Predictive Modeling on Mosquitos Carrying the West Nile Virus

Introduction/Problem:
West Nile Virus ("WNV") first started appearing in Chicago in 2002. By 2004, the City of Chicago and the Chicago Department of Public Health ("CDPH") had established a comprehensive surveillance and control program that is still in effect today.

The goal for our project is to build a model that can accurately predict the presence of West Nile Virus given weather and testing data. In doing so, the model can also help us derive insights on potential patterns that may exist in how the virus spreads. These findings will ultimately benefit the CPDH in optimizing its resources and future strategy on containing WNV.

Note that all of this data is publicly available and was provided as part of a Kaggle competition that was run in 2015 (https://www.kaggle.com/c/predict-west-nile-virus).

Data Wrangling:
We start the process by cleaning and wrangling the two datasets we'll eventually use to train our model. These will be referred to as the "Weather" data and the "Mosquito Trap" data respectively.

Weather Data Preview:

	Station	Date	Tmax	Tmin	Tavg	Depart	DewPoint	WetBulb	Heat	Cool	...	CodeSum	Depth	Water1	SnowFall	PrecipTotal	StnPressure	SeaLevel
0	1	2007-05-01	83	50	67	14	51	56	0	2	...		0	NaN	0.0	0.00	29.10	29.82
1	2	2007-05-01	84	52	68	NaN	51	57	0	3	...		NaN	NaN	NaN	0.00	29.18	29.82
2	1	2007-05-02	59	42	51	-3	42	47	14	0	...	BR	0	NaN	0.0	0.00	29.38	30.09
3	2	2007-05-02	60	43	52	NaN	42	47	13	0	...	BR HZ	NaN	NaN	NaN	0.00	29.44	30.08
4	1	2007-05-03	66	46	56	2	40	48	9	0	...		0	NaN	0.0	0.00	29.39	30.12

Mosquito Trap Data Preview:

	Date	Address	Species	Block	Street	Trap	AddressNumberAndStreet	Latitude	Longitude	AddressAccuracy	NumMosquitos	WnvPresent
0	2007-05-29	4100 North Oak Park Avenue, Chicago, IL 60634,...	CULEX PIPIENS/RESTUANS	41	N OAK PARK AVE	T002	4100 N OAK PARK AVE, Chicago, IL	41.954690	-87.800991	9	1	0
1	2007-05-29	4100 North Oak Park Avenue, Chicago, IL 60634,...	CULEX RESTUANS	41	N OAK PARK AVE	T002	4100 N OAK PARK AVE, Chicago, IL	41.954690	-87.800991	9	1	0
2	2007-05-29	6200 North Mandell Avenue, Chicago, IL 60646, USA	CULEX RESTUANS	62	N MANDELL AVE	T007	6200 N MANDELL AVE, Chicago, IL	41.994991	-87.769279	9	1	0
3	2007-05-29	7900 West Foster Avenue, Chicago, IL 60656, USA	CULEX PIPIENS/RESTUANS	79	W FOSTER AVE	T015	7900 W FOSTER AVE, Chicago, IL	41.974089	-87.824812	8	1	0
4	2007-05-29	7900 West Foster Avenue, Chicago, IL 60656, USA	CULEX RESTUANS	79	W FOSTER AVE	T015	7900 W FOSTER AVE, Chicago, IL	41.974089	-87.824812	8	4	0

Data Wrangling Part 1: Weather Data

For the columns which denote precipitation (“Snowfall” & “PrecipTotal”), there are two values we need to replace. M denotes "Missing Data" which we should replace with NaN so that it does not interfere with our ability to perform numerical analysis on these columns. T denotes "Trace" which indicates that the value is greater than zero but less than the smallest unit of measurement (0.1 inches for Snowfall and 0.01 inches for rain). We will replace "T" with a value equal to half of that smallest unit (i.e. 0.05 inches for Snowfall and 0.005 inches for rain). At this time, we can also delete our “Water1” & “Depth” columns because these respective series are empty for our dataset.

Next, we convert all of the numerical features into Float datatypes. For the “Sunrise” and “Sunset” columns, we will convert these into datetimes. Then, with our weather conditions (“CodeSum”), we split these categories into indicator variables using Panda’s get_dummies() function.

Lastly, we take the “Station” column and merge the “Station1” and “Station2” rows for each date by using the following methodology:

1. If there are any null values in one station dataset and not the other, then the merged version will use whatever is available.
2. All numerical values will be averaged.
3. For the weather condition categories, we will include every observed value between the two stations (e.g. If Station 1 recorded "BR" for and Station 2 recorded "BR HZ", then the merged row's "CodeSum" will have "BR HZ")

Our final weather dataset now has 35 columns.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2944 entries, 0 to 2943
Data columns (total 22 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Station      2944 non-null   int64
 1   Date         2944 non-null   object
 2   Tmax         2944 non-null   int64
 3   Tmin         2944 non-null   int64
 4   Tavg         2944 non-null   object
 5   Depart       2944 non-null   object
 6   DewPoint     2944 non-null   int64
 7   WetBulb      2944 non-null   object
 8   Heat         2944 non-null   object
 9   Cool         2944 non-null   object
 10  Sunrise      2944 non-null   object
 11  Sunset       2944 non-null   object
 12  CodeSum      2944 non-null   object
 13  Depth        2944 non-null   object
 14  Water1       2944 non-null   object
 15  SnowFall     2944 non-null   object
 16  PrecipTotal  2944 non-null   object
 17  StnPressure  2944 non-null   object
 18  SeaLevel     2944 non-null   object
 19  ResultSpeed  2944 non-null   float64
 20  ResultDir    2944 non-null   int64
 21  AvgSpeed     2944 non-null   object
dtypes: float64(1), int64(5), object(16)
memory usage: 506.1+ KB
```

Initial Weather Dataset

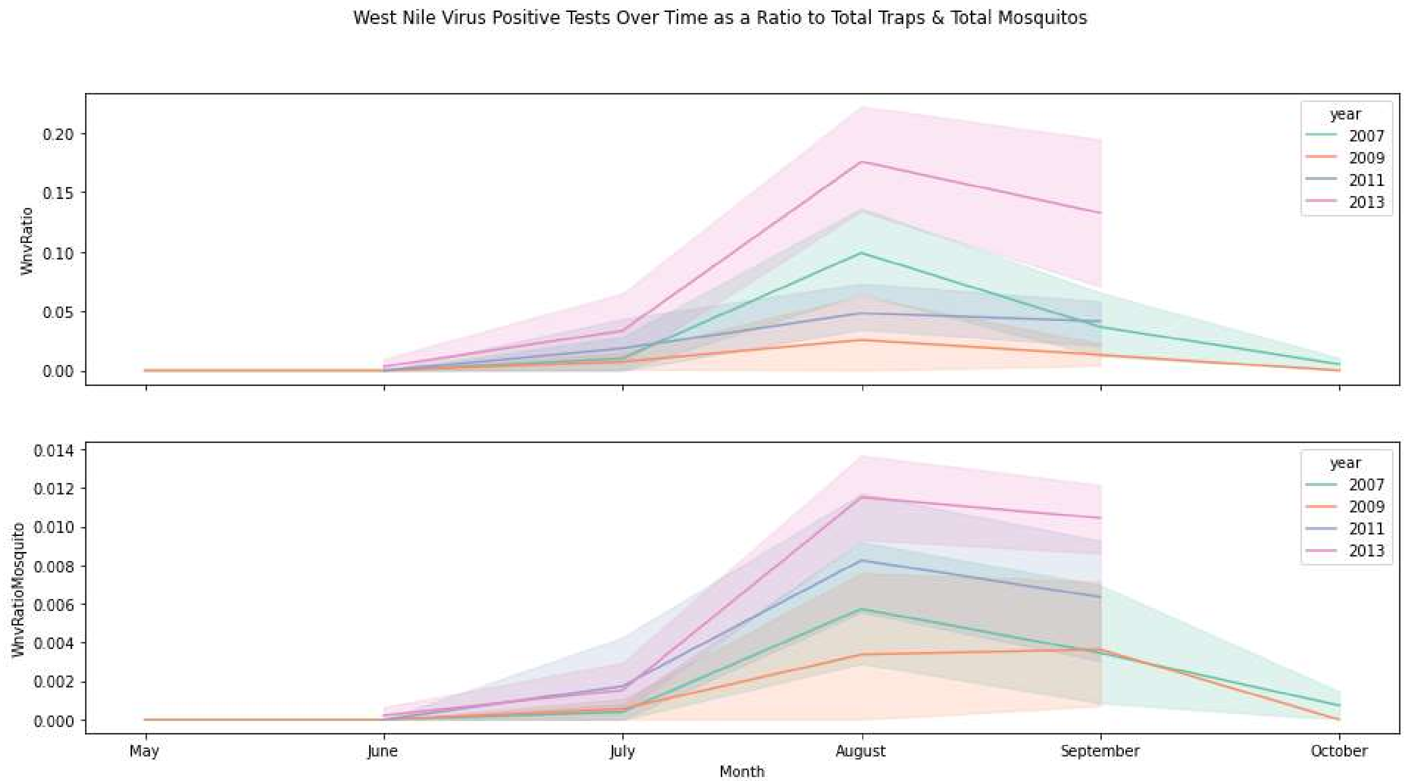
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1472 entries, 0 to 1471
Data columns (total 36 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          1472 non-null   object
 1   Tmax          1472 non-null   float64
 2   Tmin          1472 non-null   float64
 3   Tavg          1472 non-null   float64
 4   Depart        1472 non-null   float64
 5   DewPoint      1472 non-null   float64
 6   WetBulb       1469 non-null   float64
 7   Heat          1472 non-null   float64
 8   Cool          1472 non-null   float64
 9   Sunrise       1472 non-null   object
 10  Sunset        1472 non-null   object
 11  CodeSum       1472 non-null   object
 12  SnowFall      1472 non-null   float64
 13  PrecipTotal   1472 non-null   float64
 14  StnPressure   1470 non-null   float64
 15  SeaLevel      1467 non-null   float64
 16  ResultSpeed   1472 non-null   float64
 17  ResultDir     1472 non-null   int64
 18  AvgSpeed      1472 non-null   float64
 19  CodeSumSplit  1472 non-null   object
 20  BCFG          1472 non-null   int32
 21  BR            1472 non-null   int32
 22  DZ            1472 non-null   int32
 23  FG            1472 non-null   int32
 24  FG+           1472 non-null   int32
 25  FU            1472 non-null   int32
 26  GR            1472 non-null   int32
 27  HZ            1472 non-null   int32
 28  MIFG          1472 non-null   int32
 29  RA            1472 non-null   int32
 30  SN            1472 non-null   int32
 31  SQ            1472 non-null   int32
 32  TS            1472 non-null   int32
 33  TSRA          1472 non-null   int32
 34  VCFG          1472 non-null   int32
 35  VCTS          1472 non-null   int32
dtypes: float64(14), int32(16), int64(1), object(5)
memory usage: 322.1+ KB
```

Final Weather Dataset

Data Wrangling Part 2: Mosquito Traps Dataset
First, we merge our cleaned weather dataframe into our mosquito trap dataframe using the “Date” column as our ID. We then delete any of the weather condition columns which sum to 0 because this indicates that for the dates we’re working with in our mosquito trap dataset, there were no instances of those weather conditions occurring.

Next, we use get_dummies again to separate the “Species” column into identifying variables like we did with the weather conditions.

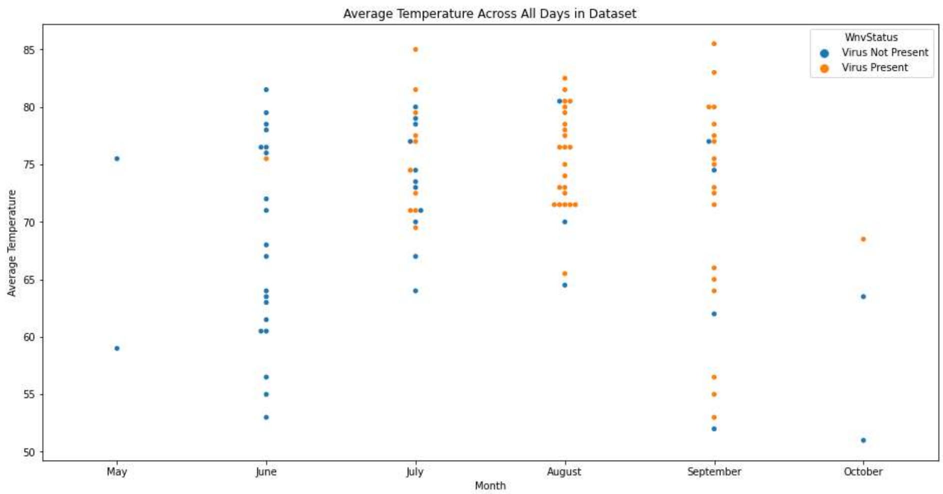
Exploratory Data Analysis:
We group our dataframe by date and then sum the values which allows to create a ratio to track the trend of the virus over time.



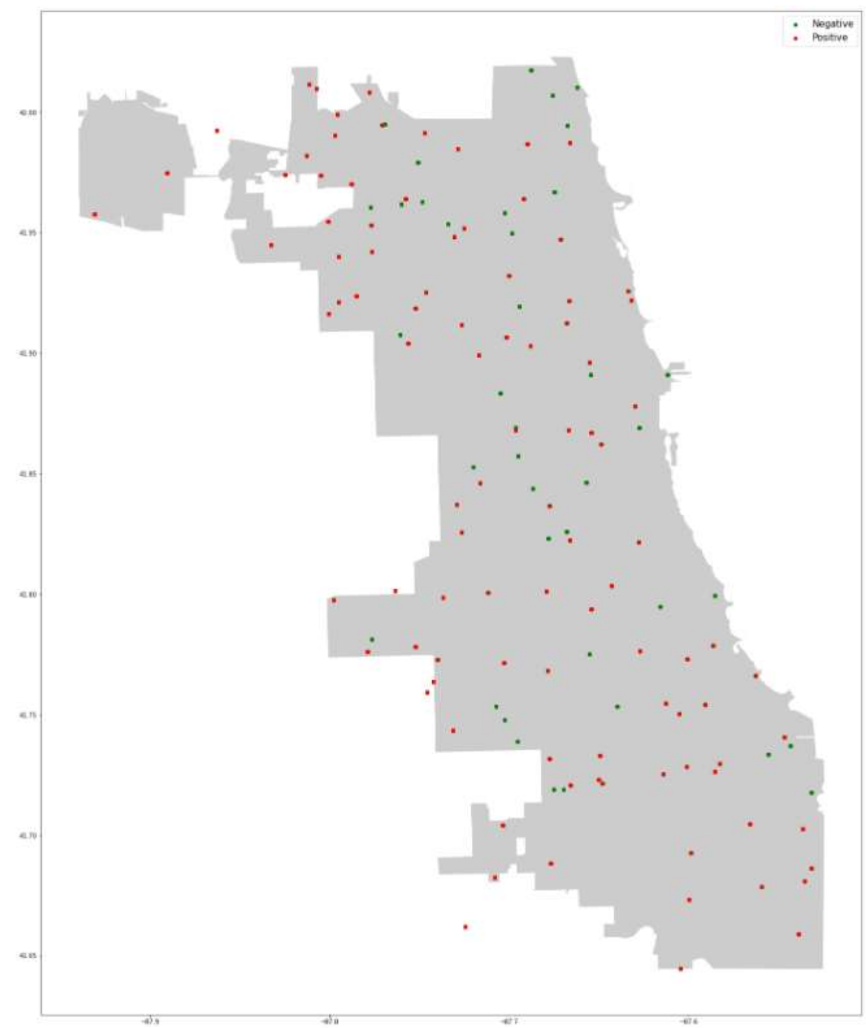
Here, we can immediately see that the virus starts to pick up in July before peaking in August and then tapering off in September and October.

Using our same aggregated dataset, this time we take the mean across each date in order to observe trends between daily temperatures and the virus.

```
In [50]: df_mean = df_final.groupby('Date').mean()
         df_mean['month'] = months
         df_mean['year'] = years
         df_mean['WnvStatus'] = virus
         df_mean['NumMosquitos'] = df_mean['NumMosquitos'].astype('float')
         swarmplot = sns.swarmplot(x='month',y='Tavg',hue='WnvStatus', data=df_mean)
         plt.title('Average Temperature Across All Days in Dataset')
         plt.ylabel('Average Temperature')
         plt.xlabel('Month')
         swarmplot.set_xticklabels(['May','June','July','August','September','October'])
         plt.show();
```



Lastly, we make use of the geopandas library to represent our data on a map of Chicago where we can see that Northwest Chicago appears to be a hotspot for the virus.



Each of these observations were all eventually corroborated by being the most impactful features in our final model.

Feature Engineering:
We add the following features to our dataframe:

1. Days of the Week
2. Month & Year
3. Days Since Previous Weather Condition: For each weather condition, we mark the number of days that pass between two consecutive instances of that weather condition occurring. Note that because there are two year gaps in our data (2007, 2009, 2011, 2013), we also have to take care to mark off the start of a new year with "N/A".
4. Municipalities: Using geocoders, we input our latitude and longitude values into Nominatim to pull up details on each of our trap locations. Municipalities was selected as the new categorical feature to be added because it had less null entries than other potential features while also having less variety in its actual values which helps prevent our data's dimensionality from growing exponentially large.

All of these added features are categorical variables for which we run through get_dummies to convert them all into indicators. After this step, we address all of the missing data by imputing any null values with the mean of their respective series.

```
In [14]: from sklearn.impute import SimpleImputer
         df_final = df2.copy(deep=True)
         mean_imputer = SimpleImputer(strategy='mean')
         df_final.loc[:,['WetBulb','StnPressure', 'TimeSinceLastBR',
                        'TimeSinceLastDZ','TimeSinceLastFG',
                        'TimeSinceLastHZ','TimeSinceLastRA','TimeSinceLastTS',
                        'TimeSinceLastTSRA','TimeSinceLastVCTS']] = mean_imputer.fit_transform(
             df2[['WetBulb','StnPressure', 'TimeSinceLastBR',
                        'TimeSinceLastDZ','TimeSinceLastFG',
                        'TimeSinceLastHZ','TimeSinceLastRA','TimeSinceLastTS',
                        'TimeSinceLastTSRA','TimeSinceLastVCTS']])
```

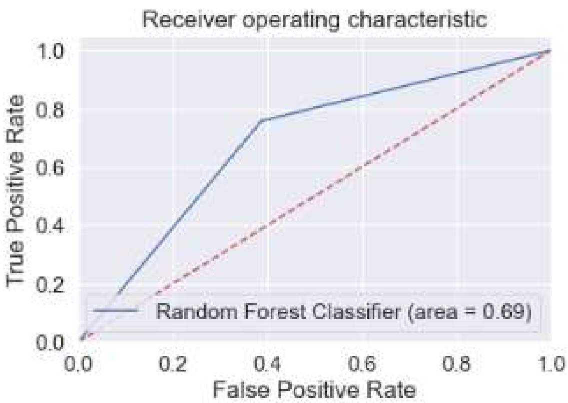
Model Preprocessing:
Now we can split our data into training and test sets. However, because there are an overwhelming number of "negative" WNV cases in our data, we need to sample an equal proportion of both positive and negative cases so that our model can fairly learn both (otherwise we end up with a model that will inevitably predict "negative" on almost everything). After taking this sample, we feed it into sklearn's train_test_split.

Our last step before entering the modeling stage is to reduce the dimensionality of our data by only selecting the features that will have the most impact in our model predictions. We do this by calculating the information value of each feature and then only selecting the ones which have IV > 0.01 (has some amount of predictive power) but also < 0.80 (is suspiciously high and would overrule the other features). We also calculate the variance inflation factor ("VIF") on each of these features and remove any which have a value >5 as this would be an indication that the feature in question has high collinearity with another feature; this process helps keep the features independent.

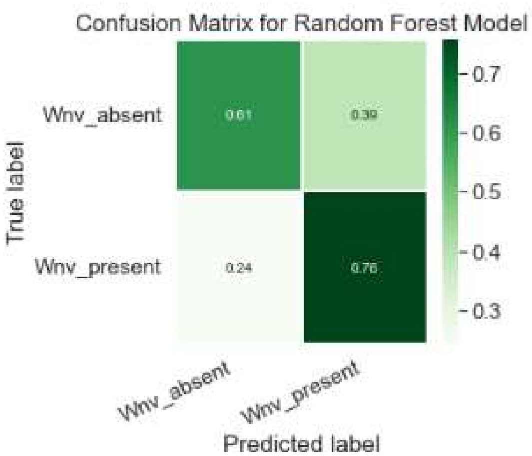
At the end of our preprocessing step, we are left with 18 features in our final dataset.

Modeling & Results:
We built a Random Forest Model and an XG Boost Model. The performance metrics for each are presented below:

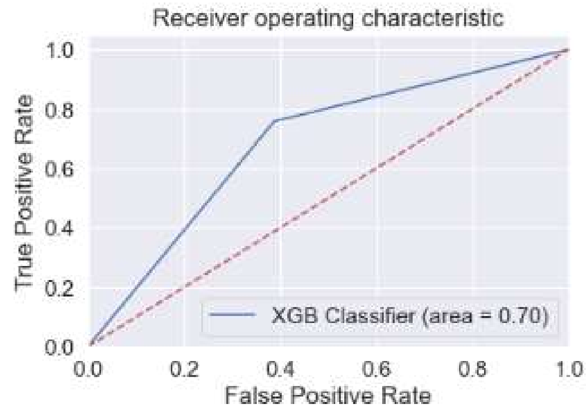
<u>Random Forest:</u>



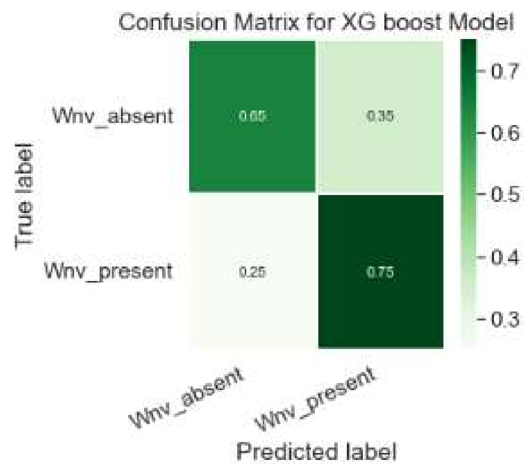

```
Random Forest: Accuracy=0.680
Random Forest: F1 Score=0.679
Random Forest: ROC Score=0.685
```



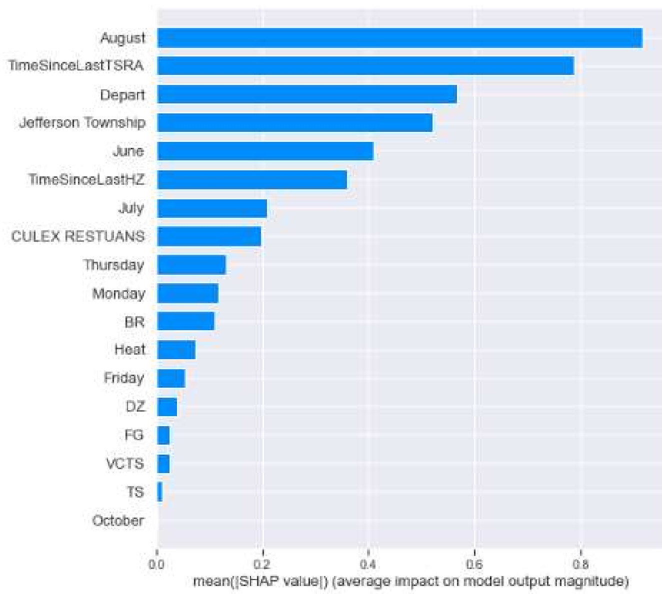
<u>XG Boost:</u>

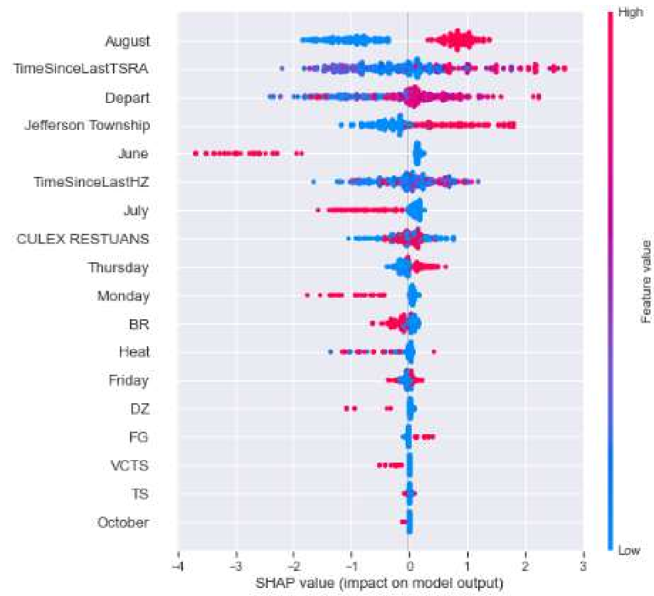



```
XG Boost: Accuracy=0.695
XG Boost: f1-score=0.695
XG Boost: ROC Score=0.699
```



Our XG Boost model does slightly better overall on accuracy, f1-score, and ROC score so it is selected as the final model for our project. However, note that Random Forest actually slightly outperformed XG Boost on recall. Future iterations of this project will likely explore new methodololgies that can improve the random forest model on the basis that true positives are the most important metric to optimize given the context of this dataset.





Our SHAP analysis reveals that the most impactful features in our model were as follows:

1. August: If the data was from August, this made the model more likely to predict positive.
2. TimeSinceLastTSRA: Generally, the longer it has been since there was last a rainy thunderstorm, the more likely the model predicts positive.
3. Departure from the 30-year normal temperature: The more extreme the temperature, the more likely the model predicts positive. This is potentially related to why August is the peak month since that's when the highest temperatures occur.
4. Jefferson Township: Traps from Jefferson Township (which encompasses most of Northwest Chicago) will make the model more likely to predict positive.

Note that these features corroborate with our earlier observations made during our exploratory data analysis.

Final Thoughts & Recommendations:

Although the CPDH has been monitoring the WNV situation since 2004, the prevalence of the virus has only continued to trend upwards year by year. Our model results produce the following recommendations on strategies to optimize the city's efforts in combating this virus for future years:

1. Increase coverage of spraying in the Northwest Chicago region particularly the areas encompassed in the Jefferson Township Municipality.
2. Focus the spraying schedule to be more concentrated in August.
3. Add in additional out-of-schedule sprays if temperatures are noted to be particularly high for a certain day OR if at least 3 weeks have gone by without any rain or thunderstorms.

Our final model had an overall ROC score of 0.699 and a recall score of 0.75. Future efforts to improve our model will focus on trying to improve the recall since the implications of positive cases are far greater than the consequences of misclassifying negative cases as false positives.