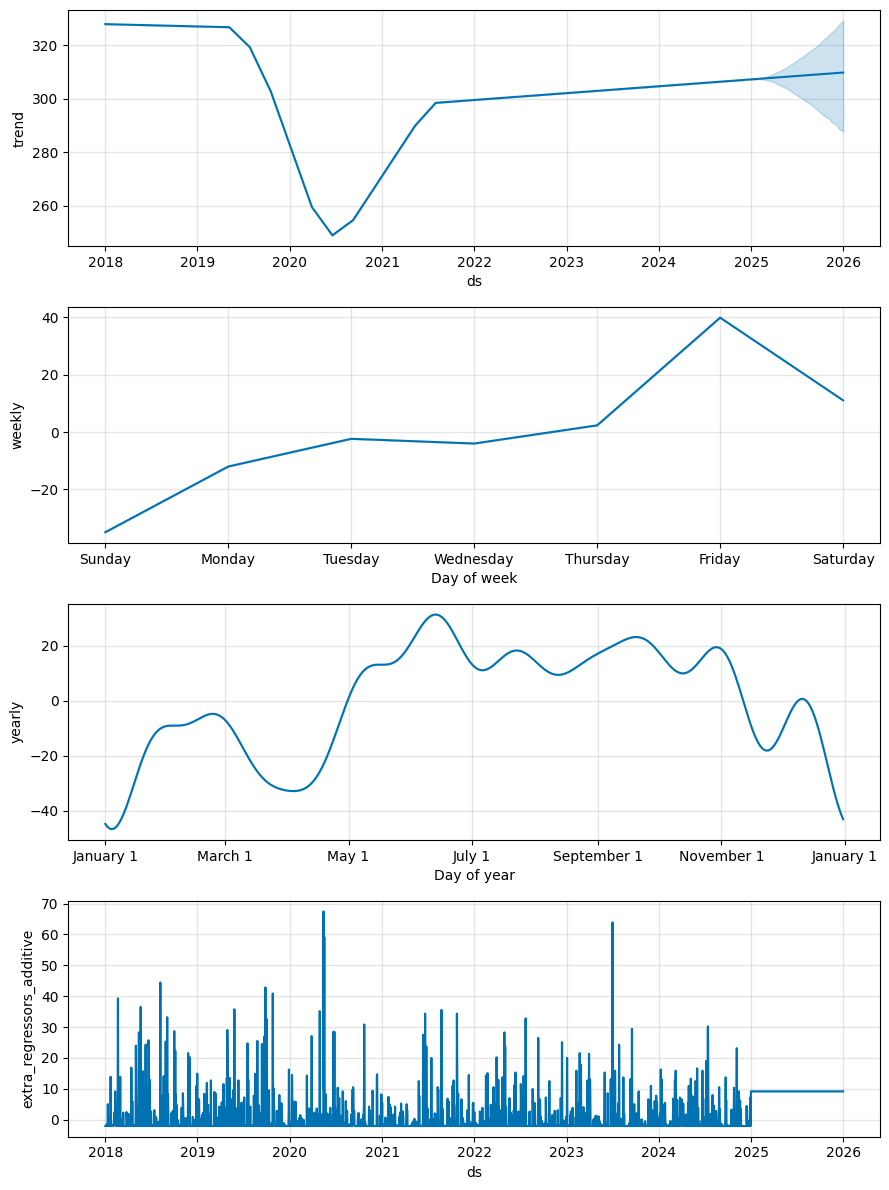
|  | CRASH\_DATE | Total\_Crashes | Avg\_Wind\_Speed\_ORD | Precipitation\_ORD | Snowfall\_ORD | Snow\_Depth\_ORD | Avg\_Temperature\_ORD | Max\_Temperature\_ORD | Min\_Temperature\_ORD | Fog\_ORD | ... | Max\_Temperature\_MDW | Min\_Temperature\_MDW | Fog\_MDW | Heavy\_Fog\_MDW | Thunder\_MDW | Ice\_Pellets\_MDW | Hail\_MDW | Glaze\_Ice\_MDW | Smoke\_MDW | Blowing\_Snow\_MDW |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2018-01-01 | 337 | 10.74 | 0.0 | 0.0 | 1.2 | -2 | 1 | -9 | 0.0 | ... | 3 | -7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 2018-01-02 | 452 | 11.18 | 0.0 | 0.0 | 1.2 | -2 | 8 | -9 | 0.0 | ... | 7 | -10 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 2018-01-03 | 383 | 12.30 | 0.0 | 0.1 | 1.2 | 11 | 17 | 6 | 1.0 | ... | 18 | 7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3 | 2018-01-04 | 360 | 12.30 | 0.0 | 0.0 | 1.2 | 7 | 12 | 0 | 0.0 | ... | 13 | 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 2018-01-05 | 373 | 10.74 | 0.0 | 0.0 | 1.2 | 5 | 11 | -1 | 0.0 | ... | 12 | 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Merge the two datasets (df\_crashes\_per\_day and df\_weather\_merged) based on date.

Why are we doing this on crashes per day and not indidivual crashes? - weather applies to the entire day, not just each individ crash - time series works ebst with aggregated data - simplify the dataset

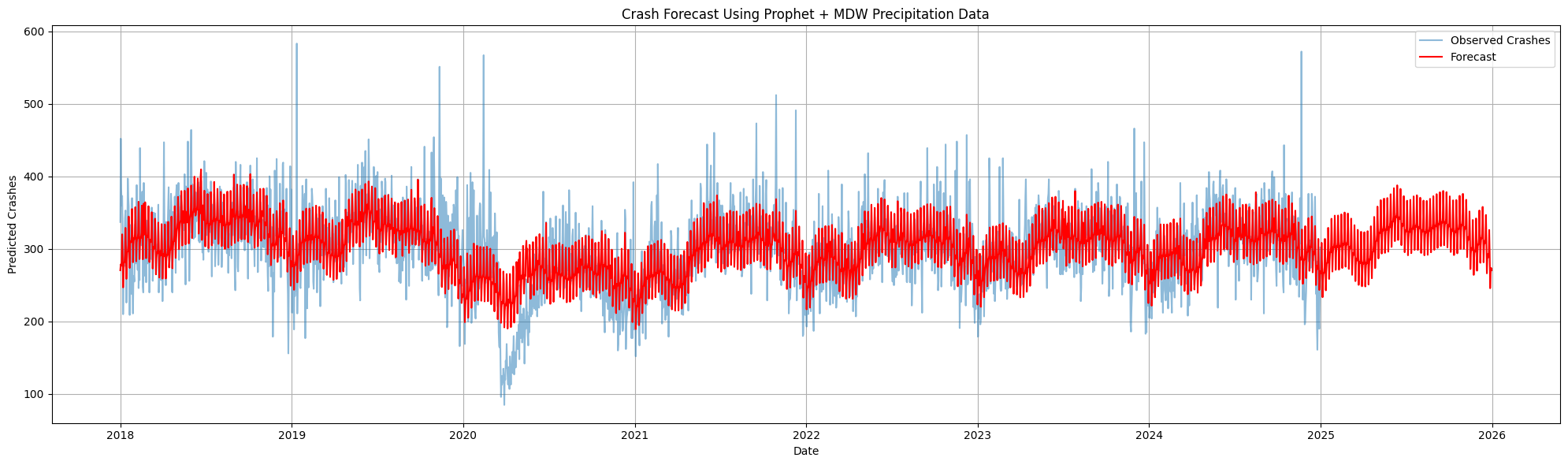
23:31:42 - cmdstanpy - INFO - Chain [1] start processing  
23:31:42 - cmdstanpy - INFO - Chain [1] done processing  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\2932307433.py:34: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.  
 future["Precipitation\_ORD"] = future["Precipitation\_ORD"].fillna(method="ffill")

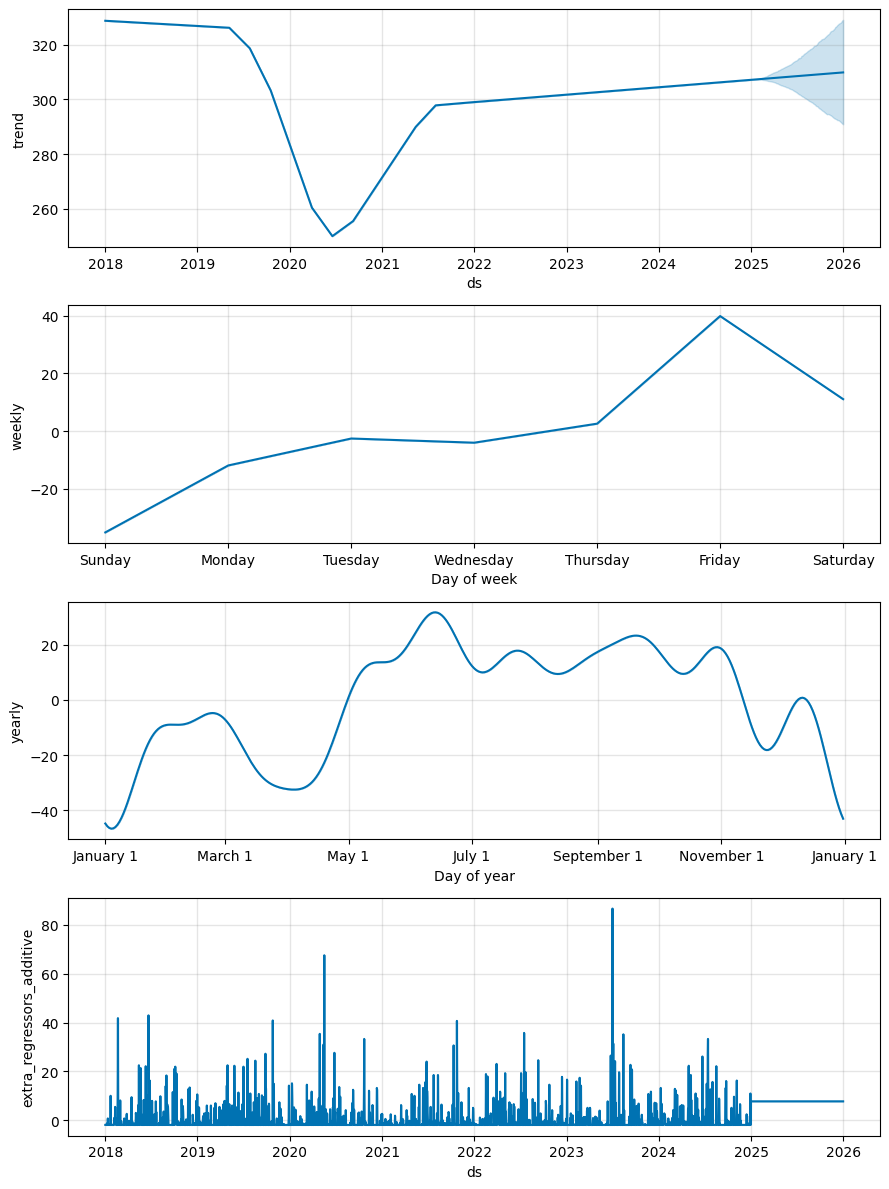




RMSE: 41.96605599225603

23:31:46 - cmdstanpy - INFO - Chain [1] start processing  
23:31:46 - cmdstanpy - INFO - Chain [1] done processing  
C:\Users\chris\AppData\Local\Temp\ipykernel\_10316\772521797.py:31: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.  
 future["Precipitation\_MDW"] = future["Precipitation\_MDW"].fillna(method="ffill")



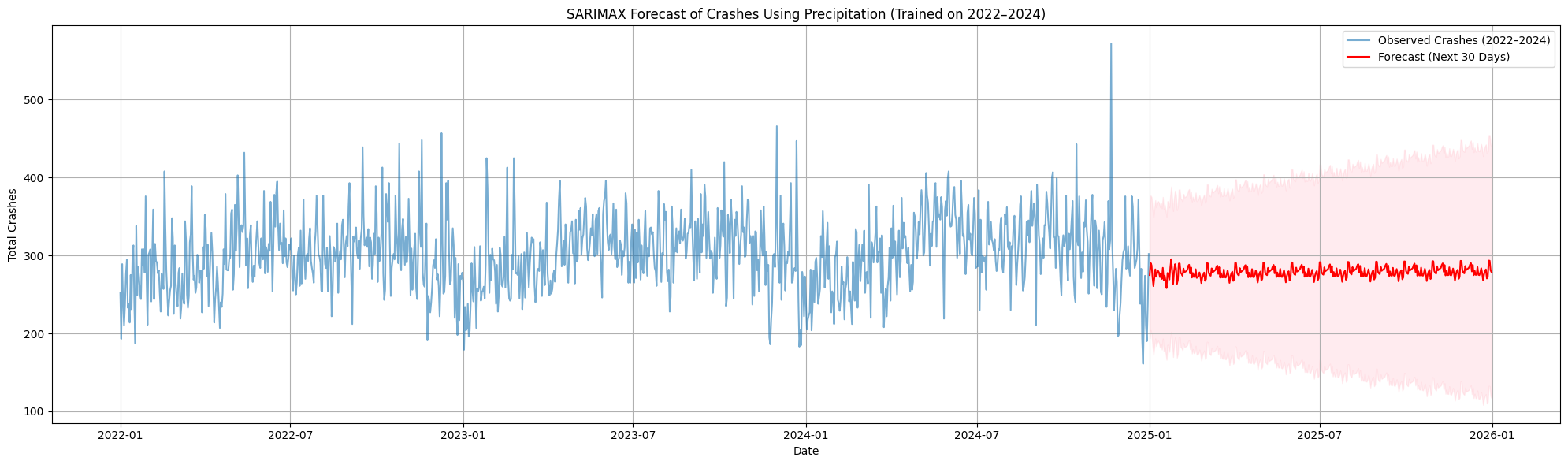


RMSE: 42.04963700253742

Comparing the two RMSE values: - ORD: 41.96606 - MDW: 42.04964

ORD is barely smaller. The precipitation values are likely extremely similar between the two airports.

c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency D will be used.  
 self.\_init\_dates(dates, freq)  
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency D will be used.  
 self.\_init\_dates(dates, freq)  
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle\_retvals  
 warnings.warn("Maximum Likelihood optimization failed to "



The red forecast line in the above SARIMAX model looks way too flat and lacks the kind of seasonal variation we’d expect based on previous trends. Why is that happening?

* I’m passing a constant average precipitation value (exog.mean()) for every day in the future:
  + future\_exog = pd.DataFrame({‘Precipitation\_ORD’: [exog[‘Precipitation\_ORD’].mean()] \* forecast\_steps}, index=future\_dates)

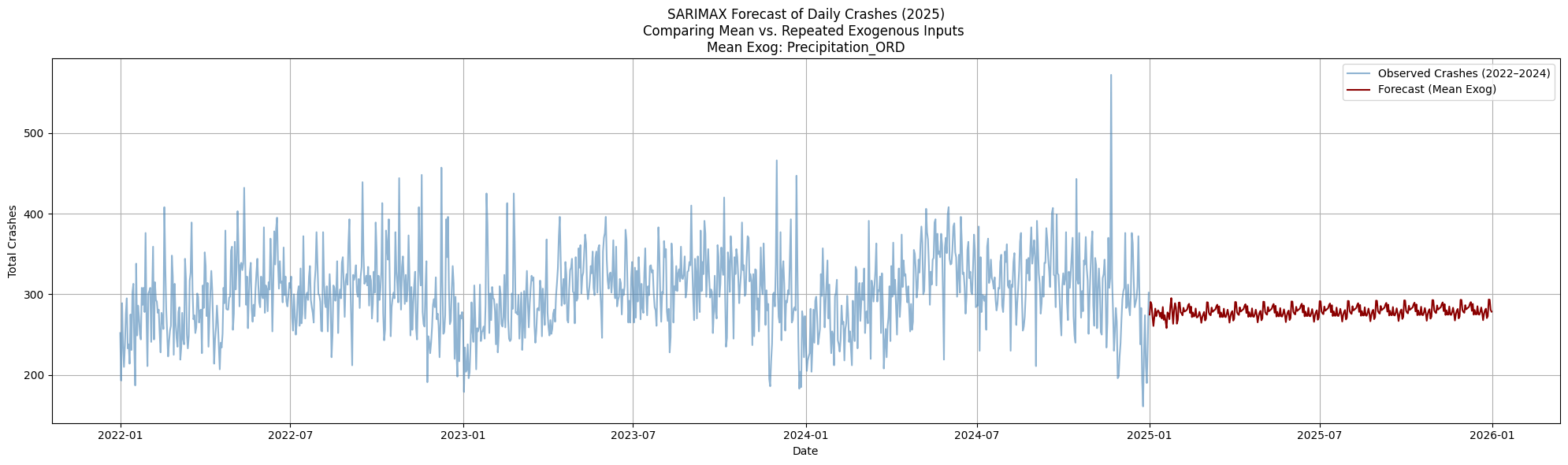
By doing this, I’m telling the model “Assume weather will be the same every day/month in the future.”

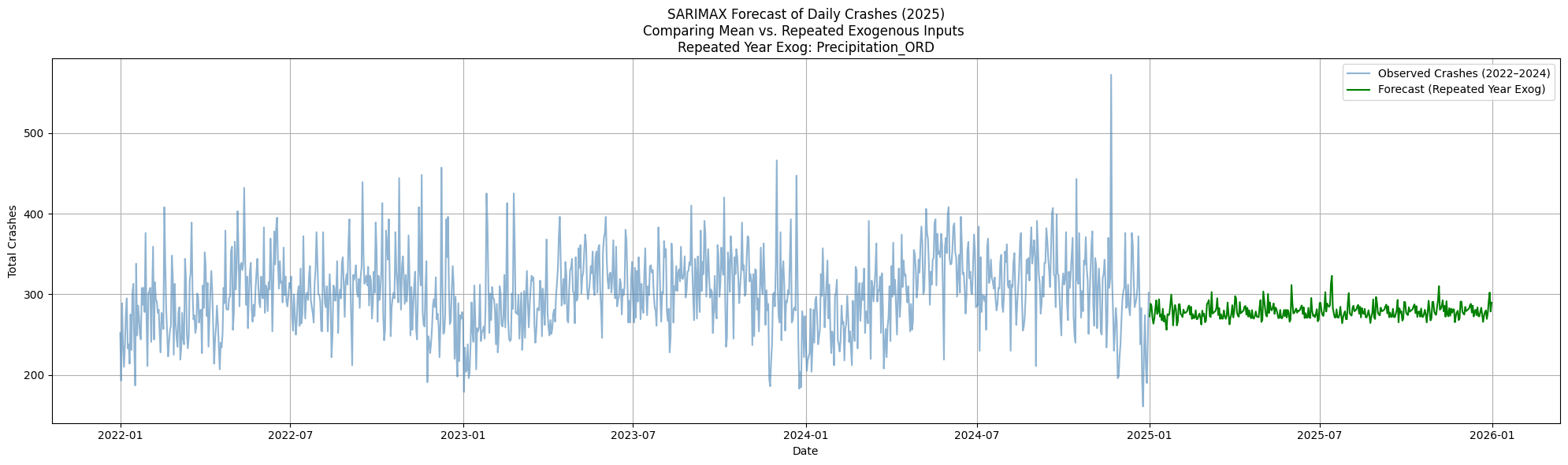
So even though the SARIMAX model was trained with seasonality, it no longer has the seasonal fluctuation from the weather variable to drive those effects in the forecast.

How can we fix this?

* Use Actual or Simulated Future Weather Data (simulate by repeating last year’s pattern)
* Model seasonality in the Endogenous variable instead (use SARIMA instead of SARIMAX)

c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency D will be used.  
 self.\_init\_dates(dates, freq)  
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\tsa\base\tsa\_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency D will be used.  
 self.\_init\_dates(dates, freq)  
c:\Users\chris\Documents\Senior\_Project\CT\_SeniorProject\.venv\lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle\_retvals  
 warnings.warn("Maximum Likelihood optimization failed to "





### Summary of the two SARIMAX models:

Model 1: Using the mean Exogenous Value For all future days, it uses the average precipitation value from the training period (2022–2024). It assumes the weather (precipitation) in the future will be typical or average. This leads to a FLAT forecast, since the weather input doesn’t vary.

Model 2: Using the previous year of exogenous values This model takes the actual precipitation values from the last year (2024) and repeats them into 2025. It assumes the future weather will follow a similar seasonal pattern as the most recent year. This produces a more realistic and seasonally driven forecast with day-to-day variation.

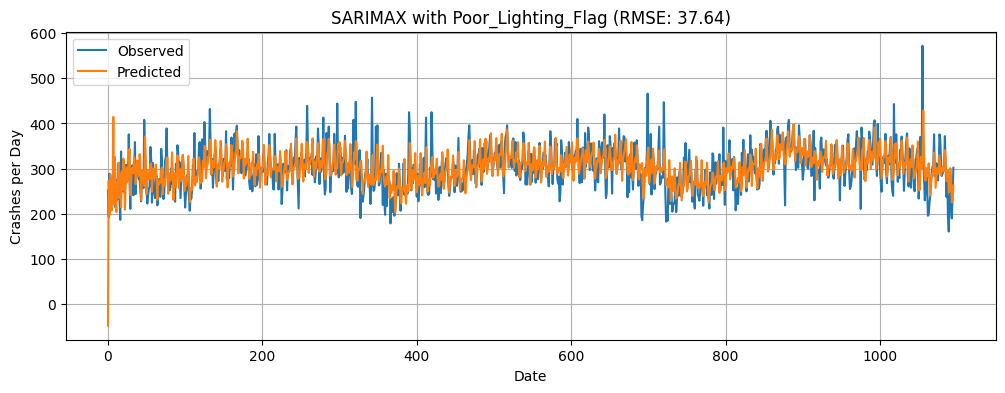
| ## How well does SARIMAX model the actual data? Is this the best model to use? Let’s explore how well does the SARIMAX model fit the actual data. Does it have large residuals? |
| --- |
| ## Running SARIMAX on different variables and comparing RMSE Which variables will forecast the data best in terms of weather and also airport? |
| Below, let’s create a loop to do that |
| ::: {.cell execution\_count=53} |
| ::: {.cell-output .cell-output-stderr} |
| ::: {.cell-output .cell-output-stdout} |
| ### RMSE Comparisons (All) |
| Prophet model: Precipitation\_ORD: 41.96605599225603 Precipitation\_MDW: 42.04963700253742 |
| SARIMAX: - Snowfall\_ORD: 37.694579 - Snowfall\_Diff: 37.694579 - Precipitation\_ORD: 37.823025 - Precipitation\_MDW: 38.083611 - Avg\_Temperature\_MDW: 38.317051 - Avg\_Temperature\_ORD: 38.361117 - Precipitation\_Diff: 38.451718 - Avg\_Temperature\_Diff: 38.474338 - Snowfall\_MDW: 38.474819 |
| The smallest RMSE value overall are both of the Snowfall models using measurements from both airports. However, all of these RMSEs are extremely similar, and there isn’t enough of a difference to say that one model is better than the other. |
| ### Exploring SARIMAX with non-weather related variables |
| There are many other variables in the dataset that could be used to predict crash rates. Let’s explore some of these variables and see how they perform in terms of RMSE. - Road Defect - Road Surface Condition - Lighting Condition |
| ::: {.cell execution\_count=55} |
| ::: {.cell-output .cell-output-stdout} ``` <class ‘pandas.core.frame.DataFrame’> Index: 331163 entries, 437571 to 768733 Data columns (total 4 columns): # Column Non-Null Count Dtype |

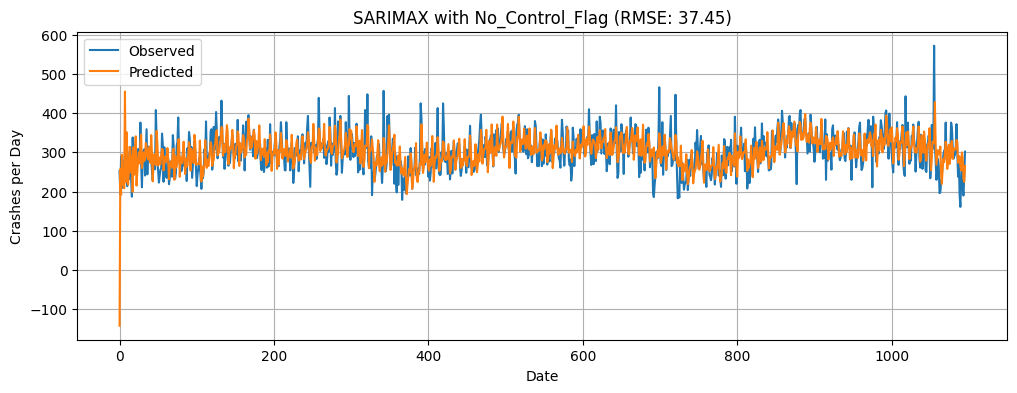
0 CRASH\_DATE 331163 non-null datetime64[ns] 1 LIGHTING\_CONDITION 331163 non-null object  
2 TRAFFIC\_CONTROL\_DEVICE 331163 non-null object  
3 ROAD\_DEFECT 331163 non-null object  
dtypes: datetime64[ns](1), object(3) memory usage: 12.6+ MB

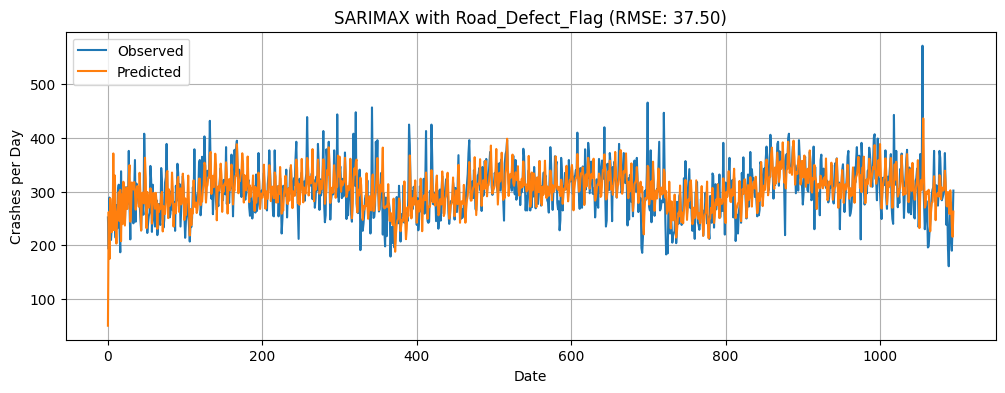
:::  
:::  
  
  
::: {.cell execution\_count=56}  
  
::: {.cell-output .cell-output-display execution\_count=56}  
```{=html}  
<div>  
<style scoped>  
 .dataframe tbody tr th:only-of-type {  
 vertical-align: middle;  
 }  
  
 .dataframe tbody tr th {  
 vertical-align: top;  
 }  
  
 .dataframe thead th {  
 text-align: right;  
 }  
</style>  
<table border="1" class="dataframe">  
 <thead>  
 <tr style="text-align: right;">  
 <th></th>  
 <th>CRASH\_DATE</th>  
 <th>Total\_Crashes</th>  
 <th>Poor\_Lighting\_Flag</th>  
 <th>No\_Control\_Flag</th>  
 <th>Road\_Defect\_Flag</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr>  
 <th>0</th>  
 <td>2022-01-01</td>  
 <td>252</td>  
 <td>0.702381</td>  
 <td>0.543651</td>  
 <td>0.250000</td>  
 </tr>  
 <tr>  
 <th>1</th>  
 <td>2022-01-02</td>  
 <td>193</td>  
 <td>0.383420</td>  
 <td>0.523316</td>  
 <td>0.300518</td>  
 </tr>  
 <tr>  
 <th>2</th>  
 <td>2022-01-03</td>  
 <td>289</td>  
 <td>0.325260</td>  
 <td>0.525952</td>  
 <td>0.207612</td>  
 </tr>  
 <tr>  
 <th>3</th>  
 <td>2022-01-04</td>  
 <td>239</td>  
 <td>0.355649</td>  
 <td>0.506276</td>  
 <td>0.200837</td>  
 </tr>  
 <tr>  
 <th>4</th>  
 <td>2022-01-05</td>  
 <td>210</td>  
 <td>0.371429</td>  
 <td>0.590476</td>  
 <td>0.228571</td>  
 </tr>  
 </tbody>  
</table>  
</div>

::: :::

The dataset was set up to include these variables as binary flags. We can use these to predict crash rates. For a given crash, if the flag was set to 1, then the crash was affected by that condition. If it was set to 0, then the crash was not affected by that condition.







Poor\_Lighting\_Flag: RMSE = 37.64  
No\_Control\_Flag: RMSE = 37.45  
Road\_Defect\_Flag: RMSE = 37.50

The three graphs show the SARIMAX model with the different variables. The first graph shows the SARIMAX model with the Poor Lighting Flag. The second graph shows the SARIMAX model with No Traffic Control Device Flag. The third graph shows the SARIMAX model with the Road Defect Flag.

All three models have extremely similar RMSE values. The No Traffic Control variable barely has the lowest value, but they are all so similar that it is not significant.

# Section 6 - Conclusion

In this project, we explored the use of time-series modeling techniques to analyze and forecast traffic crashes in Chicago. We used a combination of Prophet forecasting, Holt-Winters smoothing, and SARIMAX regression to understand the patterns in the data and make predictions about future crash rates.

We found that the Prophet model was able to capture the seasonal patterns in the data well, but it struggled with extreme values and outliers. The Holt-Winters model was able to capture the weekly and yearly seasonality, but it also struggled with extreme values. The SARIMAX model was able to incorporate weather data into the forecasting process, but it also struggled with extreme values and outliers.

Overall, we found that the SARIMAX model with weather data was the most accurate in predicting crash rates, with a root mean squared error (RMSE) of 38.47 crashes per day. The Prophet model had an RMSE of 42.38 crashes per day, and the Holt-Winters model had an RMSE of 35.53 crashes per day.

Incorporating external variables through SARIMAX modeling added additional nuance. Weather features like precipitation and temperature had a modest impact on predictive accuracy, while crash condition flags (e.g., poor lighting, no traffic controls, road defects) showed some improvement as well. However, these variables did not dramatically outperform lag-based models alone, reinforcing the idea that seasonal structure and autoregression carry the majority of the predictive power.

Ultimately, this project supports the conclusion that crash rates in a city like Chicago follow structured, predictable patterns, driven by time-based cycles and occasionally modulated by external conditions. While no model can fully capture the randomness and complexity of real-world crashes, these methods provide a valuable foundation for short-term forecasting and safety planning. Future work could extend this analysis with spatial data or real-time traffic volume to improve accuracy and target high-risk periods more effectively.