1 Summary

Business Problem: NYPD conducts too many stop-and-frisks that do not lead to an arrest. 92% of them in the last 10 years have not lead to arrest. There's a racial disparity problem with stop-and-frisk, making it especially important that NYPD lowers the stops it does that do not lead to an arrest.

In no way does this project advocate for more citywide arrests, but rather it advocates for NYPD to make more informed, better choices.

Business Solution: A time series model which can predict seasonality of stop-and-frisks that do not lead to an arrest. NYPD can act by limiting stops during times when stops are unlikey to lead to arrest.

Data Understanding: I'll be inspecting data going back 11 years to 2011, however stop-and-frisks were deemed unconstitutional in 2013, causing the rates and patterns of stops to change. Therefore I'll use data from 2014 onwards to model.

Data Limitations: There will automatically be discrepancies since the pandemic in 2020 affected many time-oriented patterns of behavior and policing.

I am using time data and not categorical data specifically because it's unethical to predict exactly what demographics of people will commit crimes in the future, especially with race. On the other hand, there is a provable season to crime spikes.

How this model is intended to be used: This model is designed to be used to predict six months out.

Data Preparation: I prepared the data in Data Prep notebook within this repo, to clean up the DateTime columns and setting that as the index.

Modeling: I am using naive modeling as a baseline, random walk, ARI, IMA and SARIMA model types. Facebook prophet would not be appropriate nor a neural network for this problem, as fb prophet is for more hourly and daily frequencies and neural networks aren't for univariate problems.

Evaluation: I used the model to forecast 6 months past the training set and compared that to the real last 6 months of data.

Conclusion overview: we can learn more from a naive model about how NYPD must become more responsive in practice, when not-arrested rates are spiking than we can from a more complex model. Complex modeling does lend itself to inference however to recognize when not-arrest rates are higher than historical precedence.

Sources: Stop and Frisk Data: https://www1.nyc.gov/site/nypd/stats/reports-analysis/stopfrisk.page (https://www1.nyc.gov/site/nypd/stats/reports-analysis/stopfrisk.page)

Precinct geojson file: https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page (https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page)

2 Imports

In [1]:

```
# importing padas, numpy, matplotlib, statsmodels, statsmodels tsa,
# datetime, and auto arima

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os, sys
import json
import glob

from sklearn.metrics import mean_squared_error
from math import sqrt

from statsmodels.tsa.arima.model import ARIMA

plt.rcParams['figure.figsize'] = (12, 4)
import geopandas as gpd
import statsmodels
import statsmodels.tsa.api as tsa

from datetime import time
import pmdarima as pm
from pmdarima.arima import ADFTest


import folium

print(f'statsmodels version : {statsmodels.__version__}')
print(f' pmdarima version : {pm.__version__}')
print(f' pandas version : {pd. __version__}')
print(f' geopandas version: {gpd.__version__}')
```

executed in 1.21s, finished 10:25:12 2022-09-29

```
statsmodels version : 0.13.2
 pmdarima version : 1.8.5
 pandas version : 1.1.3
 geopandas version: 0.11.1
```

3 Functions

In [2]:

```
def visualize_diagnostics(preds, train):

    """
    DOCSTRING:
    This function expects an input of a list of predictions and a train
    (or test/validation) set of original data. It produces a plots for visual
    diagnostics of model's predictions.

    We are looking to check both the range of values in the residuals,
    and to visually inspect the residuals for decerable patterns.


    We also get two statistics:
    1) the overall standard deviation of the residuals.
    2) the overall variance of the residuals.
    We want both of these statistics to be smaller with each model itteration.
    """

    fig, ax = plt.subplots()
    residuals = preds - train.squeeze()
    ax.plot(residuals.index, residuals)
    plt.title("Residuals")


    print(f'STANDARD DEVIATION OF RESIDUALS: {residuals.std()}')

```

executed in 3ms, finished 10:25:12 2022-09-29

In [3]:

```
def get_mae(y_true, predictions):
    """DOCSTRING:
    takes original data as an array, and the predicted values as an array
    and returns the mean of the absolute values of errors"""
    y_true, predictions = np.array(y_true), np.array(predictions)
    return np.mean(np.abs(y_true - predictions))

# taken from https://datagy.io/mae-python/
```

executed in 2ms, finished 10:25:12 2022-09-29

In [4]:

```
def last_six_visualize_preds(conf_int, train):

    """DOCSTRING:
    intakes a df of upper and lower confidence interval for a model.
    outputs a visualization showing the original data in a blue line and
    the 95% confidence interval of the algorithm as a green zone between
    two green lines. The upper green line corresponds to an
    upper limit, the lower green line corresponds to a lower limit."""

    fig, ax = plt.subplots()

    lower = sns.lineplot(data = conf_int,
                 y = 'lower NOT_ARRESTED_RATE',
                x = conf_int.index, color = 'g')

    upper = sns.lineplot(data = conf_int,
                 y = 'upper NOT_ARRESTED_RATE',
                x = conf_int.index, color = 'g')

    sns.lineplot(data = train, y = "NOT_ARRESTED_RATE",
                 x = train.index, color = 'blue', ax = ax)

    plt.fill_between(train.index,
                     conf_int['upper NOT_ARRESTED_RATE'],
                     conf_int['lower NOT_ARRESTED_RATE'],
                    color = 'g', alpha = 0.1)
    fig.tight_layout()
```

executed in 3ms, finished 10:25:12 2022-09-29

In [5]:

```
df = pd.read_csv('new_data/stop_and_frisk_w_race.csv', index_col= "DATE_TIME", )
df = df.drop(columns = 'Unique_ID')
df.head()

# loading in the data and making the DATE_TIME the index. This data has been
# prepared for either a time series model or a categorical model.
```

executed in 758ms, finished 10:25:13 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG	SUSPECT_RACE_DESCRIPTION	PRECINCT
2011-01-01 00:00	N	E. ASIAN	102
2011-01-01 00:10	N	BLACK	101
2011-01-01 00:10	N	BLACK	73
2011-01-01 00:10	N	BLACK	70
2011-01-01 00:15	N	WHITE HISPANIC	52

In [6]:
```
df.info()
# Looking at the data types of the columns tells us that we need to change
# precinct into a string since it's categorical
```

executed in 100ms, finished 10:25:13 2022-09-29

```
<class 'pandas.core.frame.DataFrame'>
Index: 1545827 entries, 2011-01-01 00:00 to 2021-12-31 23:18
Data columns (total 3 columns):
 #   Column                   Non-Null Count    Dtype 
---  ------                   --------------    ----- 
 0   SUSPECT_ARRESTED_FLAG    1545827 non-null  object
 1   SUSPECT_RACE_DESCRIPTION  1545827 non-null  object
 2   PRECINCT                 1545827 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 47.2+ MB
```

Getting a df ready just for Tableau EDA purposes:

1) dummying all the categorical variables so tableau can aggregate them

2) making a general purpose 'counts' column useful for aggregation

3) adding categorical columns back in so I can use both types of data

In [7]:
```
ten_years_tableau_data = pd.get_dummies(df, columns = ['SUSPECT_ARRESTED_FLAG',
                                                       'SUSPECT_RACE_DESCRIPTION'],)
ten_years_tableau_data["Counts"] = 1
ten_years_tableau_data["SUSPECT_RACE_CAT"] = df['SUSPECT_RACE_DESCRIPTION']
ten_years_tableau_data['SUSPECT_ARRESTED'] = df['SUSPECT_ARRESTED_FLAG']
# ten_years_tableau_data.to_csv("ten_years_tableau_data.csv")
```

executed in 402ms, finished 10:25:14 2022-09-29

4 EDA: inspecting totals

PLEASE NOTE: THIS EDA IS NOT MY FULL EDA PROCESS. THE FOLLOWING TABLEAU

DASHBOARDS ARE THE MAJORITY OF MY VISUALIZATION:

2011 - 2021 https://public.tableau.com/views/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link (https://public.tableau.com/views/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link)

JUST 2021 (the target)
https://public.tableau.com/app/profile/louis.casanave/viz/StopandFrisk2021NYPD/StopandFrisk2021Demograp
(https://public.tableau.com/app/profile/louis.casanave/viz/StopandFrisk2021NYPD/StopandFrisk2021Demograp

4.1 Arrested vs. Not Arrested Totals

In [8]:

```
df_dummied = pd.get_dummies(df, columns = ['SUSPECT_ARRESTED_FLAG',
                                           'SUSPECT_RACE_DESCRIPTION',
                                          'PRECINCT'])
df_dummied.head()

# I'm dummying all the columns because they are all categorical and we need
# to dummy them to aggrigate the nessisary statistics and reorganize the data
```

executed in 564ms, finished 10:25:14 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	SUSPECT_RACE_DESCRIPTION_BLACK	SUSPECT_RACE_DESCR
2011-01-01 00:00	1	0	0	0
2011-01-01 00:10	1	0	1	0
2011-01-01 00:10	1	0	1	0
2011-01-01 00:10	1	0	1	0
2011-01-01 00:15	1	0	0	0

5 rows × 89 columns

In [9]:

```
arrested_cols = ['SUSPECT_ARRESTED_FLAG_N', 'SUSPECT_ARRESTED_FLAG_Y']
# making a list of just arrest information

race_cols = [col for col in df_dummied.columns if 'RACE' in col]
# making a list of just the race information

precinct_cols = [col for col in df_dummied.columns if 'PRECINCT' in col]
# making a list of just the precinct information

arrested_df = df_dummied.drop(columns=[col for col in df_dummied.columns \
                                       if col not in arrested_cols])
# making arrested cols into it's own DF

arrested_df.head()

# taking a look at just the arrested information
```

executed in 76ms, finished 10:25:14 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y
2011-01-01 00:00	1	0
2011-01-01 00:10	1	0
2011-01-01 00:10	1	0
2011-01-01 00:10	1	0
2011-01-01 00:15	1	0

In [10]:

```
arrested_stats = pd.DataFrame(arrested_df.sum())
# summing up all the columns for raw stats
arrested_stats.rename({0 : "Counts"}, axis = 1, inplace = True)
# Renaming the column that holds the counts of information
arrested_stats
# Taking a look at these statistics
```

executed in 6ms, finished 10:25:14 2022-09-29

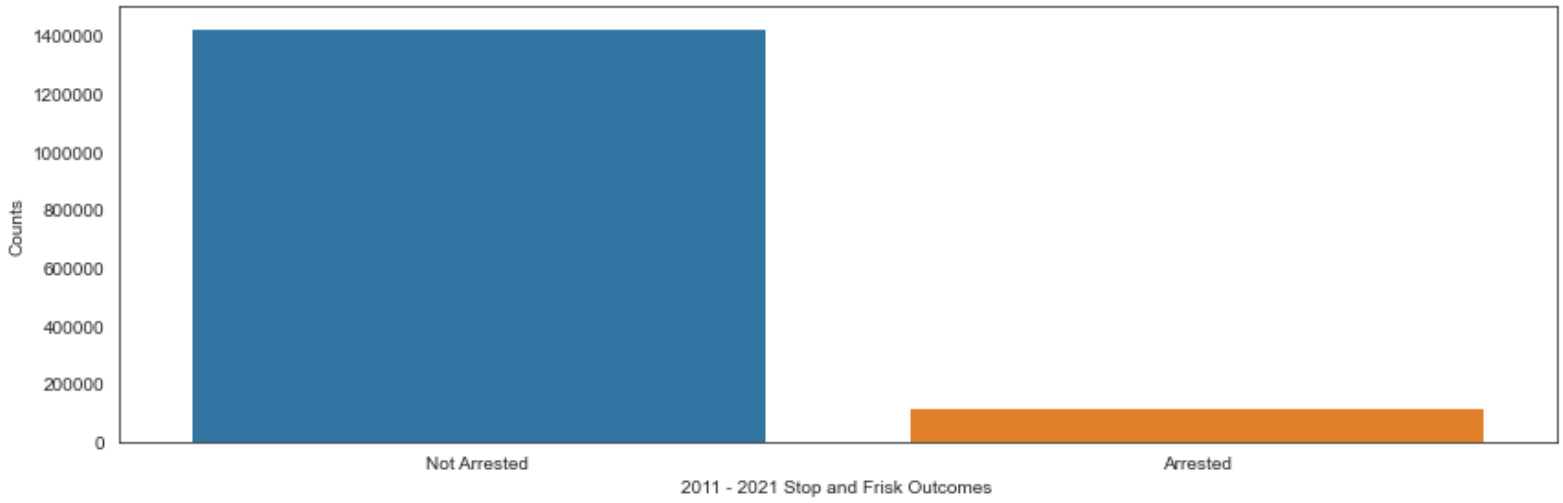
	Counts
SUSPECT_ARRESTED_FLAG_N	1425853
SUSPECT_ARRESTED_FLAG_Y	119974

4.1.1 Visualizing the total arrested vs not arrested

In [11]:

```
sns.set_style('white')
# just making sure the graphs can be seen on any background

arrested_plot = sns.barplot(x = arrested_stats.index,
                            y = 'Counts',
                            data = arrested_stats,)
# making a barplot to show the arrested stats

arrested_plot.set_xticklabels(['Not Arrested', 'Arrested'])
arrested_plot.set_xlabel('2011 - 2021 Stop and Frisk Outcomes')
plt.ticklabel_format(axis = 'y', style = 'plain', );
# giving the graphs titles and labels

plt.tight_layout()
plt.savefig("2011-2021 Stop and Frisk Outcomes")
```

executed in 104ms, finished 10:25:14 2022-09-29



4.2 Arrests by Race Totals

In [12]:

```
arrs_by_race_df = df_dummied.drop(columns=[col for col in df_dummied.columns \
                                           if col  in precinct_cols])
# making race information it's own df and taking a look at that df
arrs_by_race_df.head()
```

executed in 14ms, finished 10:25:14 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	SUSPECT_RACE_DESCRIPTION_BLACK	SUSPECT_RACE_DESCR
2011-01-01 00:00	1	0	0	0
2011-01-01 00:10	1	0	1	0
2011-01-01 00:10	1	0	1	0
2011-01-01 00:10	1	0	1	0
2011-01-01 00:15	1	0	0	0

In [13]:

```
arrs_by_race_df.rename({
          'SUSPECT_RACE_DESCRIPTION_BLACK' : 'BLACK',
          'SUSPECT_RACE_DESCRIPTION_BLACK HISPANIC' : 'BLACK_HISPANIC',
          'SUSPECT_RACE_DESCRIPTION_E. ASIAN' : 'E._ASIAN',
          'SUSPECT_RACE_DESCRIPTION_NATIVE' : 'NATIVE',
          'SUSPECT_RACE_DESCRIPTION_OTHER': 'OTHER',
          'SUSPECT_RACE_DESCRIPTION_UNKNOWN' : 'UNKNOWN',
          'SUSPECT_RACE_DESCRIPTION_W. ASIAN' : 'W._ASIAN',
          'SUSPECT_RACE_DESCRIPTION_WHITE' : 'WHITE',
          'SUSPECT_RACE_DESCRIPTION_WHITE HISPANIC' : 'WHITE_HISPANIC'},
         axis = 1, inplace = True)

# making our information more readable by renaming race columns

arrs_by_race_df.head()
```

executed in 7ms, finished 10:25:14 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	BLACK	BLACK_HISPANIC	E._ASIAN	NATIVE	OTHER	UNK
2011-01-01 00:00	1	0	0	0	1	0	0	0
2011-01-01 00:10	1	0	1	0	0	0	0	0
2011-01-01 00:10	1	0	1	0	0	0	0	0
2011-01-01 00:10	1	0	1	0	0	0	0	0
2011-01-01 00:15	1	0	0	0	0	0	0	0

In [14]:

```
outcomes = ['SUSPECT_ARRESTED_FLAG_N', 'SUSPECT_ARRESTED_FLAG_Y']
# making a list of outcomes to use

arrs_by_race = arrs_by_race_df.pivot_table(index = outcomes, aggfunc = 'sum')
# making a pivot table to show the totals of each outcome by race

arrs_by_race = arrs_by_race.reset_index(drop= True).transpose()
# flipping that table on it's side for easier readability

arrs_by_race.rename({0 : 'ARRESTED', 1 : 'NOT ARRESTED'}, axis = 1,
                    inplace = True)
# renaming outcomes for better readability

arrs_by_race.sort_values(by  = 'NOT ARRESTED', inplace = True)
# sorting races by the target, the not arrested statistic

arrs_by_race = arrs_by_race.astype(int)
# casing everything as an integer instead of a float for readability

arrs_by_race.style.bar()
# taking a look at the distributions and statistics
```

executed in 121ms, finished 10:25:15 2022-09-29

	ARRESTED	NOT ARRESTED
W._ASIAN	34	98
NATIVE	373	5964
UNKNOWN	871	10945
OTHER	1623	29367
E._ASIAN	4038	49057
BLACK_HISPANIC	9138	97089

	ARRESTED	NOT ARRESTED
WHITE	12354	134742
WHITE_HISPANIC	30078	346280
BLACK	61465	752311

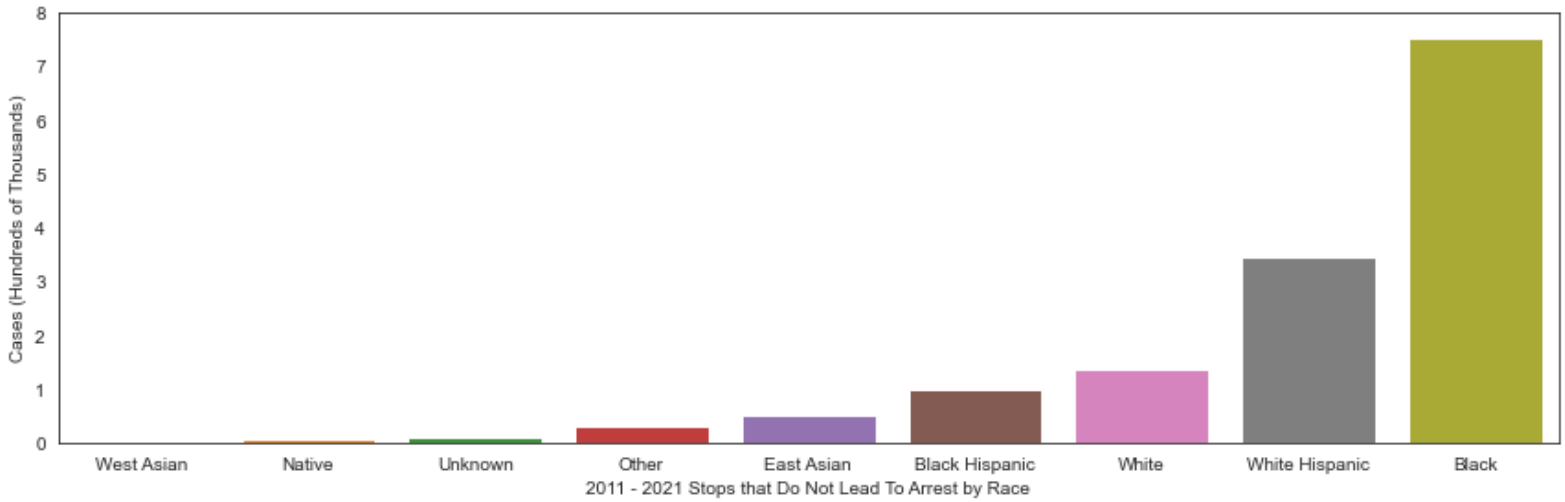
In [15]:
```
1 ### Visualizing
```

executed in 2ms, finished 10:25:15 2022-09-29

In [16]:

```
arrs_by_race_plot = sns.barplot(x = arrs_by_race.index,
                                y = 'NOT ARRESTED',
                                data = arrs_by_race)
# making a bar plot to look at how many stops did not lead to an arrest

arrs_by_race_plot.set_xlabel('2011 - 2021 Stops that Do Not Lead To Arrest by Race')
plt.xticks(ticks = [0, 1, 2, 3, 4, 5, 6, 7, 8],
          labels = ['West Asian', 'Native', 'Unknown', 'Other', 'East Asian',
                    'Black Hispanic', 'White', 'White Hispanic', 'Black'])
plt.ylabel("Cases (Hundreds of Thousands)")
plt.yticks(ticks = [0, 100000,200000,300000,400000,500000,600000,700000, 800000],
          labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8']);
# setting the label on the graph

plt.tight_layout()
plt.savefig("Not Arrested By Race")
```

executed in 147ms, finished 10:25:15 2022-09-29



4.3 Arrest by Precinct Totals

In [17]:
```
arrs_by_precinct_df = df_dummied.drop(columns=[col for col in \
                                               df_dummied.columns if col \
                                               in race_cols])
# making arrests by precinct it's own DF

arrs_by_precinct_df.head()
# taking a look at the data
```

executed in 107ms, finished 10:25:15 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	PRECINCT_0	PRECINCT_1	PRECINCT_5	PRECINCT_6	PRE
2011-01-01 00:00	1	0	0	0	0	0	0
2011-01-01 00:10	1	0	0	0	0	0	0
2011-01-01 00:10	1	0	0	0	0	0	0
2011-01-01 00:10	1	0	0	0	0	0	0
2011-01-01 00:15	1	0	0	0	0	0	0

5 rows × 80 columns

In [18]:

```
arrs_by_precinct = arrs_by_precinct_df.pivot_table(index = outcomes, aggfunc = 'sum')
# making a pivot table to look at the outcomes by race with all cases totaled

arrs_by_precinct = arrs_by_precinct.reset_index(drop= True).transpose()
# resetting the index and flipping the pivot table over for readability

arrs_by_precinct.rename({0 : 'ARRESTED', 1 : 'NOT ARRESTED'}, axis = 1,
                    inplace = True)
# renaming the new columns for readability

arrs_by_precinct = arrs_by_precinct.astype(int)
# casing everything as an integer instead of a float for readability

arrs_by_precinct.head()
# taking a look at the pivot table (note that I am keeping precint_0 as it was
# used in data preperation to denote an unknown precinct, rather than dropping
# values)
```

executed in 1.10s, finished 10:25:16 2022-09-29

	ARRESTED	NOT ARRESTED
PRECINCT_0	113	319
PRECINCT_1	704	7857
PRECINCT_10	1010	6442
PRECINCT_100	912	10891
PRECINCT_101	1486	28719

In [19]:

```
arrs_by_precinct['DIFFERENCE'] = arrs_by_precinct['NOT ARRESTED'] - arrs_by_precinct['ARRESTED']
# making a column to keep track of the differnce between stops that lead to an
# arrest and stops that do not lead to an arrest

arrs_by_precinct.sort_values(by = 'DIFFERENCE', inplace = True)
# sorting the data by that difference

arrs_by_precinct.astype(int).style.bar()
# recasting the data as an int for readability and looking at the distributions
# and statistics
```

executed in 17ms, finished 10:25:16 2022-09-29

	ARRESTED	NOT ARRESTED	DIFFERENCE
PRECINCT_0	113	319	206
PRECINCT_22	145	2294	2149
PRECINCT_121	380	3692	3312
PRECINCT_17	437	4002	3565
PRECINCT_94	555	4795	4240
PRECINCT_123	498	5338	4840
PRECINCT_50	743	5666	4923
PRECINCT_10	1010	6442	5432
PRECINCT_5	1074	6937	5863
PRECINCT_68	837	6770	5933
PRECINCT_18	953	7506	6553
PRECINCT_6	792	7731	6939

In [20]:

```
arrs_by_precinct.reset_index(inplace = True)

# setting the index for visualization
```

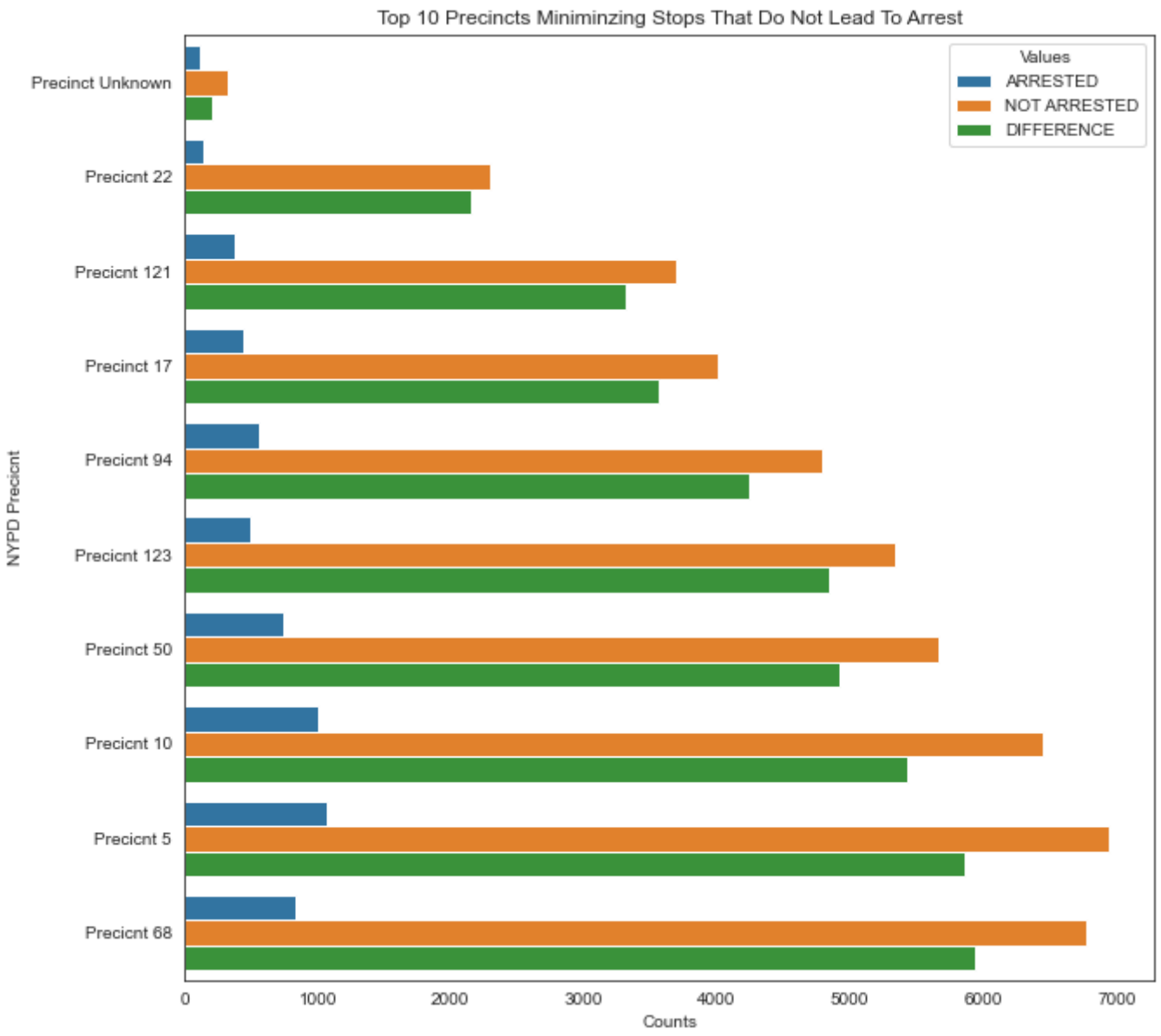
executed in 2ms, finished 10:25:16 2022-09-29

4.4 visualizing outcome disparities by precinct

In [21]:

```
plt.figure(figsize=(10, 10))
# resetting the size of the plot

sns.barplot(data = arrs_by_precinct.head(10).melt(id_vars='index',
                                        value_name='Counts',
                                        var_name='Values'),
            y='index', x='Counts', hue='Values')
# making a plot to show the precicnts where the difference is lowest aka the
# police are doing the best job determening who to stop

plt.title("Top 10 Precincts Miniminzing Stops That Do Not Lead To Arrest")
plt.yticks(ticks = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
           labels=['Precinct Unknown', 'Precicnt 22', 'Precicnt 121',
                    'Precinct 17', 'Precicnt 94', 'Precicnt 123', 'Precinct 50',
                   'Precicnt 10', 'Precicnt 5', 'Precicnt 68'])
plt.ylabel("NYPD Precicnt");
# setting labels and titles
```

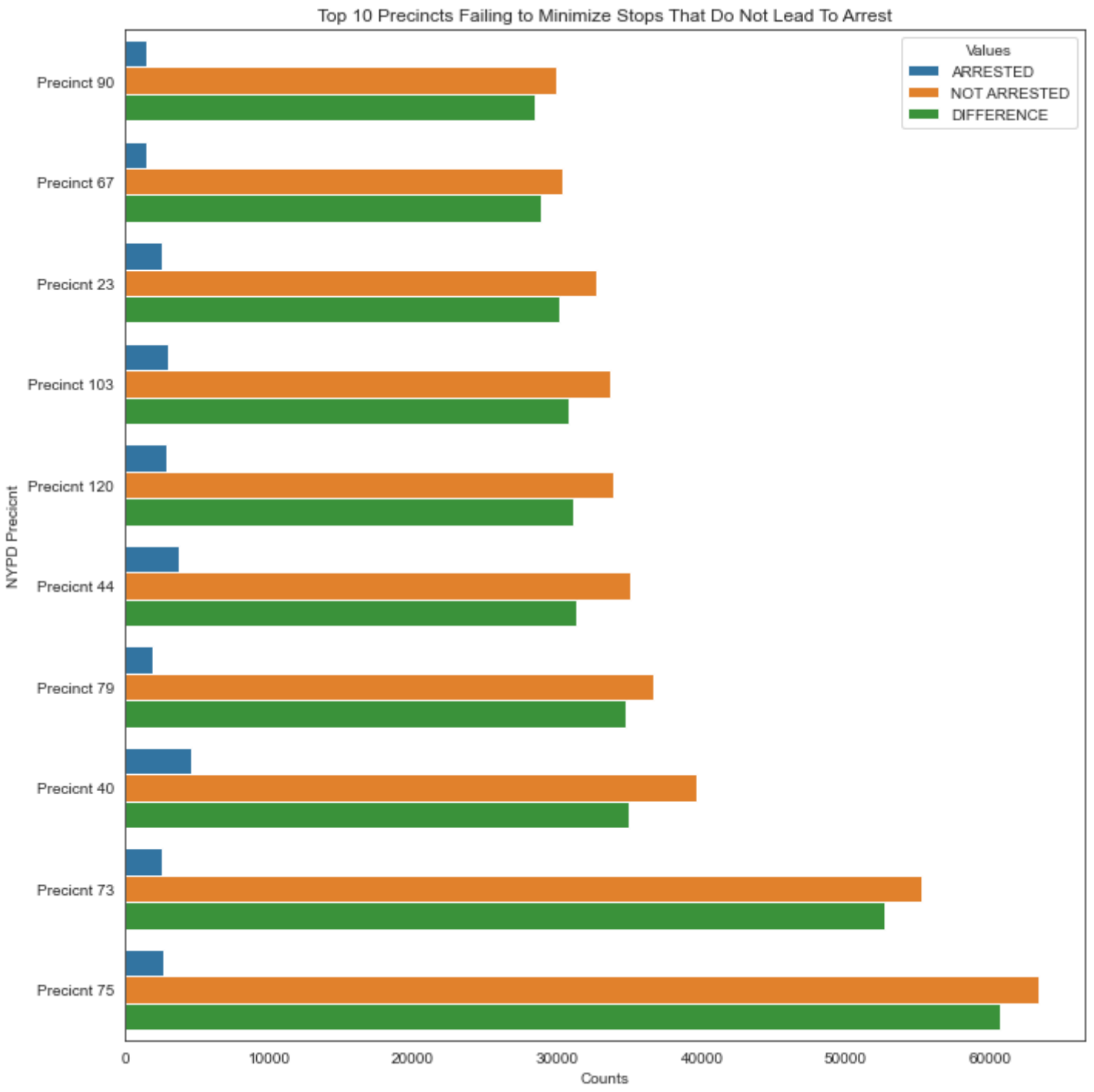
executed in 191ms, finished 10:25:16 2022-09-29


Top 10 Precincts Miniminzing Stops That Do Not Lead To Arrest
Values
ARRESTED
NOT ARRESTED
DIFFERENCE
Precinct Unknown
Precicnt 22
Precicnt 121
Precinct 17
Precicnt 94
Precicnt 123
Precinct 50
Precicnt 10
Precicnt 5
Precicnt 68
NYPD Precicnt
0
1000
2000
3000
4000
5000
6000
7000
Counts

In [22]:

```
plt.figure(figsize=(10, 10))
# resetting the size of the plot

sns.barplot(data = arrs_by_precinct.tail(10).melt(id_vars='index',
                                        value_name='Counts',
                                        var_name='Values'),
            y='index', x='Counts', hue='Values')
# making a plot to show the precicnts where the difference is lowest aka the
# police are doing the best job determening who to stop

plt.title("Top 10 Precincts Failing to Minimize Stops That Do Not Lead To Arrest")
plt.yticks(ticks = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
           labels=['Precinct 90', 'Precinct 67', 'Precicnt 23',
                    'Precinct 103', 'Precicnt 120', 'Precicnt 44', 'Precinct 79',
                  'Precicnt 40', 'Precicnt 73', 'Precicnt 75'])
plt.ylabel("NYPD Precicnt");
# setting labels and titles

plt.tight_layout()
plt.savefig("Top_10_Precicnts_Failing_To_Minimize")
```

executed in 253ms, finished 10:25:16 2022-09-29


Top 10 Precincts Failing to Minimize Stops That Do Not Lead To Arrest
Values
ARRESTED
NOT ARRESTED
DIFFERENCE
NYPD Precicnt
Precinct 90
Precinct 67
Precicnt 23
Precinct 103
Precicnt 120
Precicnt 44
Precinct 79
Precicnt 40
Precicnt 73
Precicnt 75
0
10000
20000
30000
40000
50000
60000
Counts

Observations: In Both Plots these bars look very similar, with the exception of when the precinct was unknown it's just a matter of scale.

4.5 Arrests by Precinct On a Map

In [23]:

```
arrs_by_precinct_map = arrs_by_precinct.copy()
# making a copy of arrs_by_precicnt to use for mapping

vals = [x.split('_')[1] for x in arrs_by_precinct_map['index']]
# seperating "precicnt" out of the precicnts index

arrs_by_precinct_map['index'] = vals
# replacing the index with the new list of values

arrs_by_precinct_map.drop(0, inplace = True)
# dropping precinct 0 or unknown as it will mess with the mapping

arrs_by_precinct_map
# taking a look at the data
```

executed in 7ms, finished 10:25:16 2022-09-29

	index	ARRESTED	NOT ARRESTED	DIFFERENCE
1	22	145	2294	2149
2	121	380	3692	3312
3	17	437	4002	3565
4	94	555	4795	4240
5	123	498	5338	4840
...	...	...	...	...
73	44	3752	35000	31248
74	79	1942	36667	34725
75	40	4626	39603	34977
76	73	2545	55248	52703
77	75	2668	63346	60678

77 rows × 4 columns

In [24]:

```
arrs_by_precinct_map['index'] = arrs_by_precinct_map['index'].astype(int)
# casting the index to be integers so the geojsn file will recognize it

arrs_by_precinct_map['DIFFERENCE'] = arrs_by_precinct_map['DIFFERENCE'].astype(int)
# doing the same thing with the differnce just to be sure

arrs_by_precinct_map.info()
# triple checking data types before we visualize
```

executed in 5ms, finished 10:25:16 2022-09-29

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 77 entries, 1 to 77
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   index         77 non-null     int64
 1   ARRESTED      77 non-null     int64
 2   NOT ARRESTED  77 non-null     int64
 3   DIFFERENCE    77 non-null     int64
dtypes: int64(4)
memory usage: 3.0 KB
```

4.5.1 Making a choropleth that uses the difference as the shading

to make a visualization using folium, I'll need a geojason file for the precincts, found here

citation: https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page (https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page)

In [25]:

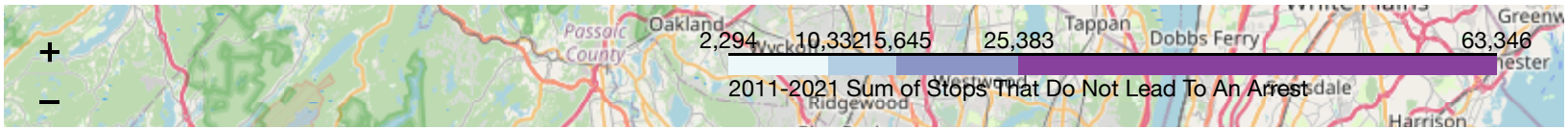
```
with open ('tableau_data/precincts.json') as f:
    json_data = json.load(f)
    # open the geojsn file and nameing it json_data
```

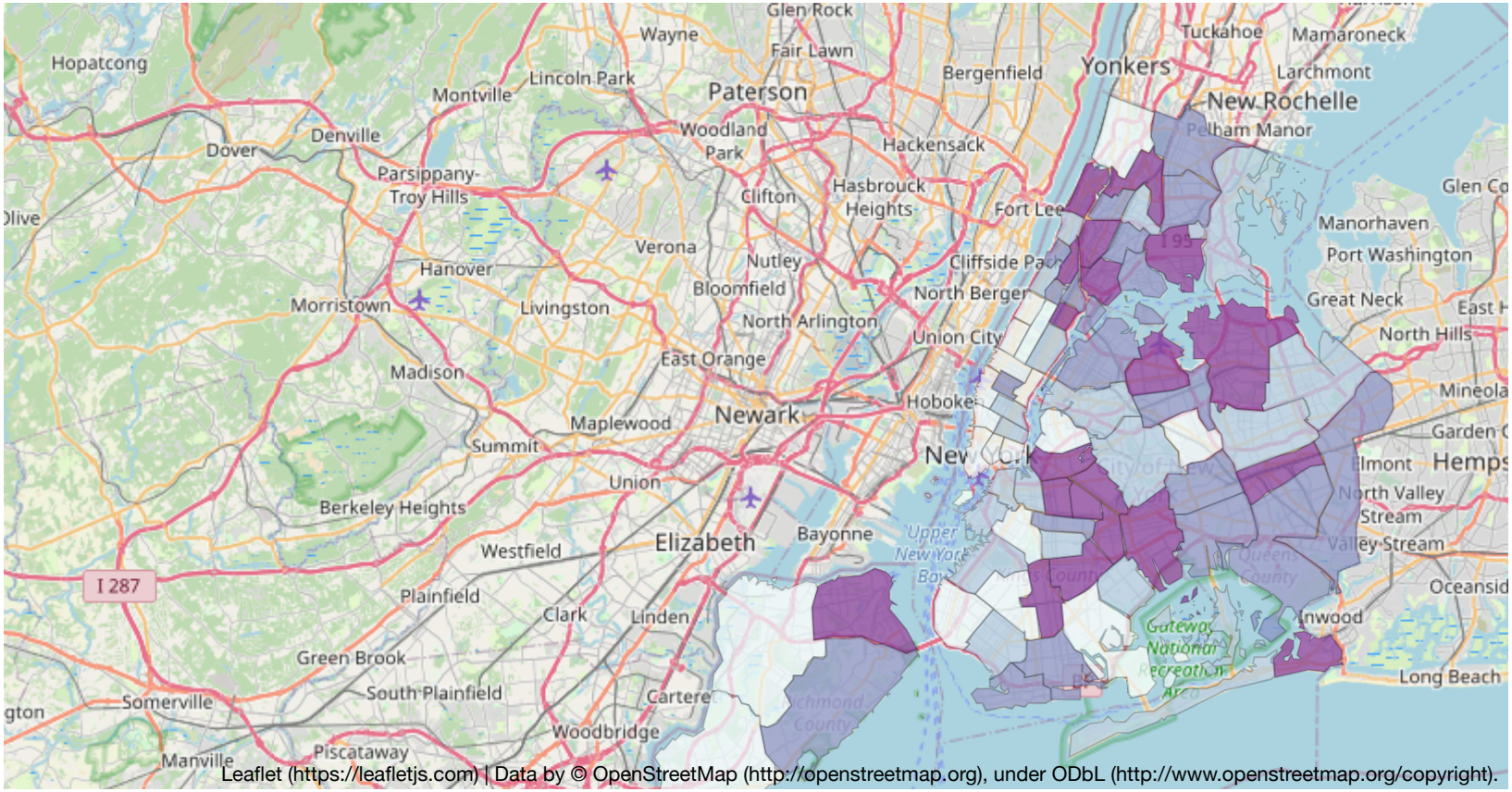
executed in 136ms, finished 10:25:17 2022-09-29

In [26]:

```
my_thresh = arrs_by_precinct_map['NOT ARRESTED'].quantile((0, 0.25, 0.5,
                                                           0.75, 1)).tolist()
# making the threshold of shades to represent the quartiles where most of the
# data is

m = folium.Map(location=[40.693943, -73.985880], default_zoom_start=10)
# making the base map of NYC

folium.Choropleth(
    geo_data = json_data,
    name = "Sum of Stops That Did Not Lead To Arrest",
    data = arrs_by_precinct_map,
    columns = ["index", 'NOT ARRESTED'],
    key_on = "feature.properties.Precinct",
    fill_color = 'BuPu',
    fill_opacity = 0.75,
    line_opacity = 0.2,
    legend_name = "2011-2021 Sum of Stops That Do Not Lead To An Arrest",
    threshold_scale = my_thresh
).add_to(m)
# adding the choropleth to the base map

folium.LayerControl().add_to(m)
# right now this map has only one layer, but in future work it may have more

m
# showing the final map
```

executed in 490ms, finished 10:25:17 2022-09-29





Observations: (I will be brief as this map is static and I've improved it using Tableau public.)

Over the last ten years, stops that do not lead to arrest have been most populous in East New York and Bushwick, Flatbush, East Harlem, and the South Bronx. Stops that do lead to arrest have been most common on the west side of Manhattan and Downtown Manhattan, Bay Ridge, and the parts of Staten Island furthest away from the city.

Most of the data skews very heavily in the upper extreme, making places where stops do lead to an arrest more common the real outliers of this data.

5 EDA: inspecting timelines

In [27]:
```
df_dummied.index = pd.to_datetime(df_dummied.index)
df_dummied = df_dummied.sort_index()
# making sure the index was set to a datetime object instead of a string
```

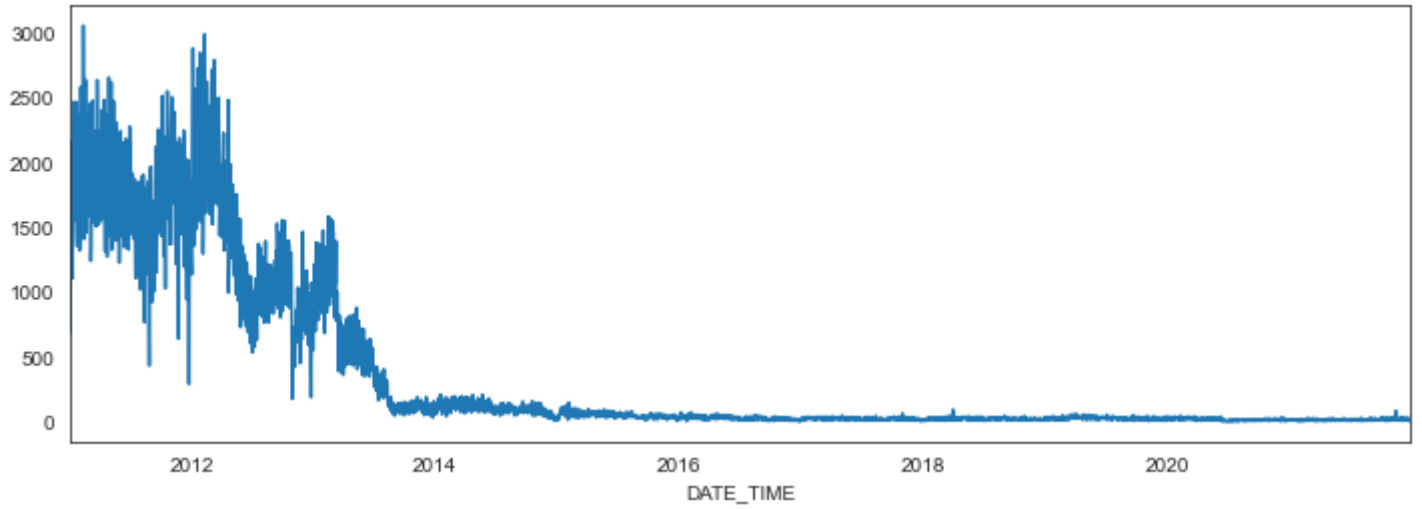
executed in 492ms, finished 10:25:18 2022-09-29

5.1 including before it was deemed unconstitutional

In [28]:
```
by_day = df_dummied['SUSPECT_ARRESTED_FLAG_N'].resample('D').sum()
by_day.plot()
# resampling is going to make an entry for each day and aggrigate (sum) all
# records for that day. then I'll show the plot. This data is very fuzzy so
# I'll be looking at weekly data.
```

executed in 159ms, finished 10:25:18 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```

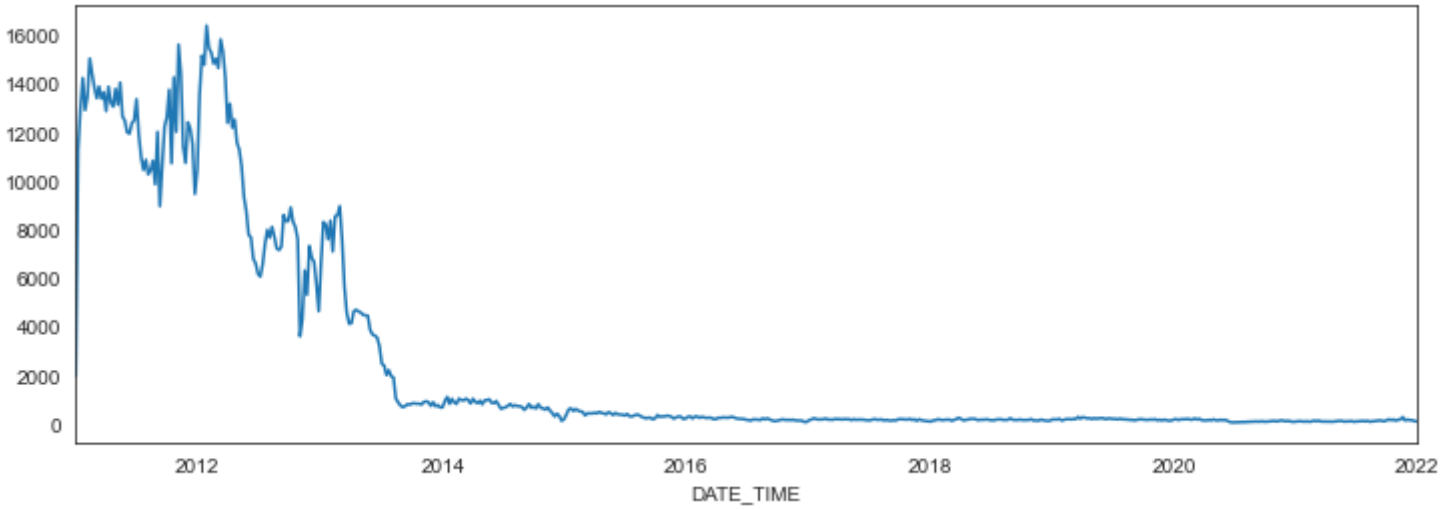


In [29]:

```
by_week = df_dummied['SUSPECT_ARRESTED_FLAG_N'].resample('W').sum()
by_week.plot()

# doing the same thing but for weekly intervals
```

executed in 124ms, finished 10:25:18 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```

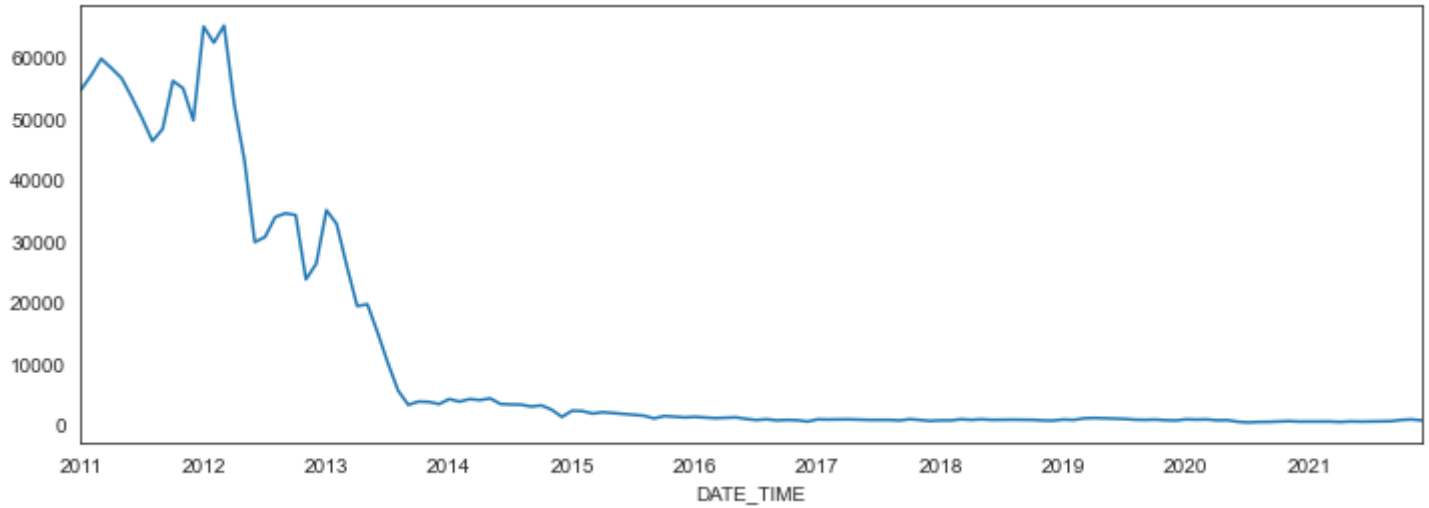


Observations: there was a huge drop after Floyd vs City of New York in Aug 2013 which we can see on this plot and the next one.

In [30]:
```
by_month = df_dummied['SUSPECT_ARRESTED_FLAG_N'].resample('M').sum()
by_month.plot()
```

executed in 216ms, finished 10:25:18 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```



Due to Floyd vs the city of New York, I will not be using all of the data I have, but I will start using data only after 2014. A model can't account for something like a change in law, so I'll only be modeling after the new policy was in place.

5.2 only after it was deemed unconstitutonal

In [31]:

```
fourteen_onward = df_dummied.loc['2014':]
fourteen_onward.head()

# making a seperate df starting in 2014 and moving on from there, taking a look
```

executed in 33ms, finished 10:25:18 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	SUSPECT_RACE_DESCRIPTION_BLACK	SUSPECT_RACE_DESCR
2014-01-01 00:10:00	1	0	1	0
2014-01-01 00:20:00	0	1	0	1
2014-01-01 00:30:00	1	0	1	0
2014-01-01 00:30:00	1	0	1	0
2014-01-01 00:30:00	1	0	1	0

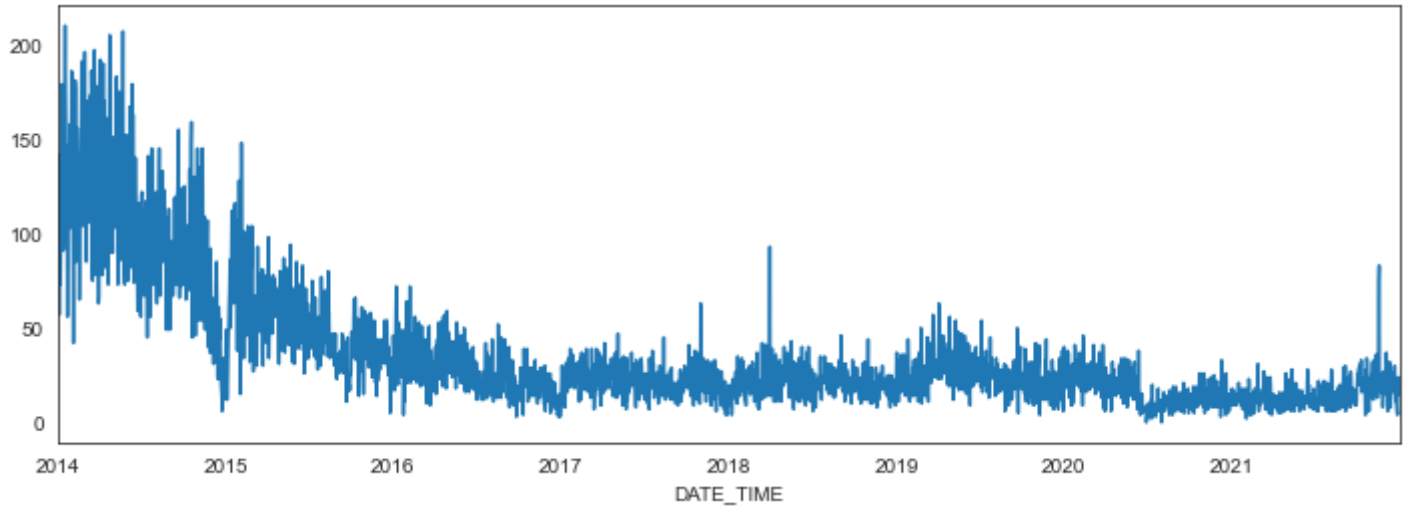
5 rows × 89 columns

In [32]:

```
fourteen_by_day = fourteen_onward['SUSPECT_ARRESTED_FLAG_N'].resample('D').sum()
fourteen_by_day.plot()
# resampling the new data by day to get a general impression of the data.
# this line is too tempremental to try and parse still
```

executed in 168ms, finished 10:25:18 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```

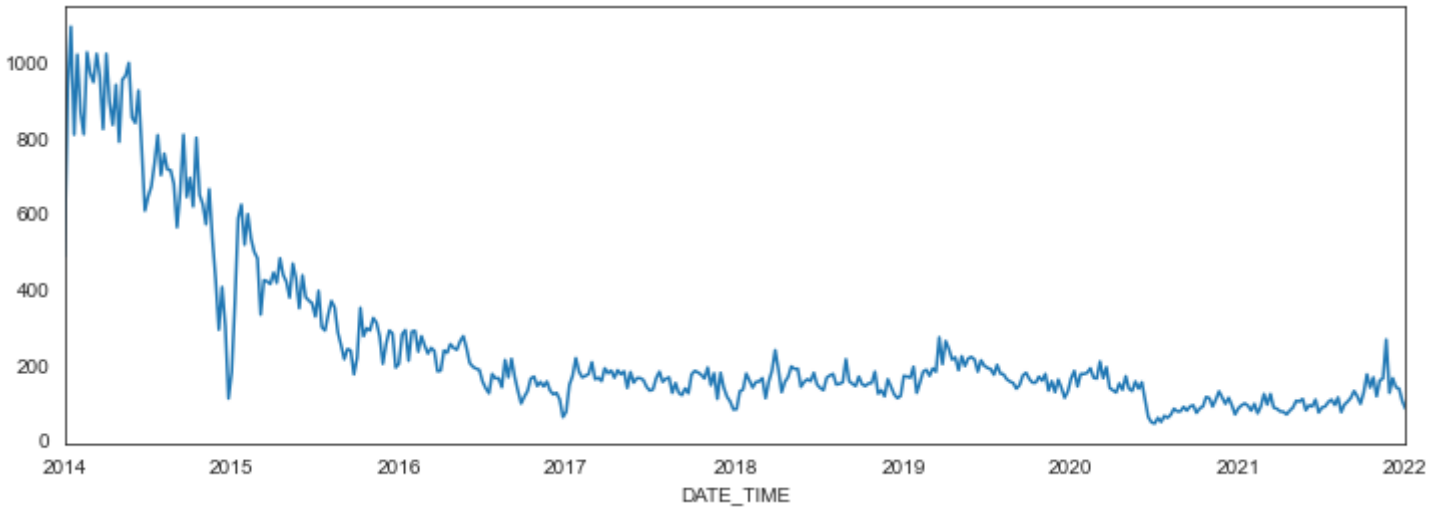


In [33]:

```
fourteen_by_week = fourteen_onward['SUSPECT_ARRESTED_FLAG_N'].resample('W').sum()
fourteen_by_week.plot()
# resampling by week does offer much more clarity.
```

executed in 160ms, finished 10:25:18 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```

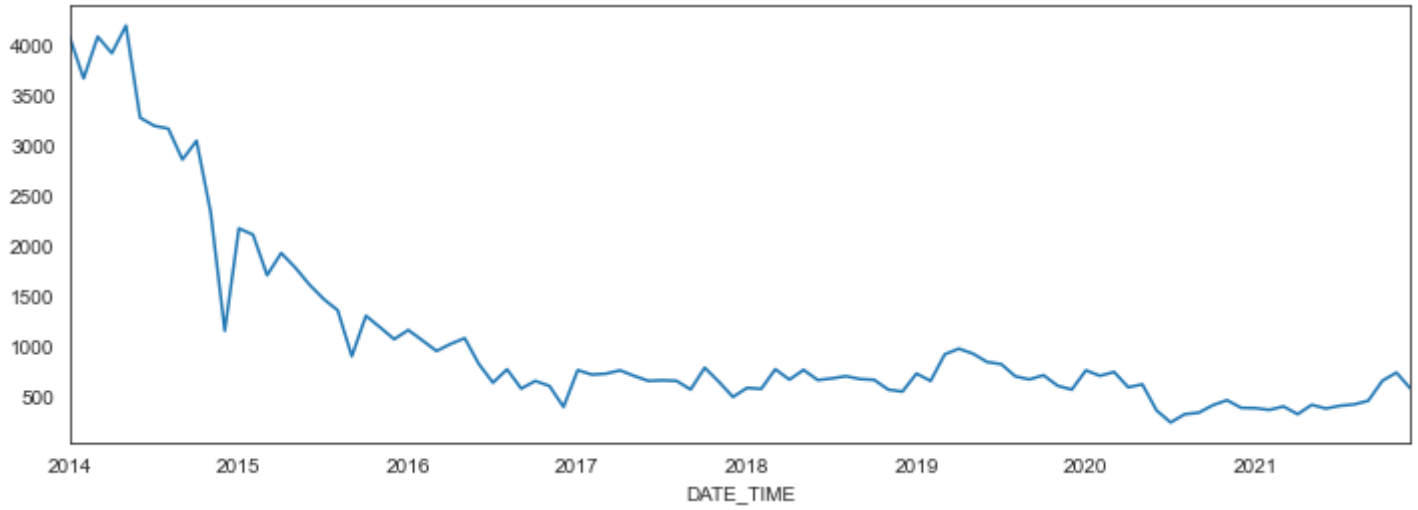


In [34]:

```
fourteen_by_month = fourteen_onward['SUSPECT_ARRESTED_FLAG_N'].resample('M').sum()
fourteen_by_month.plot()
# resampling by month and taking a look
```

executed in 163ms, finished 10:25:19 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```



Observations: when resampling by month, we can almost see two different vauge patterns in the time data, from 2014 - 2018 there are dips each year nearly on the year mark. In 2019 this dip is more slight

and even slighter in 2020 where the biggest dip looks to be in the summer. This was at the hight of the pandemic in New York City, when the police were very active.

There is not an obvious pattern however, which doesn't bode well for trying to model this data on a yearly season.

Monthly vs Weekly frequencies to model:

I wanted to try modeling both on a weekly frequency and on a monthly frequency to see which would yield a better MAE score. Realistically, this model would be theoretically utilized to forecast three months into the future on a quarterly cycle, to be able to best integrate into how NYPD reports to the city. Therefore, the monthly version of this modeling process (this version of the project) is the final notebook.

5.3 only after the new data system upgrade

Due to the complete overhaul of how the data was being recorded in 2018 in response to Floyd vs The City of New York, it's worth it for EDA purposes to take a look at the time line from 2018 onwards.

In [35]:

```
eighteen_onward = df_dummied.loc['2018':]
eighteen_onward.head()

# making 2018 and onward it's own DF and taking a look
```

executed in 10ms, finished 10:25:19 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	SUSPECT_RACE_DESCRIPTION_BLACK	SUSPECT_RACE_DESCR
2018-01-01	1	0	1	0
2018-01-01	0	1	1	0
2018-01-01	0	1	1	0
2018-01-01	0	1	1	0
2018-01-01	0	1	1	0

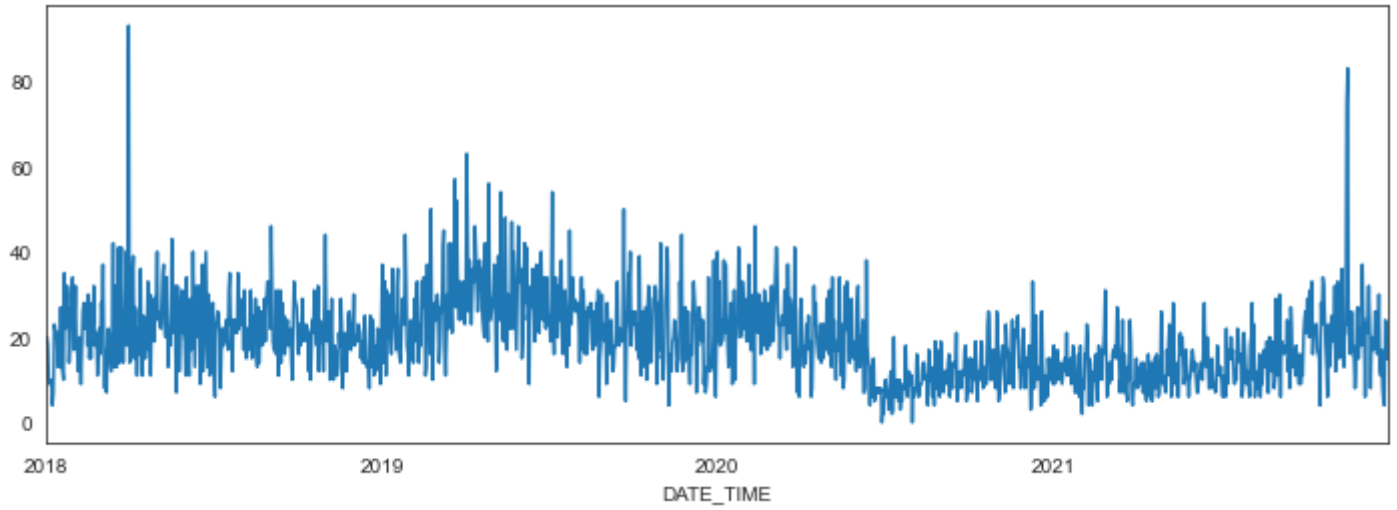
5 rows × 89 columns

In [36]:

```
eighteen_by_day = eighteen_onward['SUSPECT_ARRESTED_FLAG_N'].resample('D').sum()
eighteen_by_day.plot()
# resampling by day and taking a look
```

executed in 112ms, finished 10:25:19 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```



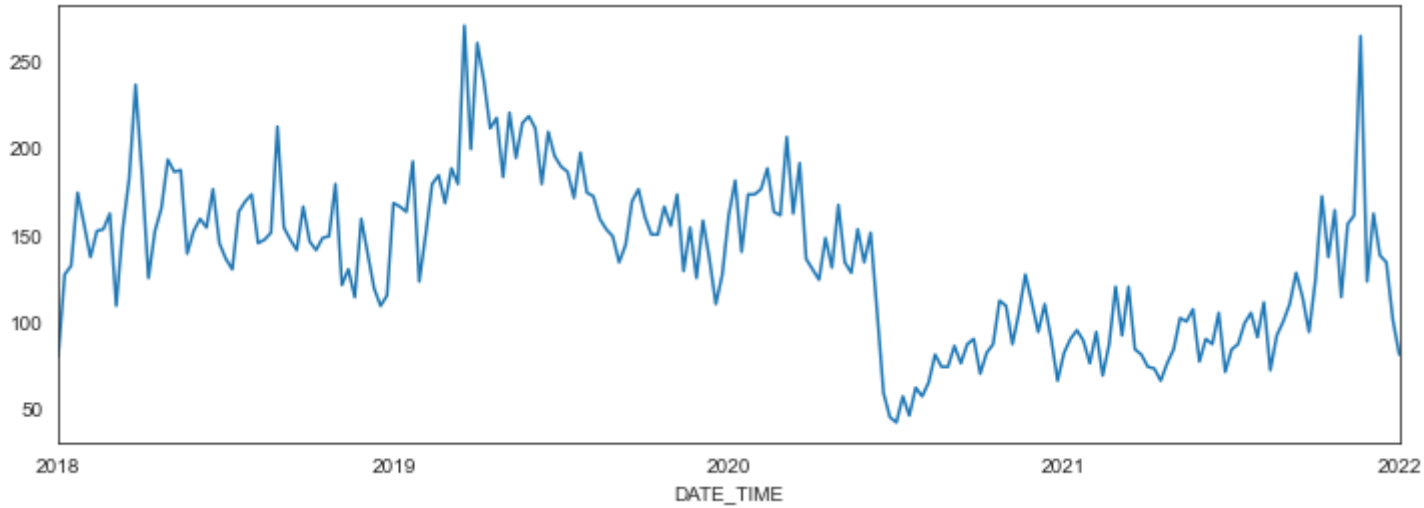
Observations: This data actually looks stationary and has a very vauge yearly pattern (which breaks down in 2020 summer.) Let's see if we can't see a better pattern by resampling weekly.

In [37]:

```
eighteen_by_week = eighteen_onward['SUSPECT_ARRESTED_FLAG_N'].resample('W').sum()
eighteen_by_week.plot()
# resampling on a weekly basis
```

executed in 111ms, finished 10:25:19 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```



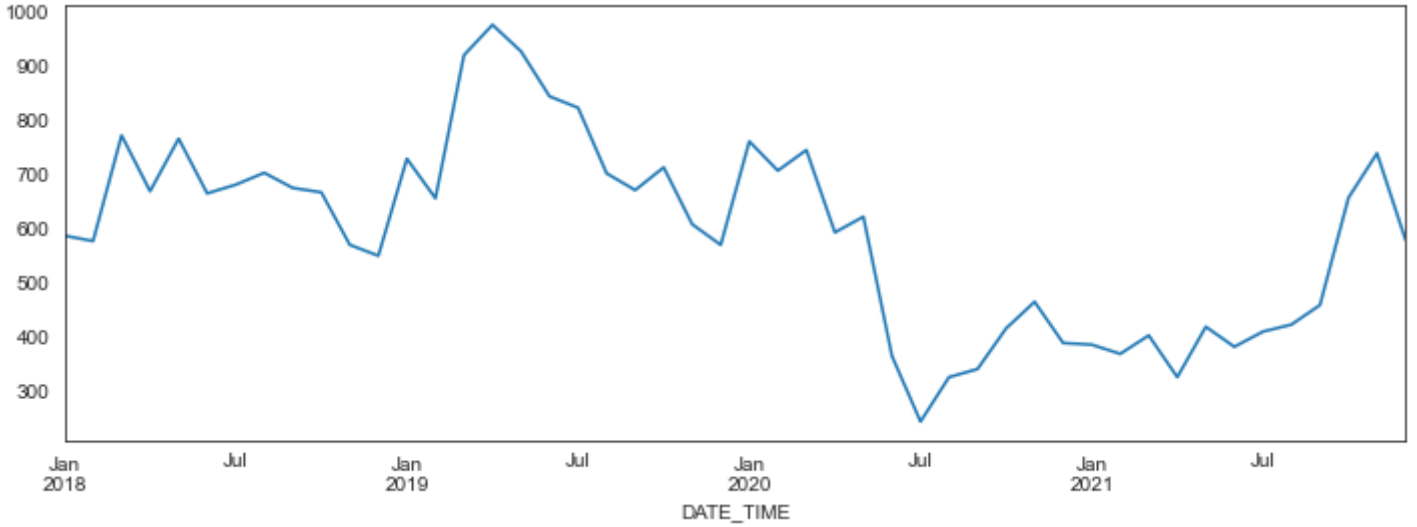
Observations: This data is def more stationary however the patterns are very loose if they exist at all.

In [38]:

```
eighteen_by_month = eighteen_onward['SUSPECT_ARRESTED_FLAG_N'].resample('M').sum()
eighteen_by_month.plot()
```

executed in 215ms, finished 10:25:19 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME'>
```



Observations: we see a dip in Jan 2019, a dip in Jan 2020 but that pattern is not continued in Jan 2021, perhaps due to the pandemic. Interestingly, stops that do not lead to an arrest rate really plummets in July 2020 but picks up hugely in the fall of 2021. Is this NYC getting back to it's normal non-pandemic routines, or is this due to the election of Mayor Adams?

6 Getting the Not-Arrested Rate by month 2014-onward

In [39]:

```
fourteen_onward['COUNTS'] = 1
fourteen_onward.head()

# by having a uniform column of 1 I can use this to count all stops regardless
# of outcome during aggrigation
```

executed in 11ms, finished 10:25:19 2022-09-29

```
<ipython-input-39-e9ff0dc1b326>:1: 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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  fourteen_onward['COUNTS'] = 1
```

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	SUSPECT_RACE_DESCRIPTION_BLACK	SUSPECT_RACE_DESCR
2014-01-01 00:10:00	1	0	1	0
2014-01-01 00:20:00	0	1	0	1
2014-01-01 00:30:00	1	0	1	0
2014-01-01 00:30:00	1	0	1	0
2014-01-01 00:30:00	1	0	1	0

5 rows × 90 columns

6.0.1 Making a Not Arrested Rate

In [40]:

```
fourteen_onward = fourteen_onward.resample('M').sum()
# resampling by month
fourteen_onward['NOT_ARRESTED_RATE'] = fourteen_onward['SUSPECT_ARRESTED_FLAG_N'] / fourteen_onwar
fourteen_onward.head()
# making a rate of stops that do not lead to an arrest to make sure the dips
# and peeks in the data is all on the same scale as itself, then look for patterns
```

executed in 69ms, finished 10:25:19 2022-09-29

DATE_TIME	SUSPECT_ARRESTED_FLAG_N	SUSPECT_ARRESTED_FLAG_Y	SUSPECT_RACE_DESCRIPTION_BLACK	SUSPECT_RACE_DESCR
2014-01-31	4073.0	863.0	2710.0	315.0
2014-02-28	3658.0	675.0	2206.0	271.0
2014-03-31	4073.0	753.0	2472.0	289.0
2014-04-30	3908.0	672.0	2376.0	273.0
2014-05-31	4182.0	613.0	2583.0	276.0

5 rows × 91 columns

In [41]:

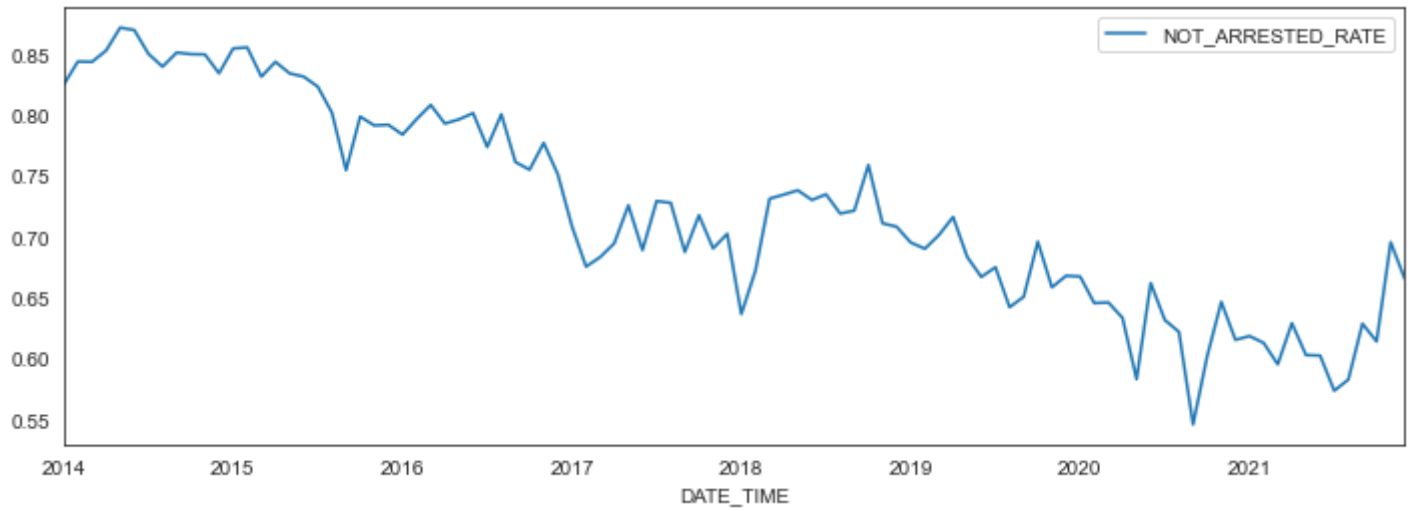
```
fourteen_not_arrested = fourteen_onward.drop(columns= [col for col in fourteen_onward.columns if r
# dropping all columns that are not the not arrested rate, as that's what I'll
# use to model. in future work I would like to add more variables.
```

executed in 3ms, finished 10:25:19 2022-09-29

In [42]:

```
fourteen_not_arrested_plot = fourteen_not_arrested.plot()
# taking a look at the Not Arrested Rate
```

executed in 171ms, finished 10:25:19 2022-09-29



Observations: this data doesn't look random but neither does it show an obvious pattern. There is more of an obvious pattern from 2014-2018 where we see a cluster of four small hills and then a drop. This pattern changes a bit from 2018 to 2020 and from 2020-2021 we're looking at a something that doesn't resemble a pattern at all.

In [43]:

```
fourteen_not_arrested.describe()
# taking a look at the descriptive statistics to get a better sence of the data
```

executed in 7ms, finished 10:25:19 2022-09-29

	NOT_ARRESTED_RATE
count	96.000000
mean	0.723864
std	0.084747
min	0.545455
25%	0.660978
50%	0.716972
75%	0.797354
max	0.872158

Observations:

The average rate of not arrest is 72%. This feels very, very high for how expensive these stops are to new york city, as was reflected in other EDA.

The Standard Deviation for the not-arrested rate is about 8 and a half percent meaning the normal amount for these values to range from the mean is about 8 and a half percent. Therefore, the model we build is only really going to be of value if it can predict the variance within that range.

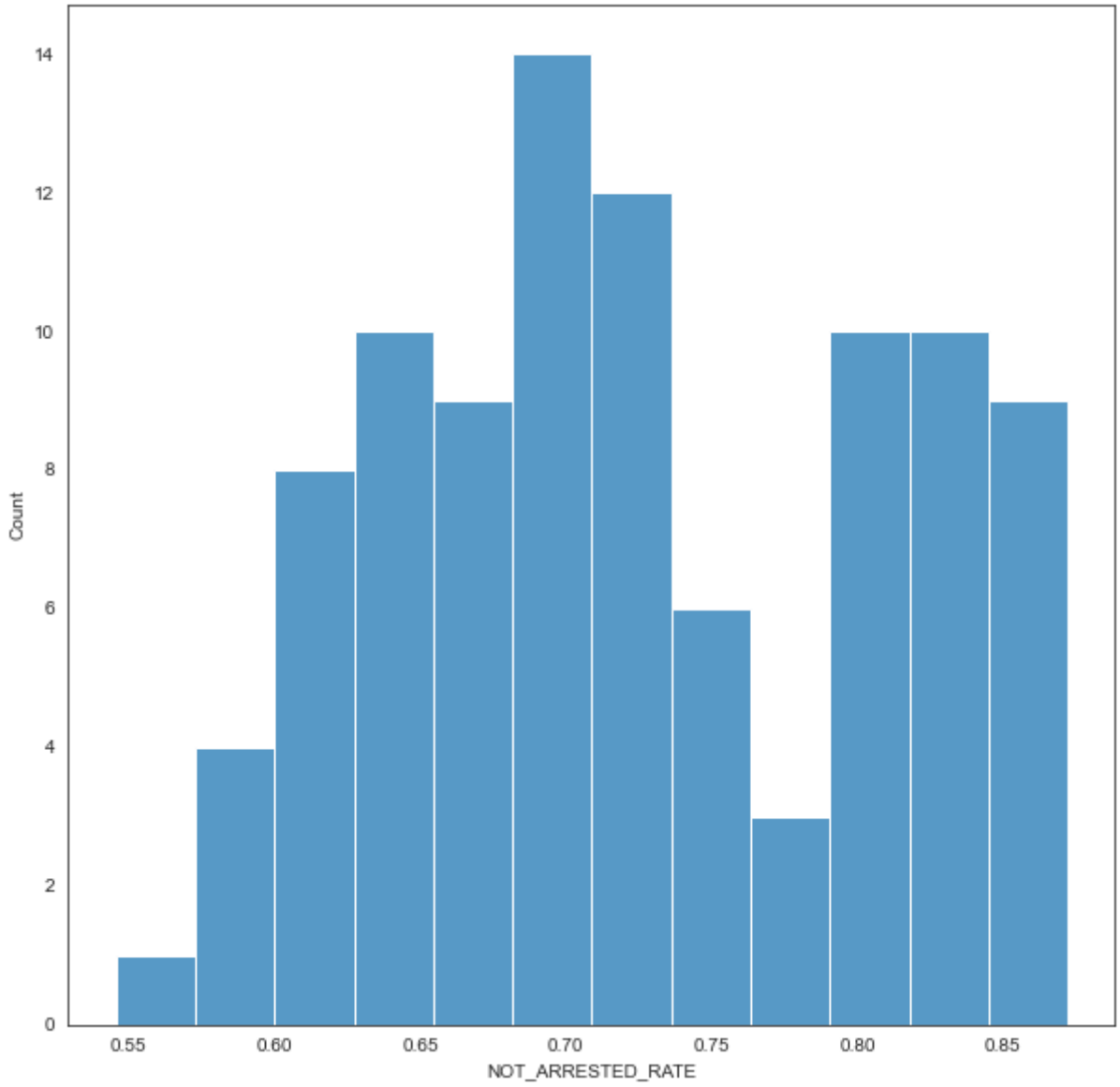
Let's take a better look at how this data is distributed.

In [44]:

```
plt.figure(figsize=(10, 10))
# making the shape of the graph

sns.histplot(data = fourteen_not_arrested, x = 'NOT_ARRESTED_RATE', bins= 12)
# making a histogram with 12 bins since there are 12 months in the year
```

executed in 101ms, finished 10:25:19 2022-09-29

```
<AxesSubplot:xlabel='NOT_ARRESTED_RATE', ylabel='Count'>
```



Observations: the distribution isn't a normal distribution exactly, it skews and is somewhat split. What's very interesting is it dips right above the mean value, and months where the not-arrested rate was just below the mean value are most common.

In [45]:

```
# Determine rolling statistics
roll_mean = fourteen_not_arrested.rolling(window=6, center=False).mean()
roll_std = fourteen_not_arrested.rolling(window=6, center=False).std()

```

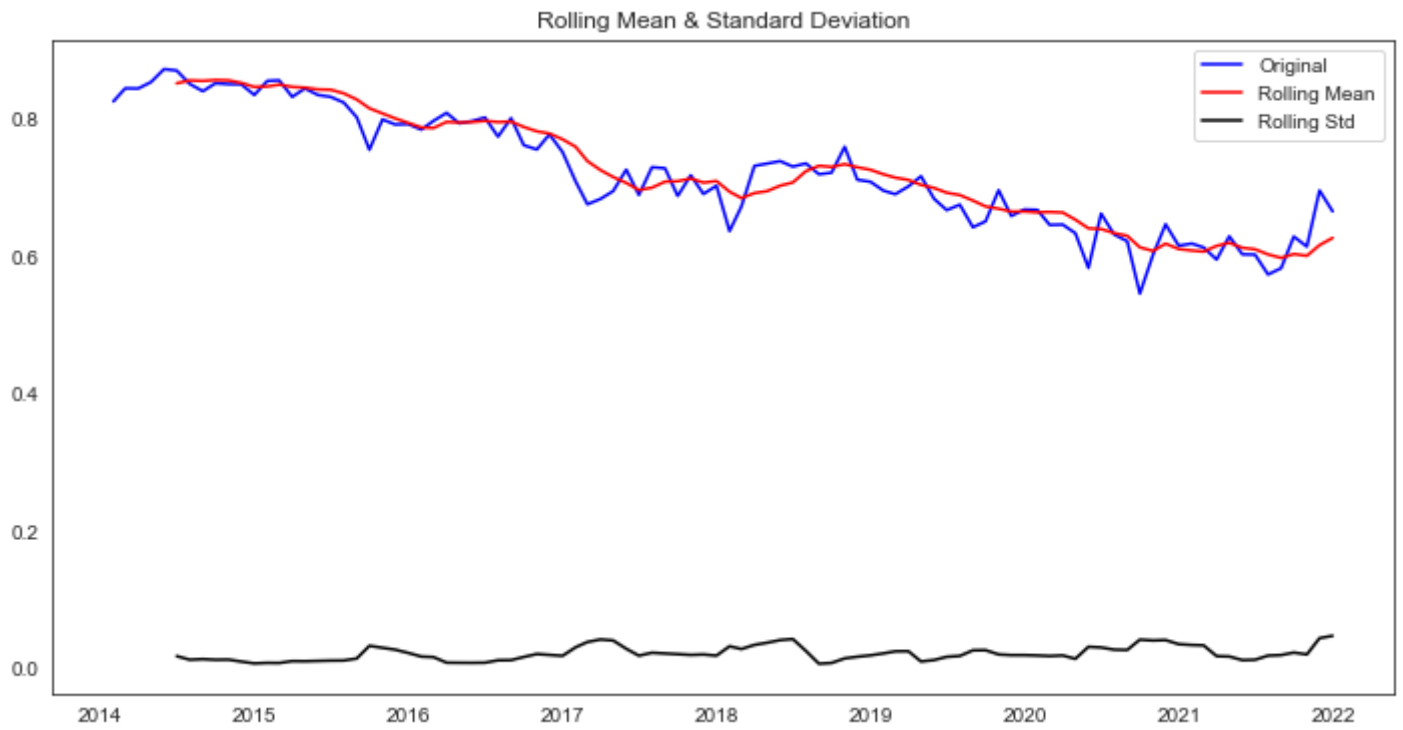
executed in 3ms, finished 10:25:19 2022-09-29

Because this model is attempting to forecast 3 months in advance, I've set the window to 3, and am getting a rolling mean and rolling standard deviation. Let's take a look at what those look like.

In [46]:

```
# Plot rolling statistics

fig = plt.figure(figsize=(12,6))
# Make the plot a differnt shape
plt.plot(fourteen_not_arrested, color='blue',label='Original')
# plotting the original data in blue
plt.plot(roll_mean, color='red', label='Rolling Mean')
# Plotting the rolling mean in red
plt.plot(roll_std, color='black', label = 'Rolling Std')
# plotting the rolling standard deviation in black
plt.legend(loc='best')
# making the legend
plt.title('Rolling Mean & Standard Deviation')
#m making the title
plt.show()
# showing the plot
```

executed in 130ms, finished 10:25:20 2022-09-29



Observations: looking at the rolling mean did not make the mean stationary so we know we can already rule out using an AR or MA model, despite the rolling making the standard deviation stationary.

7 Dickey-Fuller Test

In [47]:

```
from statsmodels.tsa.stattools import adfuller

# Perform Dickey-Fuller test:
print ('Results of Dickey-Fuller Test: \n')
dftest = adfuller(fourteen_not_arrested['NOT_ARRESTED_RATE'])

# Extract and display test results in a user friendly manner
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags Used', 'Number of Obs
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

executed in 10ms, finished 10:25:20 2022-09-29

```
Results of Dickey-Fuller Test:

Test Statistic                -1.408935
p-value                        0.577985
#Lags Used                     2.000000
Number of Observations Used    93.000000
Critical Value (1%)           -3.502705
Critical Value (5%)           -2.893158
Critical Value (10%)          -2.583637
dtype: float64
```

observations: the P value is above 0.05 so we can reject the null hypothesis, this time series is not stationary and cannot be modeled by ARMA types of models.

In [48]:

```
adf_test = ADFTest(alpha=0.05)
p_val, should_diff = adf_test.should_diff(fourteen_not_arrested)  # (0.01, False)
print(f'{p_val} : {should_diff}')
```

executed in 5ms, finished 10:25:20 2022-09-29

```
0.5284323052814037 : True
```

Observations: this is the augmented Dicky Fuller test, telling us we should take a first difference for modeling purposes.

In [49]:

```
from pmdarima.arima.utils import ndiffs

n_adf = ndiffs(fourteen_not_arrested, test='adf')
print(n_adf)

# according to auto arima, the I value in an ARIMA or SARIMA model should have
# a maximum value of 1
```

executed in 7ms, finished 10:25:20 2022-09-29

```
1
```

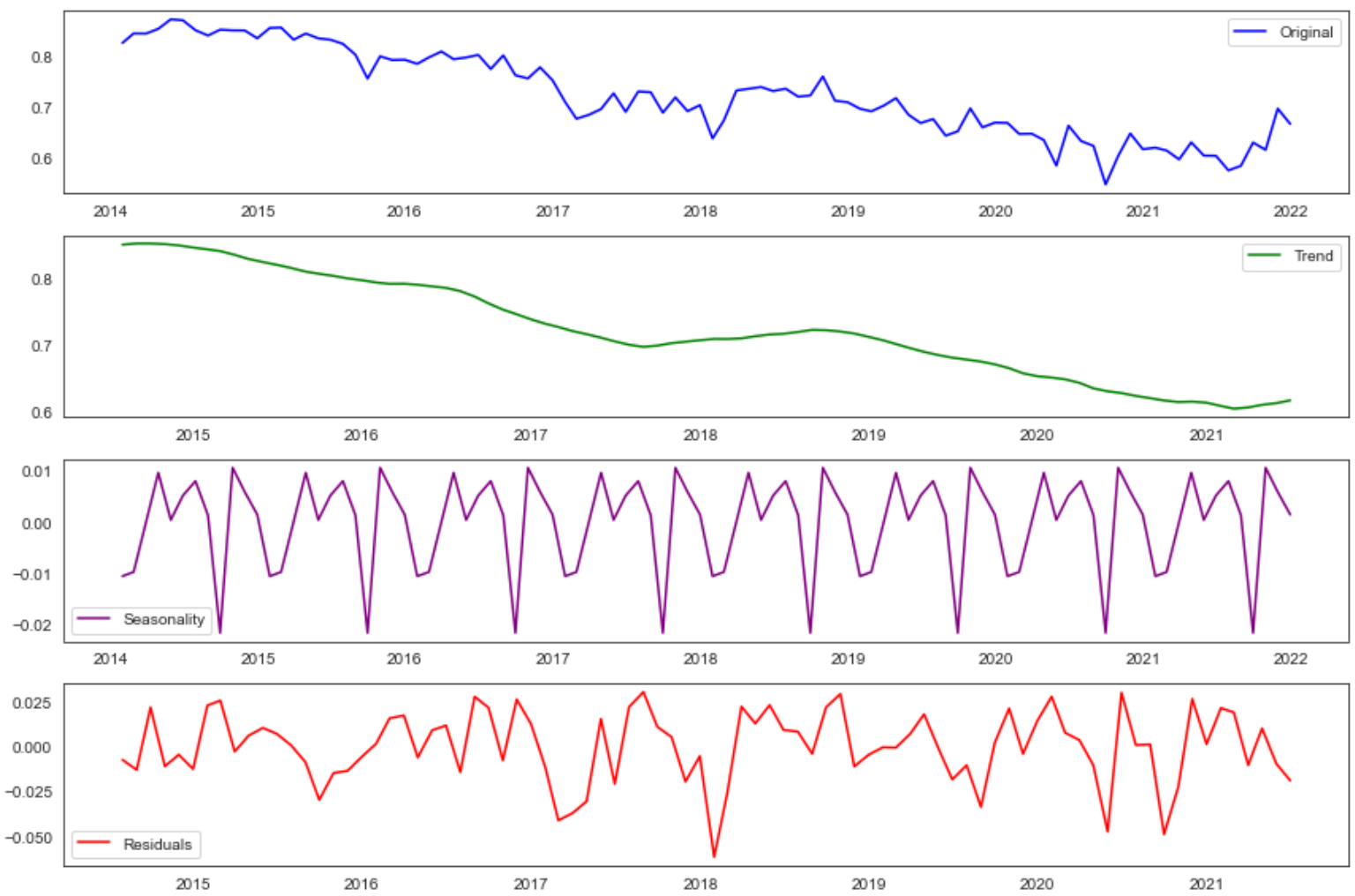
7.1 Decomposition

In [50]:

```
# Import and apply seasonal_decompose()
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(fourteen_not_arrested, model = "additive")

# Gather the trend, seasonality, and residuals
trend = decomposition.trend
# trend is the overall tilt of the data
seasonal = decomposition.seasonal
# seasonal is an reoccuring patterns
residual = decomposition.resid
# residual represents the noice in the data set that is neither explained
# by seasonal cycles or trends

# Plot gathered statistics
plt.figure(figsize=(12,8))
# changing the shape of the plot
plt.subplot(411)
# making the subplot
plt.plot(fourteen_not_arrested, label='Original', color='blue')
# taking a look at the original data in blue
plt.legend(loc='best')
# making the label stick
plt.subplot(412)
# making another subplot
plt.plot(trend, label='Trend', color='green')
# making the trend line green
plt.legend(loc='best')
# making the label stick
plt.subplot(413)
# making another subplot
plt.plot(seasonal,label='Seasonality', color='purple')
# making seasonaility purple
```

```
plt.legend(loc='best')
# making the label stick
plt.subplot(414)
# making the final subplot
plt.plot(residual, label='Residuals', color='red')
# making the noise red
plt.legend(loc='best')
# making the label stick
plt.tight_layout()
# making it all one thing
```

executed in 534ms, finished 10:25:20 2022-09-29



Observations:

- this data has an overall trend line going down (Green)
- there is repeating seasonality in this data (Purple) however that seasonality only accounts for a total range of 3 percent of the variance in the data, which is hardly anything compared to the noise in the model
- the noise in the data only accounts for about 20% of the overall rate of not-arrest,(Red) meaning that 80% of the data *is* following the trend and seasonality

WHY MODEL IF THIS DATA SET DOESN'T LOOK TO LEND ITSELF TO MODELING

This data set does not look like it would be a good candidate for modeling as is, by itself. However, I think it's worth it to try and see--if the model cannot predict peeks accurately (times when stops are unlikey to lead to an arrest,) perhaps it can predict a reasonable range for this statistic and can be used to for identification of anomalies, letting NYPD or the city know that rates-of-not-arrest are particularly high if the peak is above the predicted confidence interval.

8 Linear Train Test Split

In [51]:
```
1 fourteen_onward.sum()
```

executed in 8ms, finished 10:25:20 2022-09-29

```
SUSPECT_ARRESTED_FLAG_N                   104008.000000
SUSPECT_ARRESTED_FLAG_Y                    31333.000000
SUSPECT_RACE_DESCRIPTION_BLACK             74355.000000
SUSPECT_RACE_DESCRIPTION_BLACK HISPANIC     9746.000000
SUSPECT_RACE_DESCRIPTION_E. ASIAN           5260.000000
                                                ...
PRECINCT_121                                3249.000000
PRECINCT_122                                2117.000000
PRECINCT_123                                 835.000000
COUNTS                                    135341.000000
NOT_ARRESTED_RATE                             69.490906
Length: 91, dtype: float64
```

I'll be using a linear train test split, as the idea of this model is that it will eventually be made into a pipeline with new values adding to the training set on a regular basis.

In [52]:
```
1 valid = fourteen_not_arrested.tail(6)
2 # making the validation set
```

executed in 2ms, finished 10:25:20 2022-09-29

In [53]:

```
train = fourteen_not_arrested.loc['2014': '2021-07-31']
# making the train set and taking a look
train
```

executed in 6ms, finished 10:25:20 2022-09-29

DATE_TIME	NOT_ARRESTED_RATE
2014-01-31	0.825162
2014-02-28	0.844219
2014-03-31	0.843970
2014-04-30	0.853275
2014-05-31	0.872158
...	...
2021-03-31	0.594993
2021-04-30	0.628846
2021-05-31	0.602582
2021-06-30	0.602201
2021-07-31	0.573222

91 rows × 1 columns

In [54]:

```
train.shape
# making sure these are the values we expect
```

executed in 2ms, finished 10:25:20 2022-09-29

```
(91, 1)
```

In [55]:

```
valid.shape
# making sure these are the values we expect
```

executed in 2ms, finished 10:25:20 2022-09-29

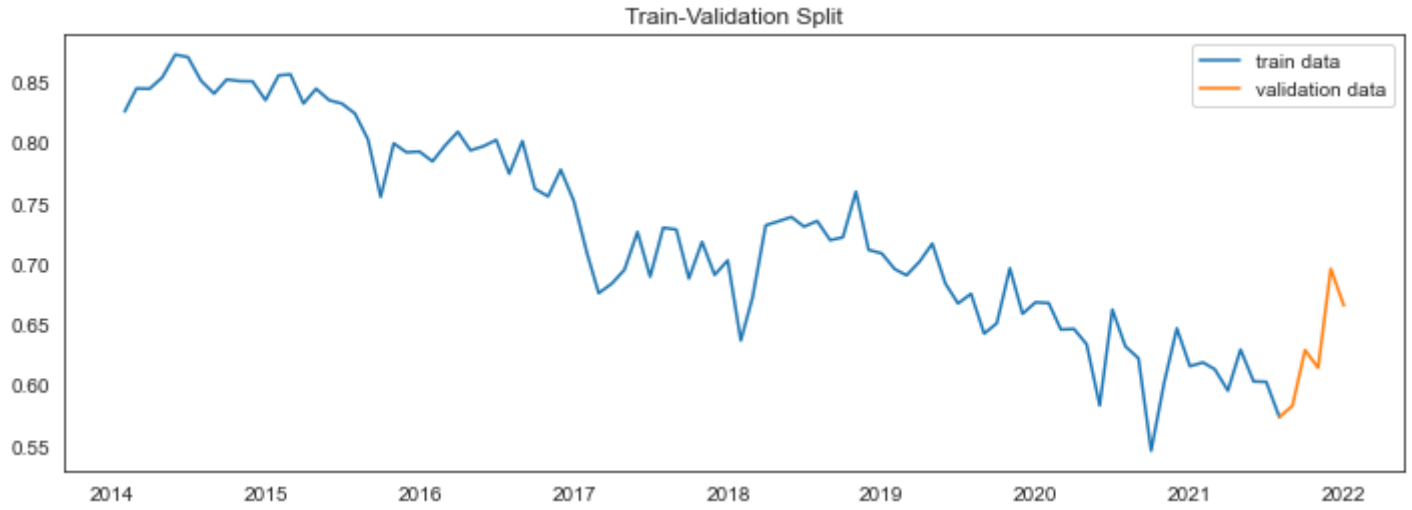
```
(6, 1)
```

In [56]:

```
# visualizing the split

fig, ax = plt.subplots()
ax.plot(train, label = 'train data')
ax.plot(valid, label = 'validation data')

ax.set_title("Train-Validation Split")

plt.legend()

# taking a look at our data
```

executed in 135ms, finished 10:25:20 2022-09-29

```
<matplotlib.legend.Legend at 0x7f806a6ca520>
```



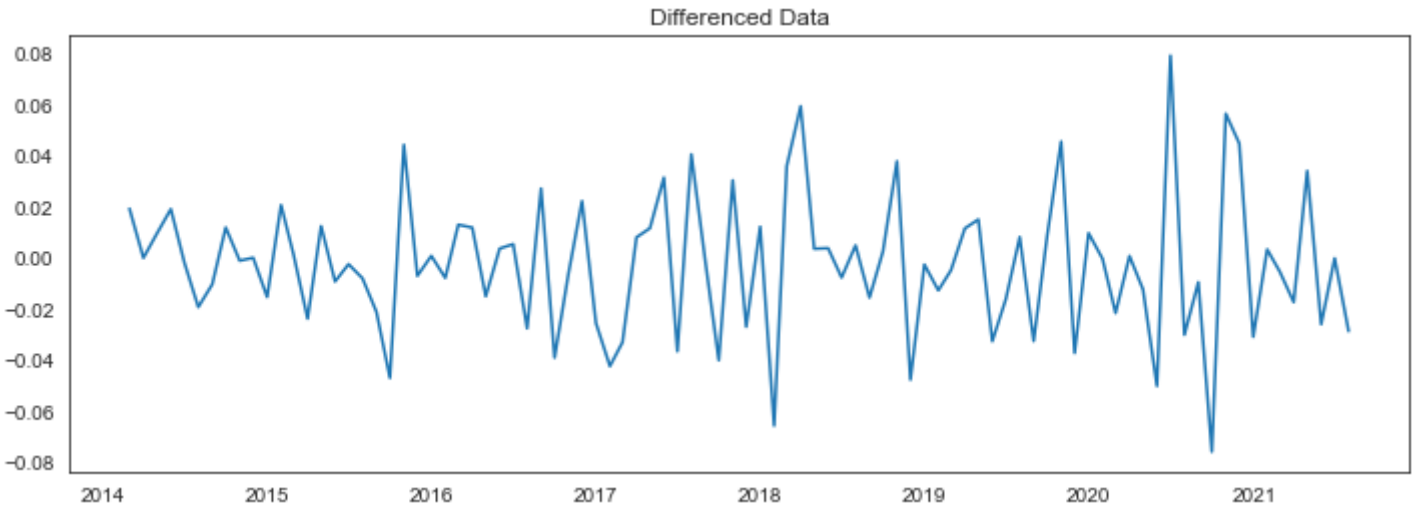
Observations: the model may have trouble! The cut off for the data starts at a particularly low point and starts peaking higher than it's been since 2019.

8.1 Investigating if a rolling difference can help us get rid of the trend

In [57]:
```
fig, ax = plt.subplots()
ax.plot(train.diff())
# making a plot that takes a first difference
ax.set_title("Differenced Data")
# observations: this gets rid of the trend but not the seasonaility
```

executed in 122ms, finished 10:25:20 2022-09-29

```
Text(0.5, 1.0, 'Differenced Data')
```



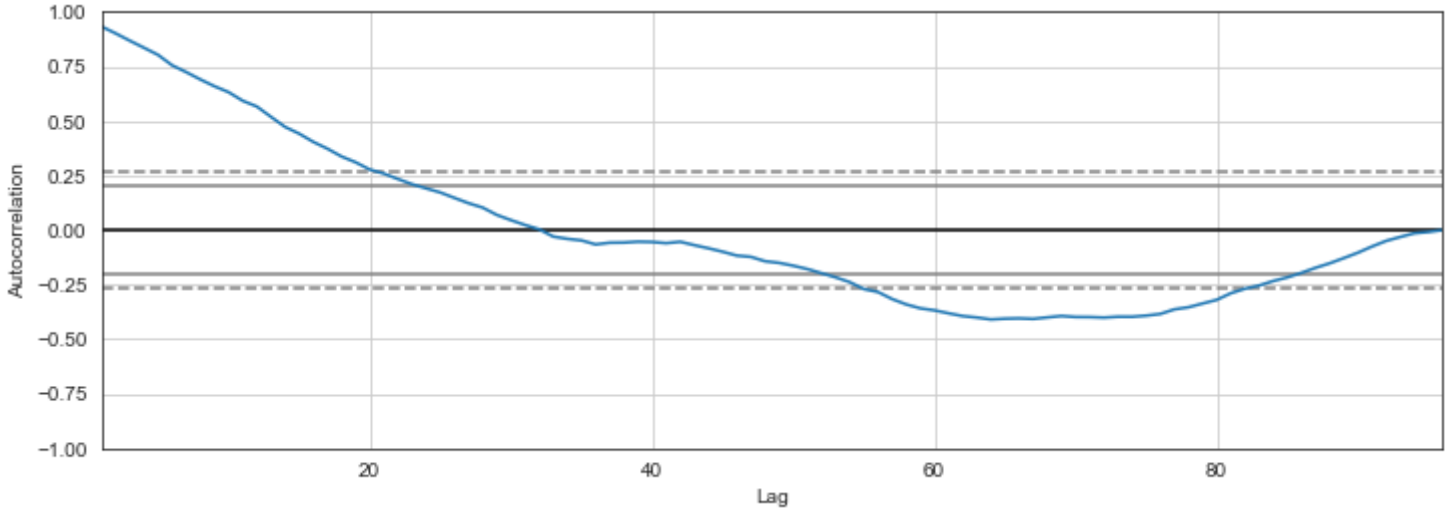
observations: this data is stationary and does not have any obvious patterns

In [58]:

```
1 from pandas.plotting import autocorrelation_plot
2
3 autocorrelation_plot(fourteen_not_arrested)
```

executed in 85ms, finished 10:25:20 2022-09-29

```
<AxesSubplot:xlabel='Lag', ylabel='Autocorrelation'>
```



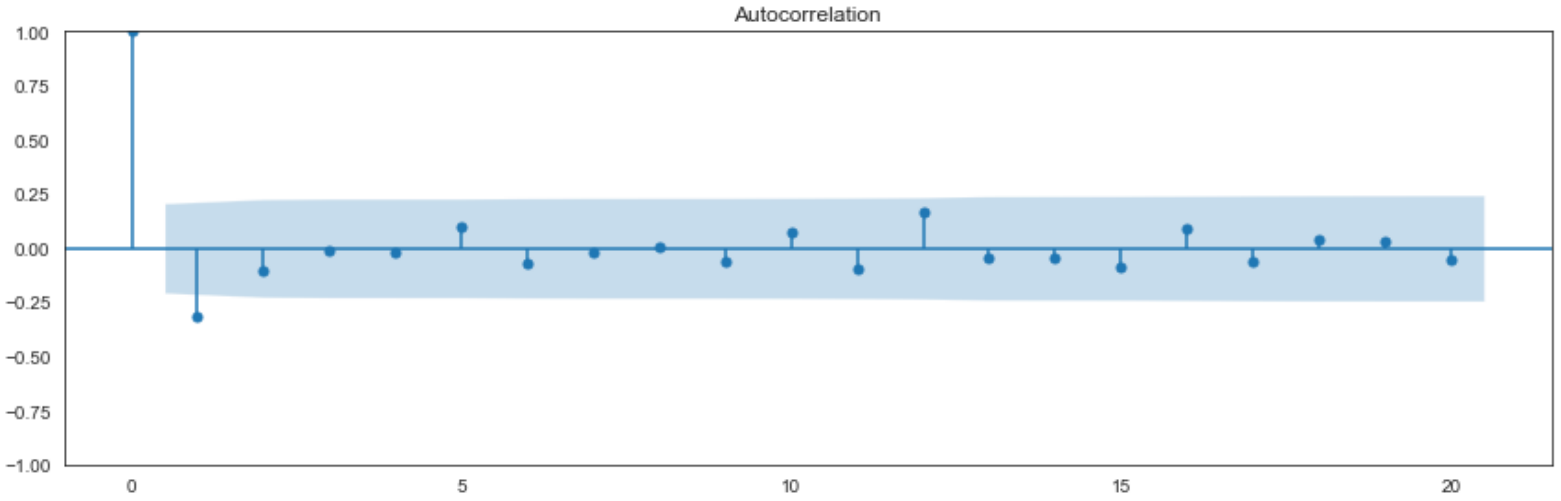
Observations: most of this plot exists outside of the confidence interval which represents that a majority of the data is not random, and has some correlation either negative or positive with itself. Where the correlation is strongest is within the first 1st-75th lags and again from the 250th-350th lag. Exploring the 52nd lag makes sense as it looked from the decomposition that there is a yearly pattern.

9 ACF and PACF

In [59]:

```
from statsmodels.graphics.tsaplots import plot_acf
# importing the relivant plot

plot_acf(train.diff().dropna())
# making sure we plot with a first difference since we know our model will
# have an I of 1

plt.tight_layout()
# making the plot non-douplicate
```

executed in 93ms, finished 10:25:21 2022-09-29



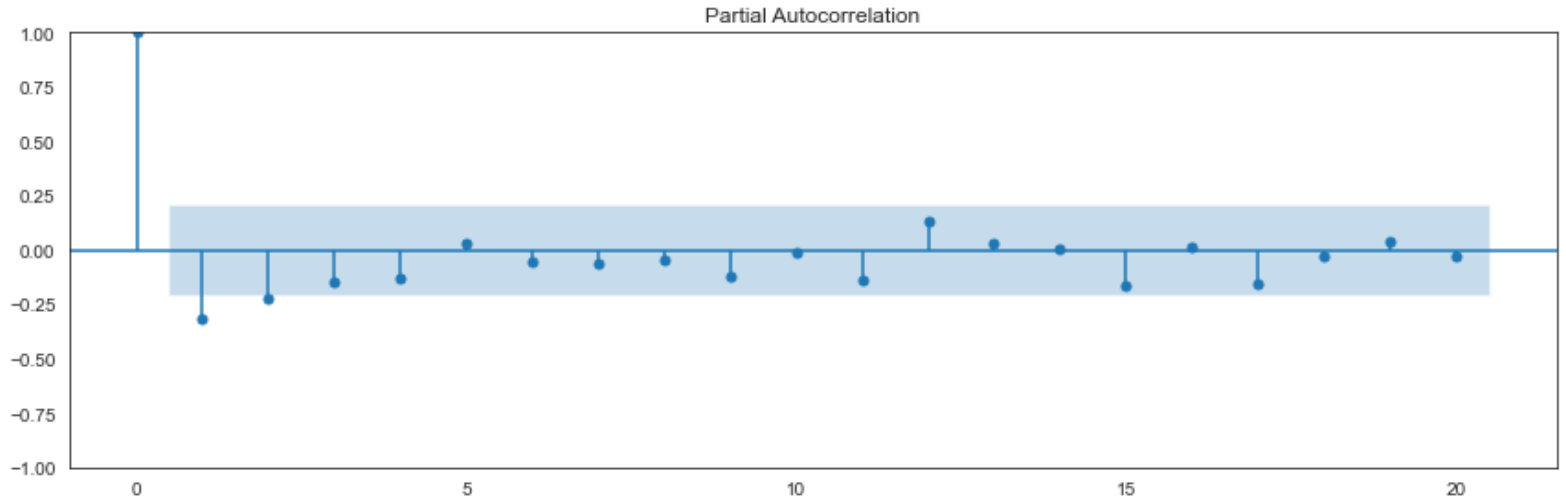
Observations: want to include 1 MA term maximum in final SARIMA Model as there is only one lollipop after the 0 that is statistically significant

In [60]:

```
from statsmodels.graphics.tsaplots import plot_pacf
# importing the plot

plot_pacf(train.diff().dropna())
# making sure to use the data with a first differene since I know I'll be
# using a first differnce

plt.tight_layout()
# making the plot non douplicate
```

executed in 95ms, finished 10:25:21 2022-09-29

```
/Users/b0ihazard/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/graphics/tsaplots.py:348: FutureW
arning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the default will ch
ange tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



Observations: want to include 1 MA maximum terms in final SARIMA Model as there is only one lollipop after the 0 that is statistically significant.

10 Baseline Model

As a baseline model I'll be shifting the data forward by an interval of one unit (month) and using that compare all my other models to.

In [61]:
```
naive = train.shift(1)
# making the naive model as simply the train set with a shift of one
naive_preds = naive[1::]
# starting the preds on the second value as the first will be blank
naive_preds.head()
# taking a look
```

executed in 5ms, finished 10:25:21 2022-09-29

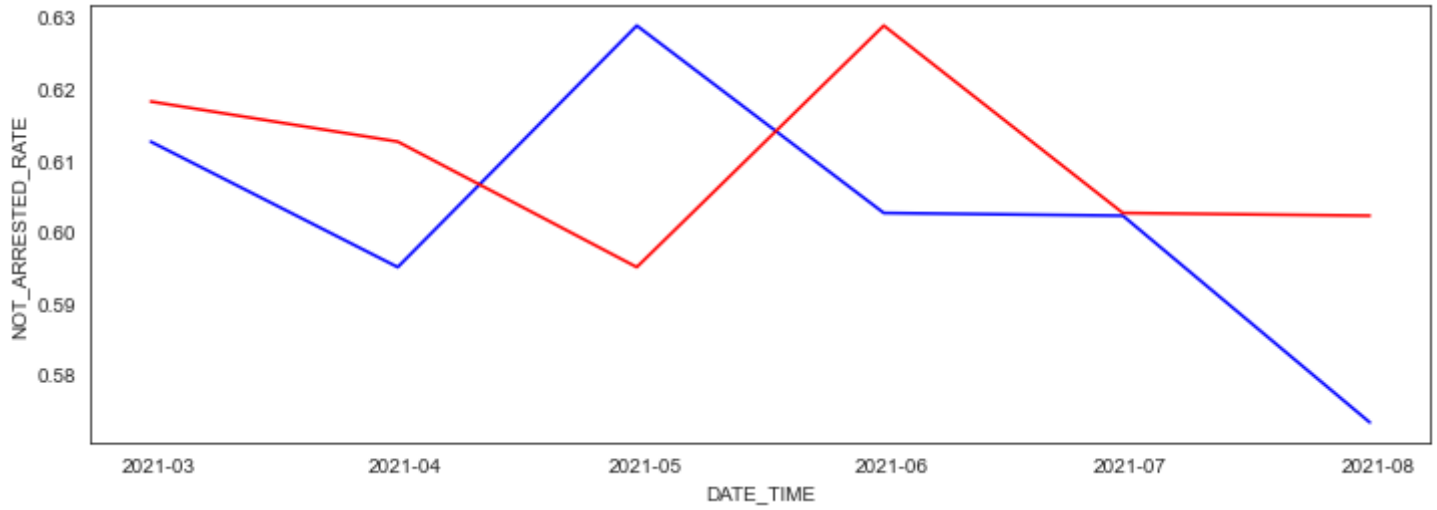
DATE_TIME	NOT_ARRESTED_RATE
2014-02-28	0.825162
2014-03-31	0.844219
2014-04-30	0.843970
2014-05-31	0.853275
2014-06-30	0.872158

In [62]:

```
train_last_six = train[-6::]
# taking the training data only from this last six months

naive_last_six = naive_preds[-6::]
# looking at the predictions only from this last six months

sns.lineplot(data = train_last_six,
                                    y = "NOT_ARRESTED_RATE",
                                    x = train_last_six.index,
                                   color = 'blue', )
# making a blue line to represent the last six months original data

sns.lineplot(data = naive_last_six,
                                y = "NOT_ARRESTED_RATE",
                                x = naive_last_six.index,
                               color = 'red')
# making a red line to represent this last six months naive predictions
```

executed in 141ms, finished 10:25:21 2022-09-29

```
<AxesSubplot:xlabel='DATE_TIME', ylabel='NOT_ARRESTED_RATE'>
```



Observations: these lines are pretty close to each other. The MAE score will tell the Mean of the Absolute Value of all the errors and I'll be using that as my main statistic for judging my models in this case. Since I'm using percentages, it's important to keep the results of the test interpretable.

In [63]:
```
1 train.shape
```

executed in 2ms, finished 10:25:21 2022-09-29

```
(91, 1)
```

In [64]:
```
1 naive.shape
```

executed in 3ms, finished 10:25:21 2022-09-29

```
(91, 1)
```

In [65]:

```
get_mae(train[1::], naive[1::])

# getting the mae score for this first model,
```

executed in 3ms, finished 10:25:21 2022-09-29

```
0.020871437460361183
```

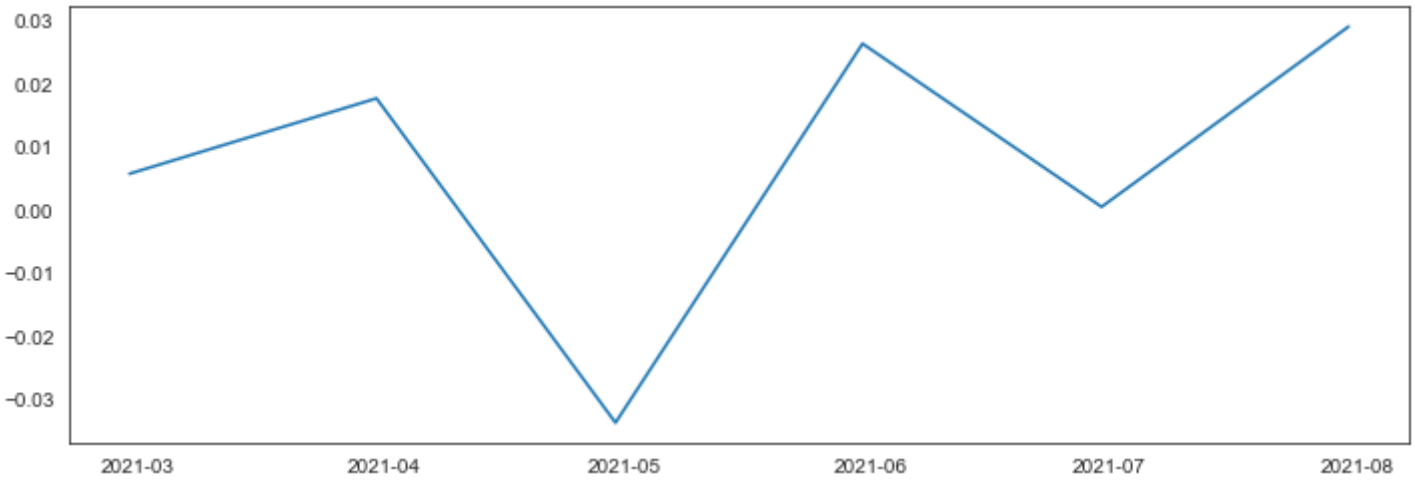
On average, the naive model's forecasts distance from the true value is 2%.

In [66]:

```
fig, ax = plt.subplots()
# make a plot
residuals_last_six = naive_preds[-6::] - train[-6::]
# get the residuals of last six months by subtracting the original data from the
# predictions from last six months

ax.plot(residuals_last_six.index, residuals_last_six)
# plotting the residuals to see over time where the model is farthest away
# from the original data

print(f'STANDARD DEVIATION OF RESIDUALS: {residuals_last_six.std()}')
```

executed in 119ms, finished 10:25:21 2022-09-29

```
STANDARD DEVIATION OF RESIDUALS: NOT_ARRESTED_RATE    0.02315
dtype: float64
```



The standard deviations of the residuals was 2%, meaning the residuals were on average about 4% away from the mean. This model struggled the most in May and August of 2021.

11 Random Walk Model

A random walk model is a model that is only using a first difference or I term in the ARIMA model, and using that to predict.

ARIMA(p,d,q)

- p is the order (number of time lags) of the autoregressive model, or the AR term
- d is the degree of differencing (the number of times the data have had past values subtracted) or the I term
- q is the order of the moving-average model or the MA term

In [67]:

```
random_walk_model = ARIMA(train, order = (0, 1, 0)).fit()
# making a model with only a first difference or a 1 for the d / I term
random_walk_model.summary()
# getting the summary of the model
```

executed in 25ms, finished 10:25:21 2022-09-29

SARIMAX Results

Dep. Variable:	NOT_ARRESTED_RATE	No. Observations:	91
Model:	ARIMA(0, 1, 0)	Log Likelihood	195.487
Date:	Thu, 29 Sep 2022	AIC	-388.975
Time:	10:25:21	BIC	-386.475
Sample:	01-31-2014	HQIC	-387.967
	- 07-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
sigma2	0.0008	0.000	7.311	0.000	0.001	0.001

Ljung-Box (L1) (Q):	9.18	Jarque-Bera (JB):	1.24
Prob(Q):	0.00	Prob(JB):	0.54

In [68]:

```
random_walk_model.aic

# ok I didn't get an AIC score for the naive model, so we don't have anything
# to compare this to, but let's keep this metric as something to compare future
# models to
```

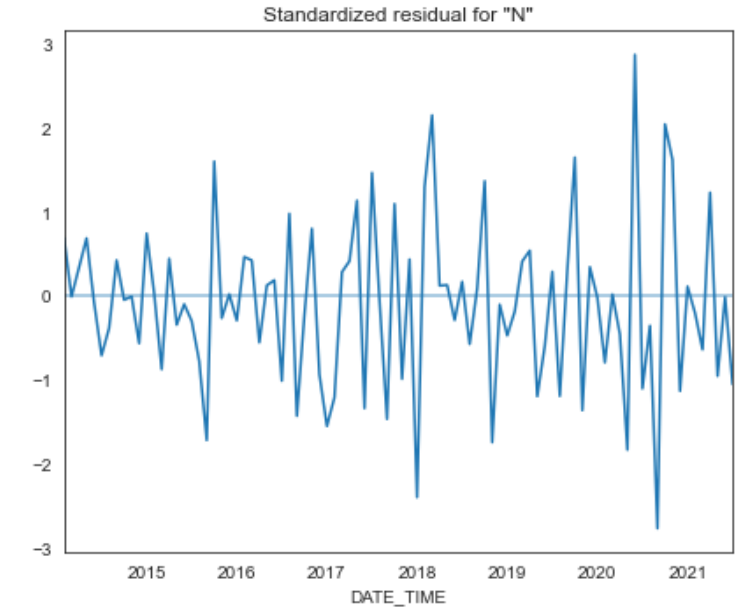
executed in 2ms, finished 10:25:21 2022-09-29

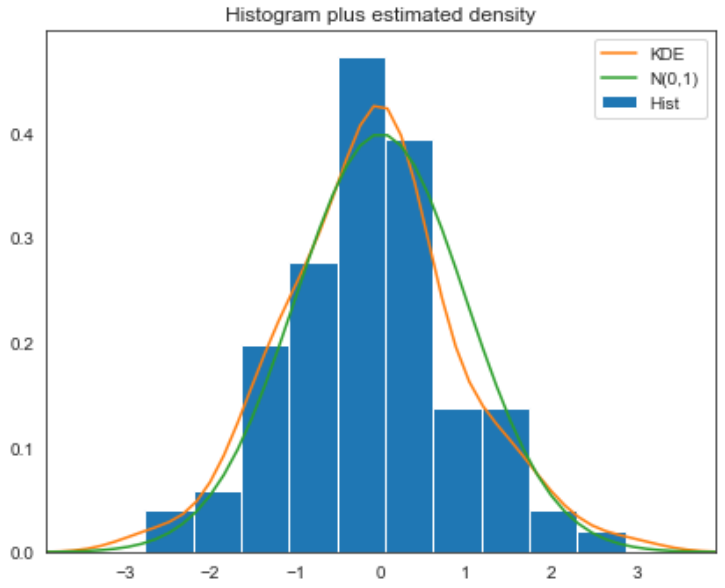
```
-388.97482570078154
```

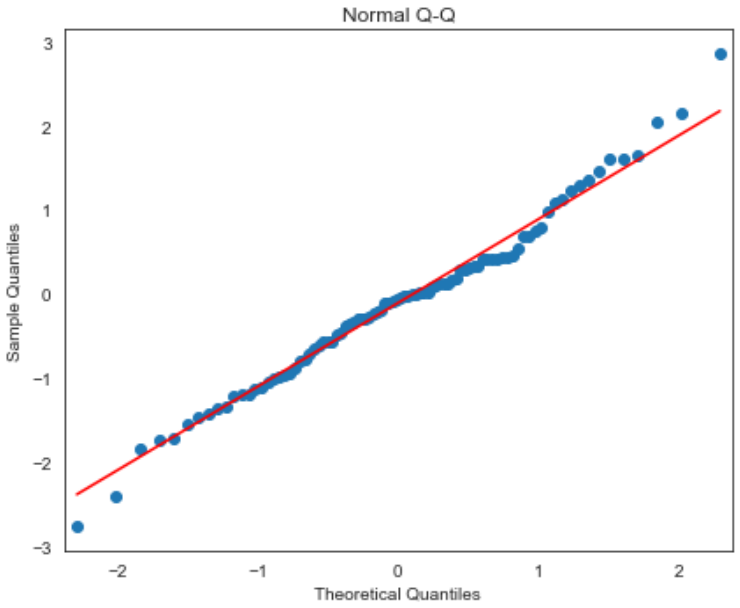
In [69]:

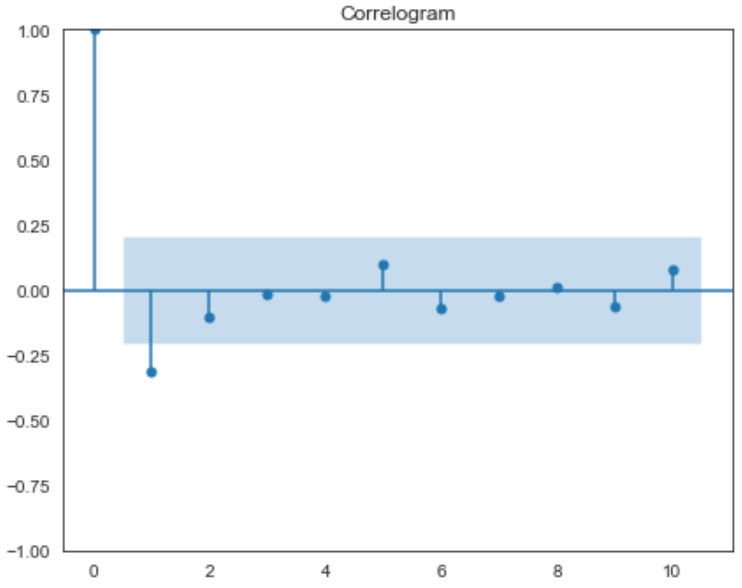
```
1 random_walk_model.plot_diagnostics(figsize=(15,12))
2 plt.show()
```

executed in 398ms, finished 10:25:21 2022-09-29









Observations:

1) the residuals show no obvious pattern 2) the predictions are mostly normally distributed 3) The data is mostly on the red line 4) There is one correlation that is statistically significant in the residuals that needs to be eliminated for future models to improve.

In [70]:

```
random_walk_preds = random_walk_model.predict(typ = 'levels')
# getting the predictions from the random walk model

random_walk_preds_last_six = random_walk_preds[-6::]
# grabbing only the preds from last six months

get_mae(train, random_walk_preds)
# getting the MAE score from the model
```

executed in 3ms, finished 10:25:21 2022-09-29

```
0.10228930522589381
```

On average, the random walk model's forecasts distance from the true value is 10%. This is much higher than our naive model!

In [71]:

```
forecast_last_six = random_walk_model.get_prediction(start = -6)
# getting the predictions from last six months

random_walk_conf_int_last_six = forecast_last_six.conf_int(alpha=0.05)
# getting the 95% confidence interval for the predictions last six months

random_walk_conf_int_last_six.head()
# taking a look at the range of the conf interval
```

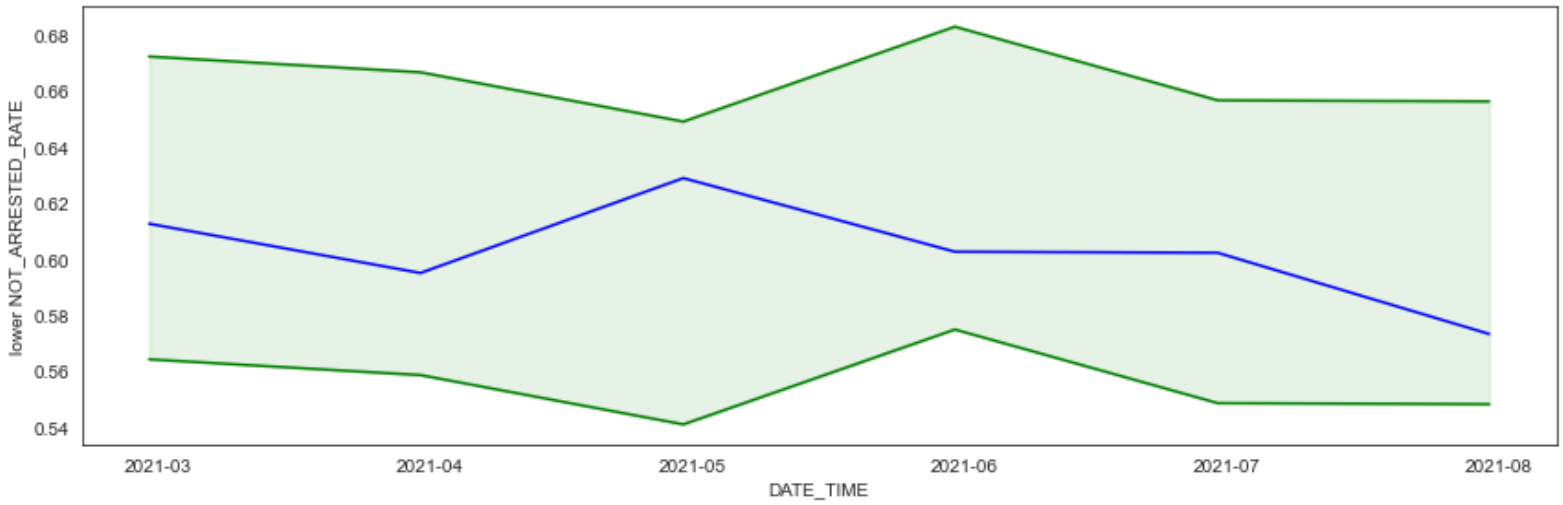
executed in 6ms, finished 10:25:21 2022-09-29

DATE_TIME	lower NOT_ARRESTED_RATE	upper NOT_ARRESTED_RATE
2021-02-28	0.564182	0.672240
2021-03-31	0.558554	0.666611
2021-04-30	0.540964	0.649021
2021-05-31	0.574817	0.682875
2021-06-30	0.548554	0.656611

In [72]:

```
last_six_visualize_preds(random_walk_conf_int_last_six, train[-6::])
# taking a look to see how well the data did or didn't fit into the confidence interval
```

executed in 173ms, finished 10:25:22 2022-09-29

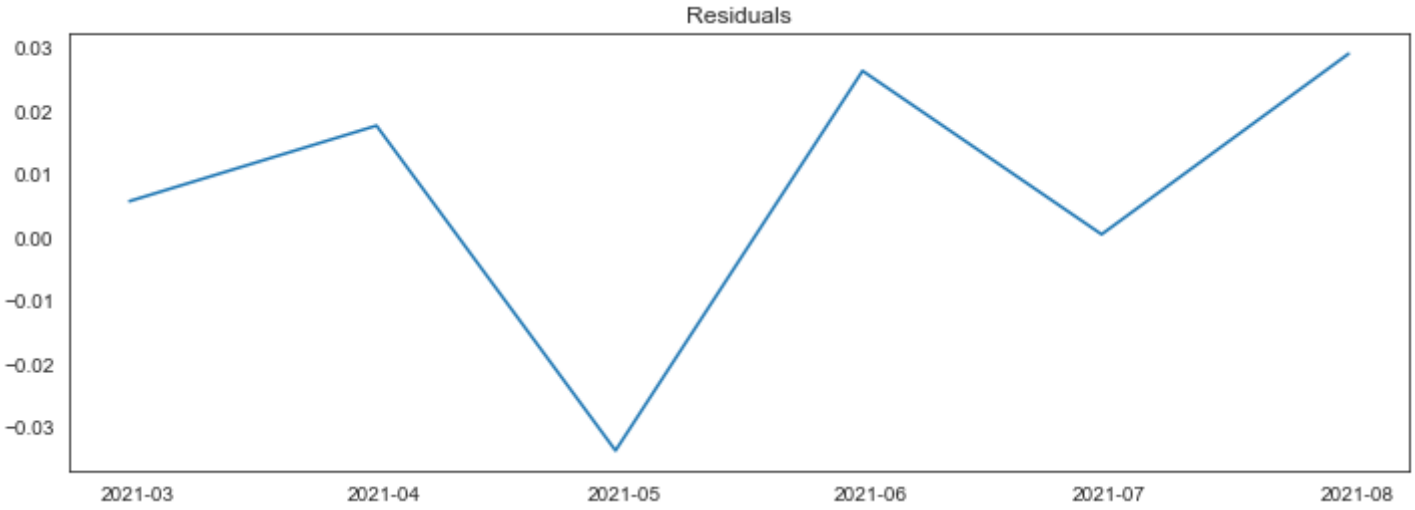


Observations: The data is sitting within the confidence interval. This is a good sign, but the predicted low during may of 2021 is actually when the rate was slightly higher, and the predicted high during june of 2021 is during a slight lul. The range of values this interval represents is between 54% and 68%--the range were most of the data lies anyway. This model while not inaccurate isn't accurate enough to be useful.

In [73]:
```
1 visualize_diagnostics(random_walk_preds_last_six, train[-6::])
```

executed in 122ms, finished 10:25:22 2022-09-29

```
STANDARD DEVIATION OF RESIDUALS: 0.023149725953247928
```



This model also struggled during may and august of last year, and has a very similar std of the residuals with only a slight advantage.

12 ARI Model

In [74]:

```
ari_1 = ARIMA(train, order=(1, 1, 0)).fit()
# making a ARI model with 1 auto regressive term and a first difference

ari_1.summary()
# getting the summary
```

executed in 22ms, finished 10:25:22 2022-09-29

SARIMAX Results

Dep. Variable:	NOT_ARRESTED_RATE	No. Observations:	91
Model:	ARIMA(1, 1, 0)	Log Likelihood	199.787
Date:	Thu, 29 Sep 2022	AIC	-395.573
Time:	10:25:22	BIC	-390.574
Sample:	01-31-2014	HQIC	-393.557
	- 07-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3045	0.091	-3.331	0.001	-0.484	-0.125
sigma2	0.0007	8.93e-05	7.724	0.000	0.001	0.001

Ljung-Box (L1) (Q):	0.60	Jarque-Bera (JB):	3.29
Prob(Q):	0.44	Prob(JB):	0.19
Heteroskedasticity (H):	3.22	Skew:	0.12
Prob(H) (two-sided):	0.00	Kurtosis:	3.90

Warnings:

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

In [75]:

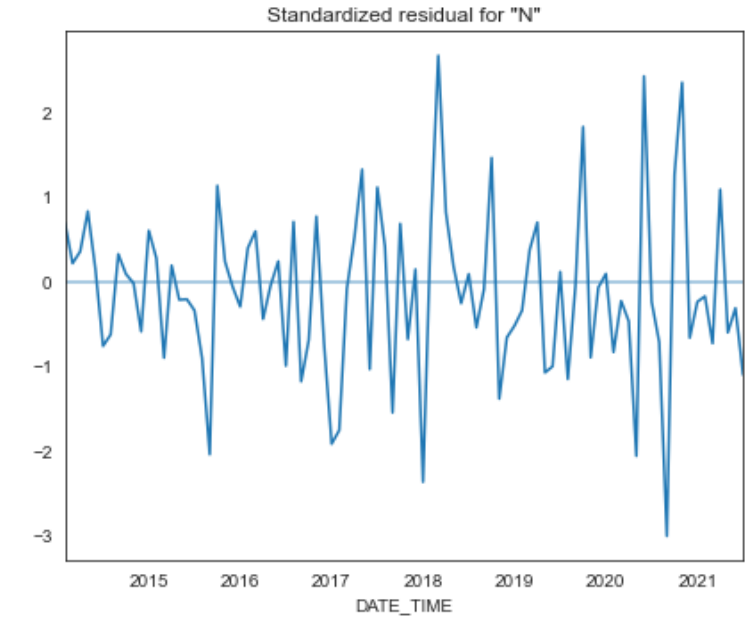
```
ari_1.aic

# ok we are moving in the correct direction! This is a lower number than our
# random walk model, so we know this model is better.
```

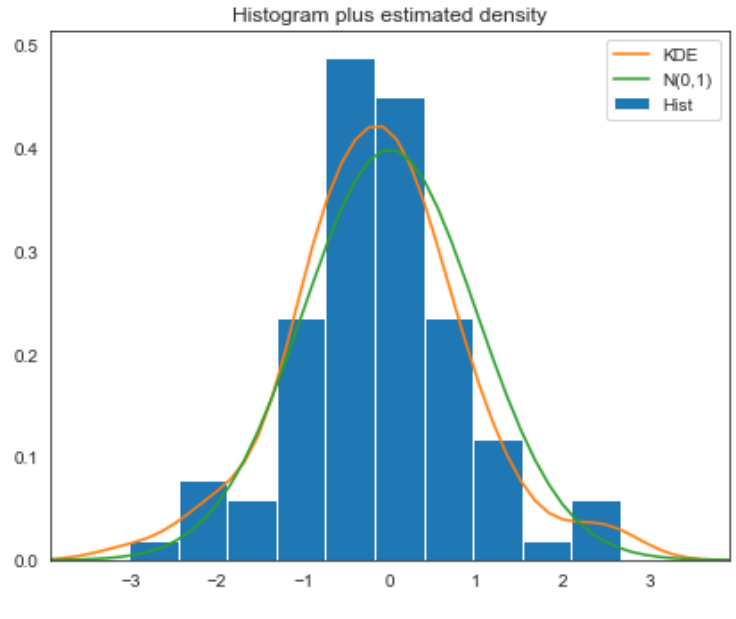
executed in 3ms, finished 10:25:22 2022-09-29

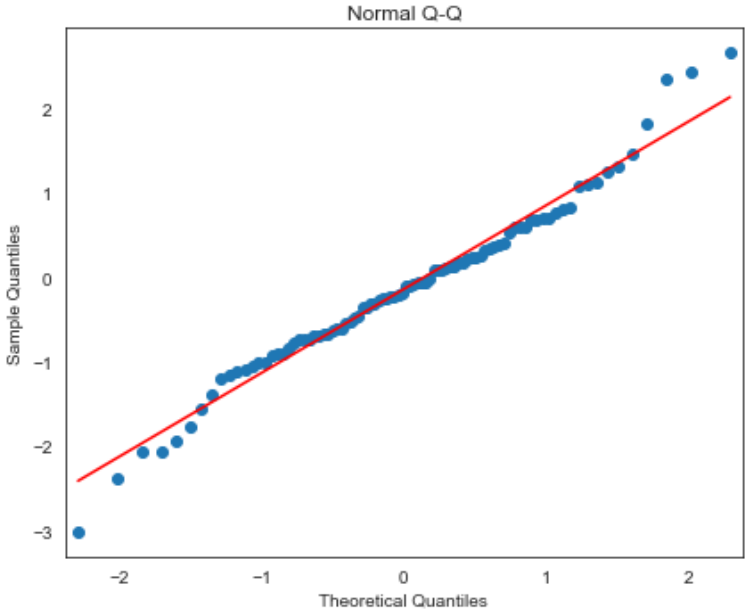
```
-395.5732590361313
```

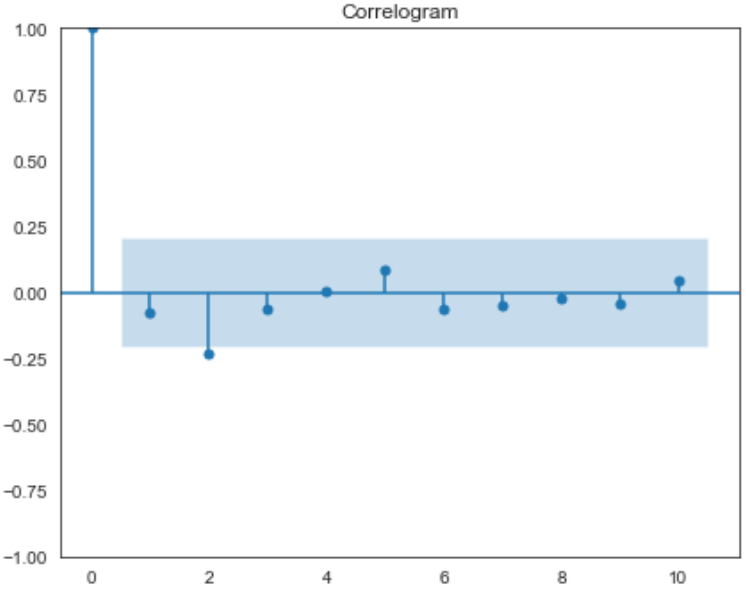
In [76]:
```
ari_1.plot_diagnostics(figsize=(15,12))
plt.show()
```

executed in 394ms, finished 10:25:22 2022-09-29









Observations:

1) still no obvious pattern to residuals

2) predictions are normally distributed

3) the data is further off the red line compared to the random walk model

4) there is less correlation in the residuals so we know the model is improving to pick up on more of the correlation.

In [77]:

```
ari_1_preds = ari_1.predict(typ = 'levels')
# getting the predictions from the ARI model

ari_1_preds_last_six = ari_1_preds[-6::]
# seperating out the predictions from last six months

get_mae(train, ari_1_preds)
# getting the MAE score
```

executed in 3ms, finished 10:25:22 2022-09-29

```
0.101727424626525
```

On average, the random walk model's forecasts distance from the true value is 10%. This is much higher than our naive model!

In [78]:

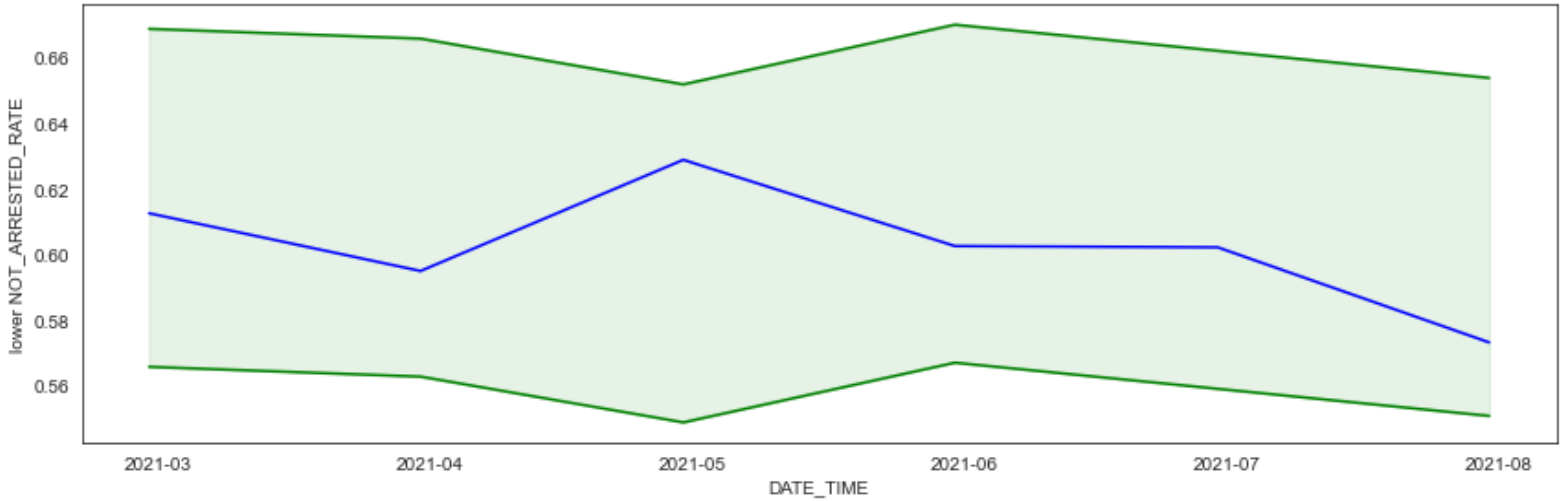
```
forecast_last_six = ari_1.get_prediction(start = -6)
# getting the predictions from last six months

ari_1_conf_int_last_six = forecast_last_six.conf_int(alpha=0.05)
# getting the confidence interval from those predictions

ari_1_conf_int_last_six.head()
# taking a look at the range
```

executed in 6ms, finished 10:25:22 2022-09-29

DATE_TIME	lower NOT_ARRESTED_RATE	upper NOT_ARRESTED_RATE
2021-02-28	0.565789	0.668763
2021-03-31	0.562810	0.665784
2021-04-30	0.548862	0.651837
2021-05-31	0.567049	0.670024
2021-06-30	0.559094	0.662068

In [79]:
```
1 last_six_visualize_preds(ari_1_conf_int_last_six, train[-6::])
```

executed in 170ms, finished 10:25:22 2022-09-29

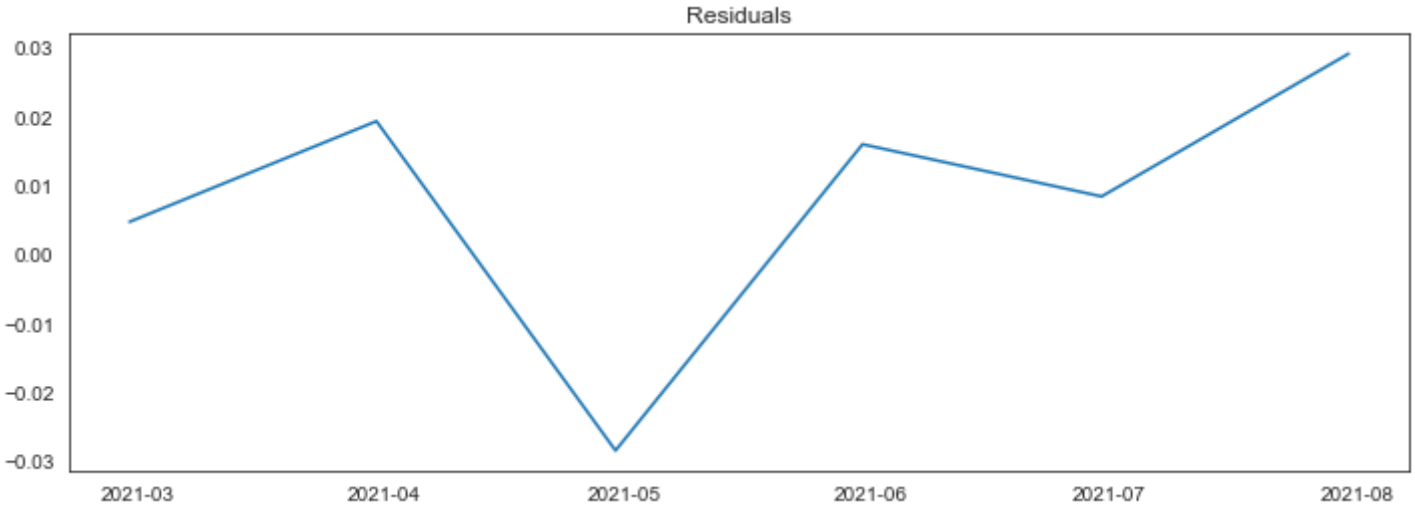


Observations: The data is sitting within the confidence interval, that's the good news. The bad news is it's still predicting a dip during may 2021 when there was a slight spike. This conf interval ranges from 56% to 66% which is again, where most of the data lies, so not precise enough to be useful yet.

In [80]:
```
visualize_diagnostics(ari_1_preds_last_six, train[-6::])
```

executed in 121ms, finished 10:25:22 2022-09-29

```
STANDARD DEVIATION OF RESIDUALS: 0.019894764905982285
```



This plot is again showing the struggle in May of 2021 and August 2021. We have brought down the std of the residuals, meaning the residuals are more standardized.

13 IMA Model

In [81]:

```
ima_1 = ARIMA(train, order = (0, 1, 1)).fit()
# the ima model is a model with a first difference, and 1 moving average term

ima_1.summary()
# showing the summary
```

executed in 40ms, finished 10:25:22 2022-09-29

SARIMAX Results

Dep. Variable:	NOT_ARRESTED_RATE	No. Observations:	91
Model:	ARIMA(0, 1, 1)	Log Likelihood	201.945
Date:	Thu, 29 Sep 2022	AIC	-399.889
Time:	10:25:22	BIC	-394.889
Sample:	01-31-2014	HQIC	-397.873
	- 07-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.4407	0.089	-4.935	0.000	-0.616	-0.266
sigma2	0.0007	8.11e-05	8.097	0.000	0.000	0.001

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	3.54
Prob(Q):	0.96	Prob(JB):	0.17
Heteroskedasticity (H):	2.94	Skew:	-0.17
Prob(H) (two-sided):	0.00	Kurtosis:	3.91

Warnings:

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

In [82]:

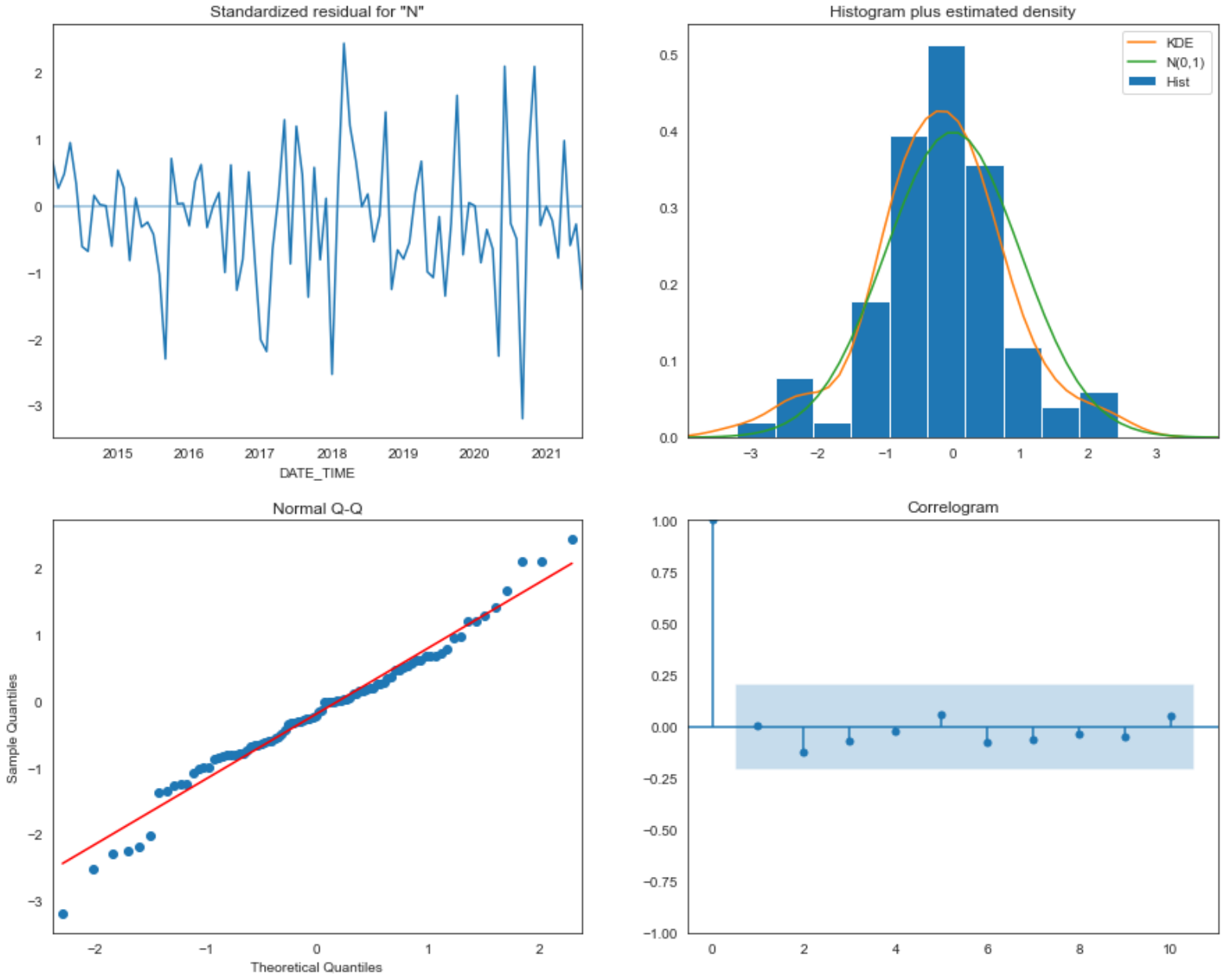
```
ima_1.aic

# ok we're still moving in the right direction as the number is still getting lower
```

executed in 3ms, finished 10:25:22 2022-09-29

```
-399.88900746544243
```

In [83]:
```
ima_1.plot_diagnostics(figsize=(15,12))
plt.show()
```

executed in 394ms, finished 10:25:23 2022-09-29



Observations:

1) We can finally start to see some pattern or seasonality that the model is failing to reproduce on the residuals.

2) This data is more normally distributed.

3) There are increasing amounts of outliers.

4) The correlations found in the residuals is not statistically significant so that's good news.

In [84]:

```
ima_1_preds = ima_1.predict(typ = 'levels')
# getting the predictions from the IMA model

ima_1_preds_last_six = ima_1_preds[-6::]
# separating out last six months predictions

get_mae(train, ima_1_preds)
# taking a look at the MAE score.
```

executed in 3ms, finished 10:25:23 2022-09-29

```
0.10119363855619312
```

On average, the random walk model's forecasts distance from the true value is 10%. This is much higher than our naive model!

In [85]:

```
forecast_last_six = ima_1.get_prediction(start = -6)
# the predictions from last six months

ima_1_conf_int_last_six = forecast_last_six.conf_int(alpha=0.05)
# getting the confidence interval

ima_1_conf_int_last_six.head()
# taking a look at the range
```

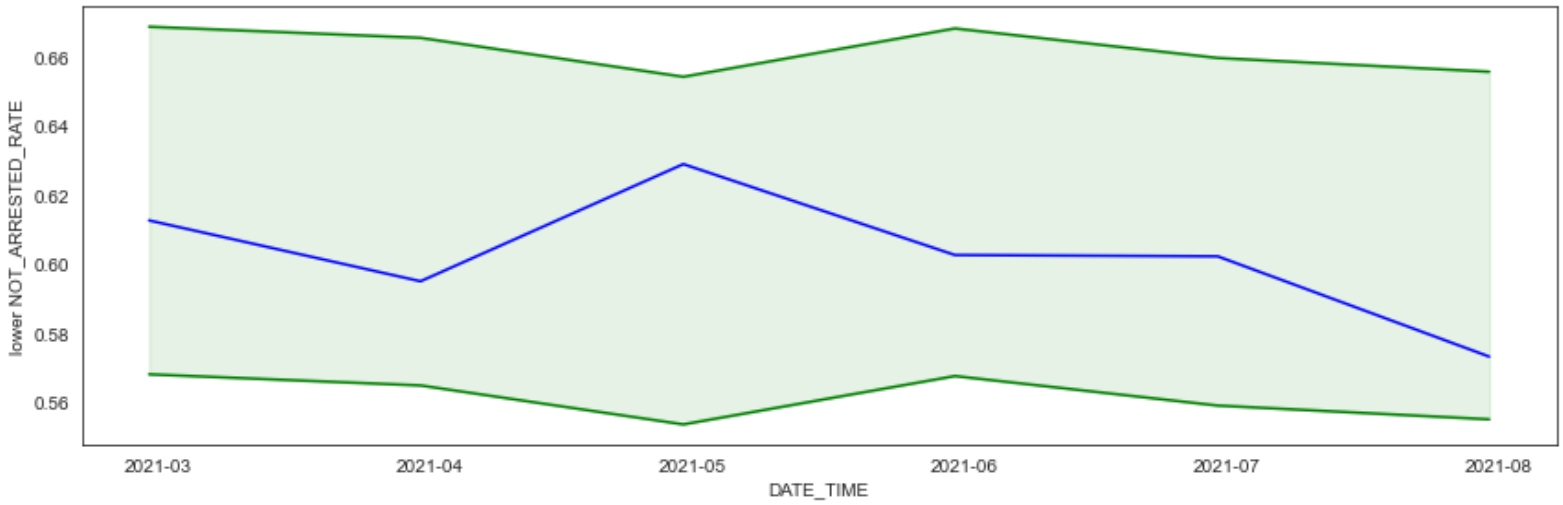
executed in 6ms, finished 10:25:23 2022-09-29

DATE_TIME	lower NOT_ARRESTED_RATE	upper NOT_ARRESTED_RATE
2021-02-28	0.568110	0.668561
2021-03-31	0.564892	0.665343
2021-04-30	0.553636	0.654087
2021-05-31	0.567610	0.668061
2021-06-30	0.559079	0.659530

In [86]:

```
last_six_visualize_preds(ima_1_conf_int_last_six, train[-6::])
# looking to see if our confidence interval is better representing the data
```

executed in 169ms, finished 10:25:23 2022-09-29

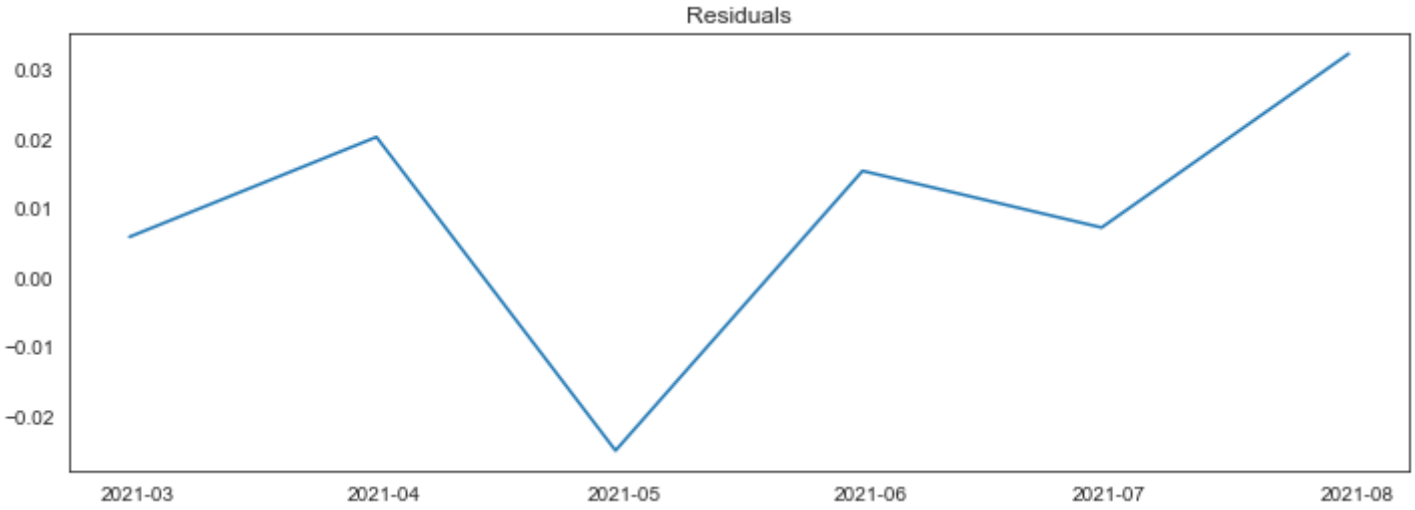


This model doesn't appear to offer much improvement visually over the IMA and Random walk models, it's struggling to be precise enough to be used for it's original intention of predicting spikes.

In [87]:
```
visualize_diagnostics(ima_1_preds_last_six, train[-6::])
```

executed in 121ms, finished 10:25:23 2022-09-29

```
STANDARD DEVIATION OF RESIDUALS: 0.01931695571630262
```



This model is struggling slightly less during may of last year, and more during august of last year. We've brought down the std of the residuals again slightly, making the residuals more uniform.

14 SARIMA Model with Auto Arima

AUTO ARIMA:

I'll be using Auto Arima is like a grid search for ARIMA terms and I'm going to iterate over auto regressive terms (p), difference terms (d) and moving average terms (q). I will specifically not cap on our pre-established maximums, because I want to visually inspect the AIC for terms above the maximum and confirm that those scores are higher than the expected order of terms we think auto arima will give us.

EXPECTED ORDER (1, 1, 1) (?, ?, ?,)[7]

I'm setting 'm' to 7 because there are 7 years that this data represents and we are working with a yearly cycle.

In [88]:

```
sarima_model_params = pm.arima.auto_arima(train, error_action = 'warn', m = 7,
                                          start_p= 0, start_d = 0, start_q = 0,
                                          start_P = 0, start_D = 0, start_Q = 0,
                                          seasonal= True, trace = True,
                                          supress_warning = True,
                                          stepwise = False,
                                          random_state = 1)
```

executed in 14.9s, finished 10:25:38 2022-09-29

```
ARIMA(0,1,0)(0,0,0)[7] intercept   : AIC=-387.907, Time=0.02 sec
ARIMA(0,1,0)(0,0,1)[7] intercept   : AIC=-385.956, Time=0.04 sec
ARIMA(0,1,0)(0,0,2)[7] intercept   : AIC=-384.117, Time=0.14 sec
ARIMA(0,1,0)(1,0,0)[7] intercept   : AIC=-385.952, Time=0.08 sec
ARIMA(0,1,0)(1,0,1)[7] intercept   : AIC=-383.971, Time=0.04 sec
ARIMA(0,1,0)(1,0,2)[7] intercept   : AIC=-382.146, Time=0.20 sec
ARIMA(0,1,0)(2,0,0)[7] intercept   : AIC=-384.178, Time=0.07 sec
ARIMA(0,1,0)(2,0,1)[7] intercept   : AIC=-382.197, Time=0.18 sec
ARIMA(0,1,0)(2,0,2)[7] intercept   : AIC=-380.595, Time=0.31 sec
ARIMA(0,1,1)(0,0,0)[7] intercept   : AIC=-401.919, Time=0.03 sec
ARIMA(0,1,1)(0,0,1)[7] intercept   : AIC=-400.499, Time=0.06 sec
ARIMA(0,1,1)(0,0,2)[7] intercept   : AIC=-399.132, Time=0.15 sec
ARIMA(0,1,1)(1,0,0)[7] intercept   : AIC=-400.394, Time=0.09 sec
ARIMA(0,1,1)(1,0,1)[7] intercept   : AIC=-398.787, Time=0.15 sec
ARIMA(0,1,1)(1,0,2)[7] intercept   : AIC=-397.097, Time=0.21 sec
ARIMA(0,1,1)(2,0,0)[7] intercept   : AIC=-399.280, Time=0.22 sec
ARIMA(0,1,1)(2,0,1)[7] intercept   : AIC=-397.316, Time=0.19 sec
ARIMA(0,1,1)(2,0,2)[7] intercept   : AIC=-395.363, Time=0.36 sec
ARIMA(0,1,2)(0,0,0)[7] intercept   : AIC=-402.056, Time=0.02 sec
ARIMA(0,1,2)(0,0,1)[7] intercept   : AIC=-400.438, Time=0.08 sec
ARIMA(0,1,2)(0,0,2)[7] intercept   : AIC=-398.894, Time=0.19 sec
ARIMA(0,1,2)(1,0,0)[7] intercept   : AIC=-400.378, Time=0.05 sec
ARIMA(0,1,2)(1,0,1)[7] intercept   : AIC=-398.492, Time=0.09 sec
ARIMA(0,1,2)(1,0,2)[7] intercept   : AIC=-396.866, Time=0.27 sec
ARIMA(0,1,2)(2,0,0)[7] intercept   : AIC=-398.992, Time=0.22 sec
ARIMA(0,1,2)(2,0,1)[7] intercept   : AIC=-397.028, Time=0.29 sec
ARIMA(0,1,3)(0,0,0)[7] intercept   : AIC=-400.269, Time=0.02 sec
ARIMA(0,1,3)(0,0,1)[7] intercept   : AIC=-398.576, Time=0.22 sec
ARIMA(0,1,3)(0,0,2)[7] intercept   : AIC=-397.189, Time=0.22 sec
ARIMA(0,1,3)(1,0,0)[7] intercept   : AIC=-398.524, Time=0.10 sec
```

```
ARIMA(0,1,3)(1,0,1)[7] intercept   : AIC=-396.628, Time=0.10 sec
ARIMA(0,1,3)(2,0,0)[7] intercept   : AIC=-397.339, Time=0.33 sec
ARIMA(0,1,4)(0,0,0)[7] intercept   : AIC=-398.453, Time=0.09 sec
ARIMA(0,1,4)(0,0,1)[7] intercept   : AIC=-396.685, Time=0.16 sec
ARIMA(0,1,4)(1,0,0)[7] intercept   : AIC=-396.650, Time=0.10 sec
ARIMA(0,1,5)(0,0,0)[7] intercept   : AIC=-396.449, Time=0.11 sec
ARIMA(1,1,0)(0,0,0)[7] intercept   : AIC=-395.319, Time=0.02 sec
ARIMA(1,1,0)(0,0,1)[7] intercept   : AIC=-393.594, Time=0.06 sec
ARIMA(1,1,0)(0,0,2)[7] intercept   : AIC=-392.271, Time=0.10 sec
ARIMA(1,1,0)(1,0,0)[7] intercept   : AIC=-393.544, Time=0.05 sec
ARIMA(1,1,0)(1,0,1)[7] intercept   : AIC=-391.790, Time=0.13 sec
ARIMA(1,1,0)(1,0,2)[7] intercept   : AIC=-390.302, Time=0.24 sec
ARIMA(1,1,0)(2,0,0)[7] intercept   : AIC=-392.511, Time=0.16 sec
ARIMA(1,1,0)(2,0,1)[7] intercept   : AIC=-390.511, Time=0.36 sec
ARIMA(1,1,0)(2,0,2)[7] intercept   : AIC=-388.729, Time=0.32 sec
ARIMA(1,1,1)(0,0,0)[7] intercept   : AIC=-402.466, Time=0.09 sec
ARIMA(1,1,1)(0,0,1)[7] intercept   : AIC=-400.681, Time=0.08 sec
ARIMA(1,1,1)(0,0,2)[7] intercept   : AIC=-399.291, Time=0.20 sec
ARIMA(1,1,1)(1,0,0)[7] intercept   : AIC=-400.591, Time=0.09 sec
ARIMA(1,1,1)(1,0,1)[7] intercept   : AIC=-398.743, Time=0.20 sec
ARIMA(1,1,1)(1,0,2)[7] intercept   : AIC=-397.262, Time=0.31 sec
ARIMA(1,1,1)(2,0,0)[7] intercept   : AIC=-399.407, Time=0.35 sec
ARIMA(1,1,1)(2,0,1)[7] intercept   : AIC=-397.357, Time=0.15 sec
ARIMA(1,1,2)(0,0,0)[7] intercept   : AIC=-400.241, Time=0.04 sec
ARIMA(1,1,2)(0,0,1)[7] intercept   : AIC=-398.584, Time=0.15 sec
ARIMA(1,1,2)(0,0,2)[7] intercept   : AIC=-397.121, Time=0.23 sec
ARIMA(1,1,2)(1,0,0)[7] intercept   : AIC=-398.533, Time=0.12 sec
ARIMA(1,1,2)(1,0,1)[7] intercept   : AIC=-396.643, Time=0.12 sec
ARIMA(1,1,2)(2,0,0)[7] intercept   : AIC=-397.233, Time=0.30 sec
ARIMA(1,1,3)(0,0,0)[7] intercept   : AIC=-398.069, Time=0.04 sec
ARIMA(1,1,3)(0,0,1)[7] intercept   : AIC=-396.438, Time=0.11 sec
ARIMA(1,1,3)(1,0,0)[7] intercept   : AIC=-396.383, Time=0.15 sec
ARIMA(1,1,4)(0,0,0)[7] intercept   : AIC=-396.532, Time=0.12 sec
ARIMA(2,1,0)(0,0,0)[7] intercept   : AIC=-397.941, Time=0.03 sec
ARIMA(2,1,0)(0,0,1)[7] intercept   : AIC=-396.274, Time=0.11 sec
ARIMA(2,1,0)(0,0,2)[7] intercept   : AIC=-394.561, Time=0.14 sec
ARIMA(2,1,0)(1,0,0)[7] intercept   : AIC=-396.235, Time=0.09 sec
ARIMA(2,1,0)(1,0,1)[7] intercept   : AIC=-394.389, Time=0.23 sec
ARIMA(2,1,0)(1,0,2)[7] intercept   : AIC=-392.536, Time=0.29 sec
ARIMA(2,1,0)(2,0,0)[7] intercept   : AIC=-394.651, Time=0.19 sec
ARIMA(2,1,0)(2,0,1)[7] intercept   : AIC=-392.656, Time=0.40 sec
ARIMA(2,1,1)(0,0,0)[7] intercept   : AIC=-400.431, Time=0.09 sec
ARIMA(2,1,1)(0,0,1)[7] intercept   : AIC=-398.698, Time=0.12 sec
```

```
ARIMA(2,1,1)(0,0,2)[7] intercept   : AIC=-397.325, Time=0.20 sec
ARIMA(2,1,1)(1,0,0)[7] intercept   : AIC=-398.652, Time=0.12 sec
ARIMA(2,1,1)(1,0,1)[7] intercept   : AIC=-396.761, Time=0.22 sec
ARIMA(2,1,1)(2,0,0)[7] intercept   : AIC=-397.666, Time=0.34 sec
ARIMA(2,1,2)(0,0,0)[7] intercept   : AIC=-398.344, Time=0.02 sec
ARIMA(2,1,2)(0,0,1)[7] intercept   : AIC=-396.669, Time=0.10 sec
ARIMA(2,1,2)(1,0,0)[7] intercept   : AIC=-396.622, Time=0.14 sec
ARIMA(2,1,3)(0,0,0)[7] intercept   : AIC=-397.201, Time=0.12 sec
ARIMA(3,1,0)(0,0,0)[7] intercept   : AIC=-398.128, Time=0.05 sec
ARIMA(3,1,0)(0,0,1)[7] intercept   : AIC=-396.412, Time=0.10 sec
ARIMA(3,1,0)(0,0,2)[7] intercept   : AIC=-394.820, Time=0.21 sec
ARIMA(3,1,0)(1,0,0)[7] intercept   : AIC=-396.371, Time=0.15 sec
ARIMA(3,1,0)(1,0,1)[7] intercept   : AIC=-394.556, Time=0.25 sec
ARIMA(3,1,0)(2,0,0)[7] intercept   : AIC=-394.935, Time=0.39 sec
ARIMA(3,1,1)(0,0,0)[7] intercept   : AIC=-401.880, Time=0.12 sec
ARIMA(3,1,1)(0,0,1)[7] intercept   : AIC=-398.456, Time=0.19 sec
ARIMA(3,1,1)(1,0,0)[7] intercept   : AIC=-396.946, Time=0.21 sec
ARIMA(3,1,2)(0,0,0)[7] intercept   : AIC=-395.603, Time=0.13 sec
ARIMA(4,1,0)(0,0,0)[7] intercept   : AIC=-397.752, Time=0.08 sec
ARIMA(4,1,0)(0,0,1)[7] intercept   : AIC=-396.226, Time=0.24 sec
ARIMA(4,1,0)(1,0,0)[7] intercept   : AIC=-396.156, Time=0.13 sec
ARIMA(4,1,1)(0,0,0)[7] intercept   : AIC=-395.656, Time=0.08 sec
ARIMA(5,1,0)(0,0,0)[7] intercept   : AIC=-395.780, Time=0.07 sec

Best model:  ARIMA(1,1,1)(0,0,0)[7] intercept
Total fit time: 14.880 seconds
```

Auto Arima agreed with our interpretation of the ACF and PACF plots, and our assumptions that the most correct order of terms would be (1, 1, 1)

In [89]:

```
# auto arima agrees with our ACF and PACF plots after taking a first differnce,
#1 AR term, 1 difference and 1 MA terms

sarima_1 = ARIMA(train, order = (1,1,1), seasonal_order=(0, 0, 0, 7)).fit()
sarima_1.summary()
```

executed in 81ms, finished 10:25:38 2022-09-29

```
/Users/b0ihazard/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/base/model.py:604: ConvergenceWar
ning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

SARIMAX Results

Dep. Variable:	NOT_ARRESTED_RATE	**No. Observations:**	91
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	202.262
Date:	Thu, 29 Sep 2022	**AIC**	-398.524
Time:	10:25:38	**BIC**	-391.025
Sample:	01-31-2014	**HQIC**	-395.500
	- 07-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.1665	0.242	0.687	0.492	-0.309	0.642
ma.L1	-0.5686	0.191	-2.980	0.003	-0.943	-0.195
sigma2	0.0007	8.37e-05	7.792	0.000	0.000	0.001

Ljung-Box (L1) (Q):	0.13	**Jarque-Bera (JB):**	2.95
Prob(Q):	0.72	**Prob(JB):**	0.23
Heteroskedasticity (H):	2.93	**Skew:**	-0.22
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.76

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

In [90]:
```
1 random_walk_model.aic
```

executed in 2ms, finished 10:25:38 2022-09-29

```
-388.97482570078154
```

In [91]:
```
1 ari_1.aic
```

executed in 3ms, finished 10:25:38 2022-09-29

```
-395.5732590361313
```

In [92]:
```
1 ima_1.aic
```

executed in 3ms, finished 10:25:38 2022-09-29

```
-399.88900746544243
```

In [93]:
```
1 sarima_1.aic
```

executed in 2ms, finished 10:25:38 2022-09-29

```
-398.5243854989987
```

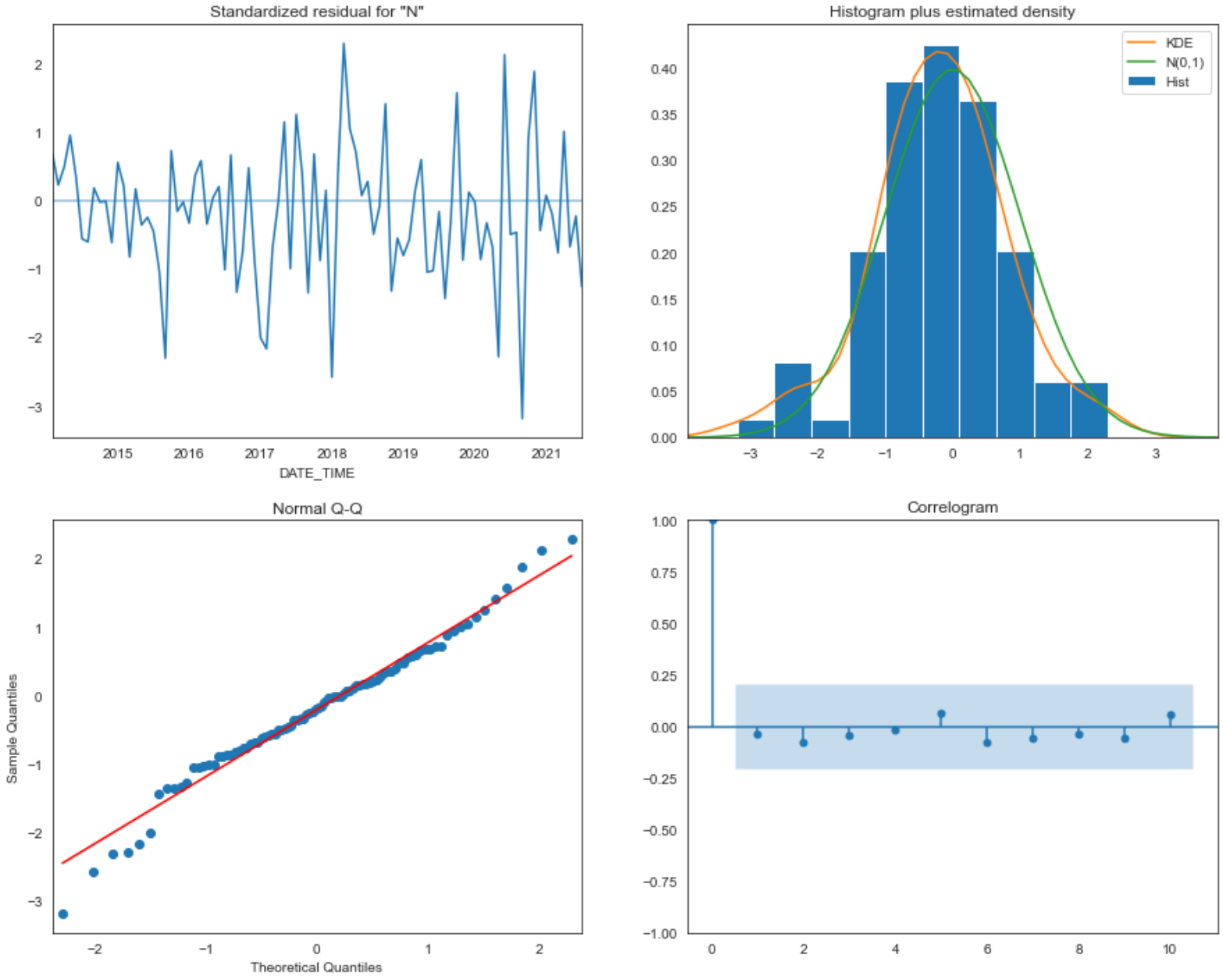
In [ ]:
```
1
```

The random walk was actually the best fit for the data

In [94]:

```
sarima_1.plot_diagnostics(figsize=(15,12))
plt.show()
```

executed in 406ms, finished 10:25:39 2022-09-29



Observations:

1) there is still some pattern that I can clearly see repeating once a year on 2015, 2016 and 2017. 2) this is the most normally distributed the predictions have come. 3) the outliers are becoming more of a problem.

In [95]:

```
sarima_1_preds = sarima_1.predict(typ = 'levels')
# getting predictions from the model

sarima_1_preds_last_six = sarima_1_preds[-6::]
# getting last six months predictions

get_mae(train, sarima_1_preds)
# getting the MAE Score
```

executed in 4ms, finished 10:25:39 2022-09-29

```
0.10108887249771228
```

14.0.1 SARIMA MAE SCORE INTERPRETATION

On average, the SARIMA's forecasts distance from the true value is 10%. This is much higher than our naive model! This percentage has not gone down for any of our models except our baseline naive model, despite our AIC number going down slightly with IMA and SARIMA models compared to the ARI model. What this tells us is that it's not viable to try and model the data for predicting spikes, this model's intended use.

In [96]:

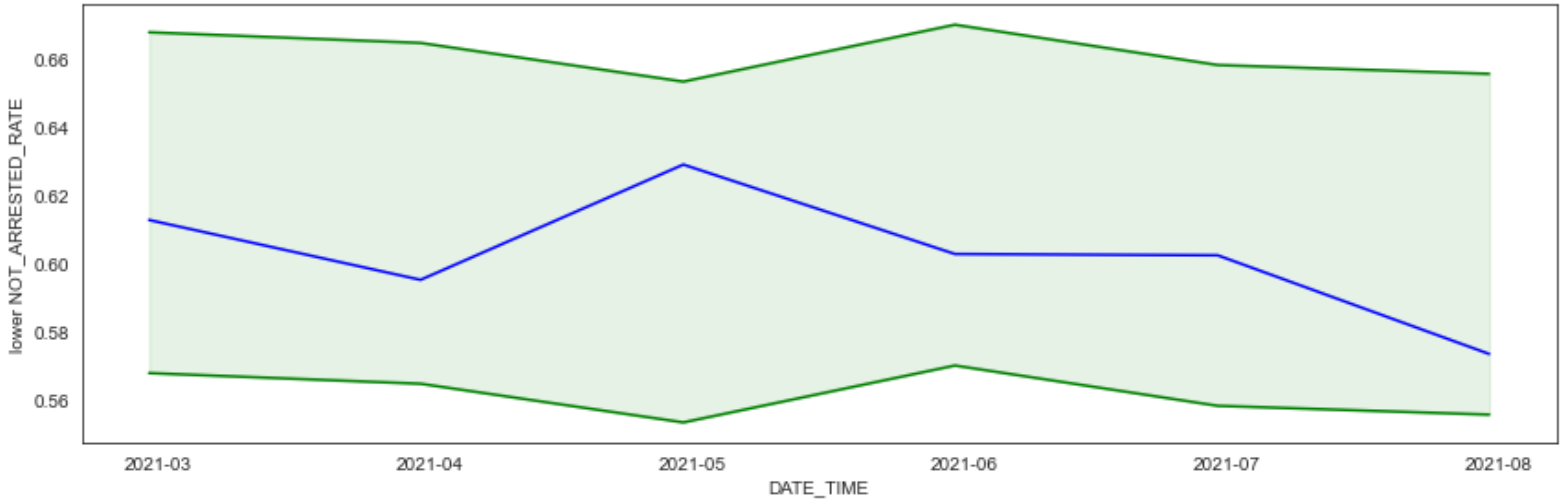
```
forecast_last_six = sarima_1.get_prediction(start = -6)
# getting predictions from the model

sarima_1_conf_int_last_six = forecast_last_six.conf_int(alpha=0.05)
# getting the confidence interval from those predictions

sarima_1_conf_int_last_six.head()
# taking a look
```

executed in 6ms, finished 10:25:39 2022-09-29

DATE_TIME	lower NOT_ARRESTED_RATE	upper NOT_ARRESTED_RATE
2021-02-28	0.567611	0.667706
2021-03-31	0.564485	0.664579
2021-04-30	0.553127	0.653222
2021-05-31	0.569838	0.669933
2021-06-30	0.558001	0.658096

In [97]:
```
1 last_six_visualize_preds(sarima_1_conf_int_last_six, train[-6::])
```

executed in 167ms, finished 10:25:39 2022-09-29

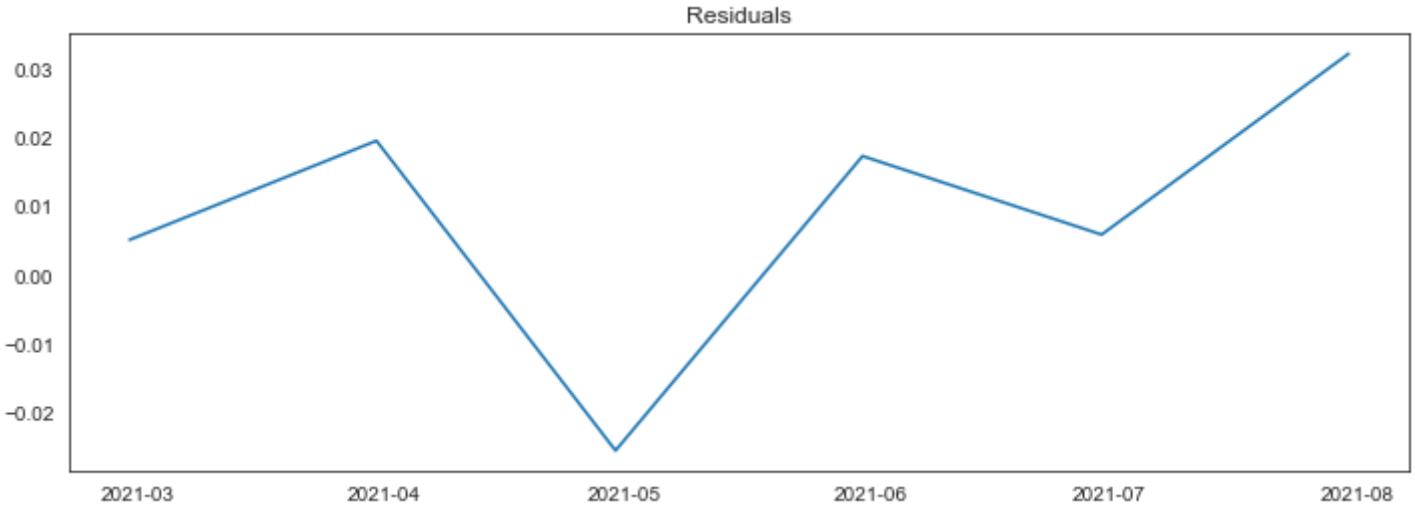


Observations: This model's cofidance interval, while being again similar to the pattern we are seeing with the original data, isn't close enough to the original data to provide us with something we can use to make predictions.

In [98]:
```
visualize_diagnostics(sarima_1_preds_last_six, train[-6::])
```

executed in 118ms, finished 10:25:39 2022-09-29

```
STANDARD DEVIATION OF RESIDUALS: 0.01973626914841105
```



The Sarima model is struggling with may and august of 2021, same as all our other models, and the std of the residuals is slightly higher than the std of the residuals of the IMA model.

WHY GO WITH THE IMA MODEL FOR VALIDATION:

Our naive model has told us all it can already. The IMA Model had a slightly lower AIC score than the other complex models.

15 Validating IMA model

In [99]:

```
forecast_twenty_one = ima_1.forecast(6)
# the predictions from last six months

conf_int_twenty_one = ima_1.get_forecast(6).summary_frame()
# getting the confidence interval

forecast_twenty_one.head()
# taking a look at the range
```

executed in 10ms, finished 10:25:39 2022-09-29

```
2021-08-31    0.587372
2021-09-30    0.587372
2021-10-31    0.587372
2021-11-30    0.587372
2021-12-31    0.587372
Freq: M, Name: predicted_mean, dtype: float64
```

In [100]:

```
get_mae(valid, forecast_twenty_one)

#get_mae(valid, twenty_one_preds.prediction_results)
```

executed in 2ms, finished 10:25:39 2022-09-29

```
0.04541069714156276
```

In [101]:
```
1 conf_int_twenty_one
```

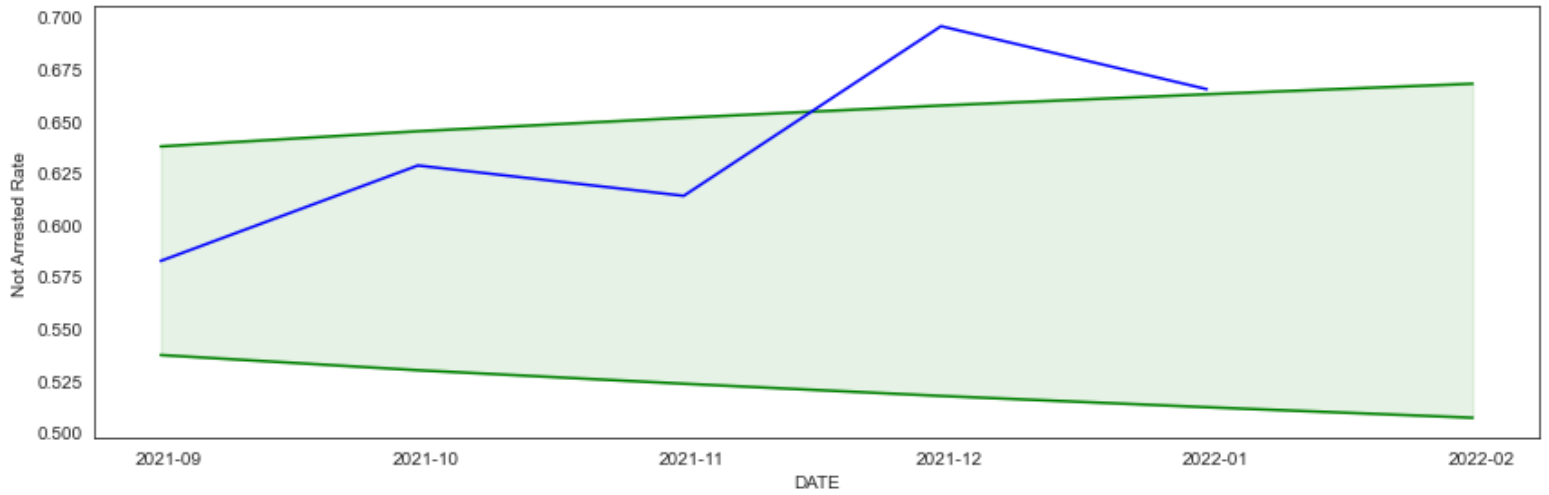
executed in 6ms, finished 10:25:39 2022-09-29

NOT_ARRESTED_RATE	mean	mean_se	mean_ci_lower	mean_ci_upper
2021-08-31	0.587372	0.025626	0.537147	0.637598
2021-09-30	0.587372	0.029362	0.529824	0.644920
2021-10-31	0.587372	0.032673	0.523334	0.651410
2021-11-30	0.587372	0.035679	0.517443	0.657301
2021-12-31	0.587372	0.038450	0.512011	0.662733
2022-01-31	0.587372	0.041035	0.506946	0.667798

In [102]:

```
fig, ax = plt.subplots()

lower = sns.lineplot(data = conf_int_twenty_one,
                     y = 'mean_ci_lower',
                    x = conf_int_twenty_one.index, color = 'g')

upper = sns.lineplot(data = conf_int_twenty_one,
                     y = 'mean_ci_upper',
                    x = conf_int_twenty_one.index, color = 'g')

sns.lineplot(data = valid[1::], y = "NOT_ARRESTED_RATE",
                     x = valid[1::].index, color = 'blue', ax = ax)

plt.fill_between(conf_int_twenty_one.index,
                         conf_int_twenty_one['mean_ci_lower'],
                         conf_int_twenty_one['mean_ci_upper'],
                        color = 'g', alpha = 0.1)

plt.ylabel("Not Arrested Rate")
plt.xlabel("DATE")
fig.tight_layout()

plt.savefig("IMA_in_action")
```

executed in 231ms, finished 10:25:39 2022-09-29

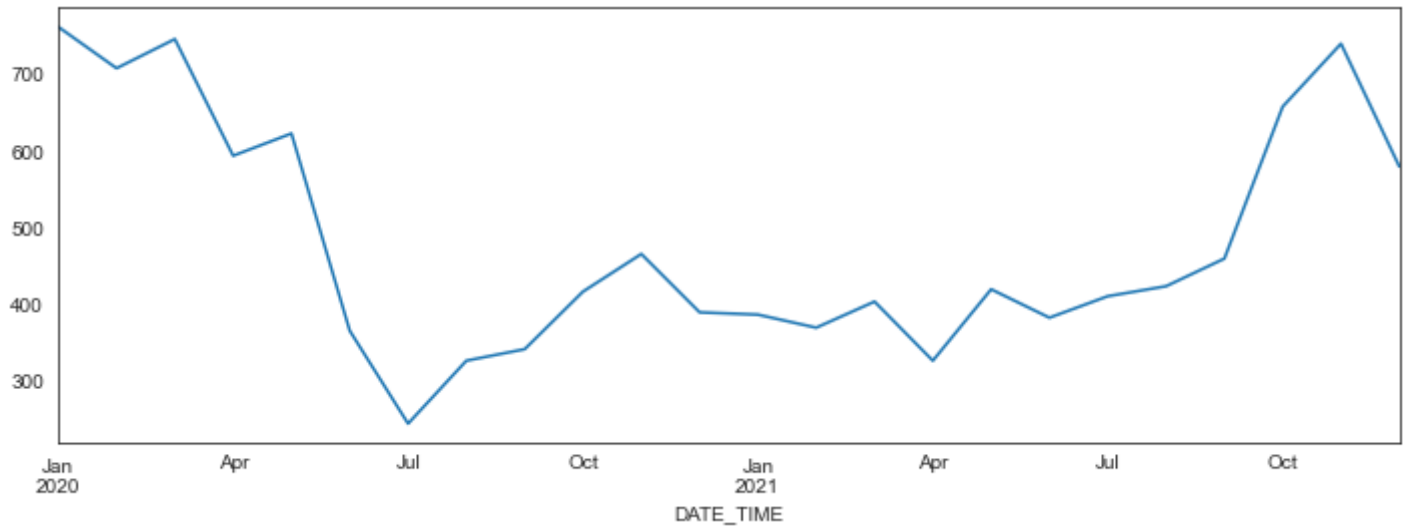


Observations: Our data is outside of our confidence interval! When inspecting where the model is struggling, we can see that december 2021 is a historical high. The slope off the peak of fall 2021 is actually an outlier in this data, and the model is proving it may have some use for anomaly detection. More testing is needed to prove if this model could be used inferentially!

In [103]:
```
1 fourteen_by_month[-24::].plot()
```

executed in 138ms, finished 10:25:39 2022-09-29

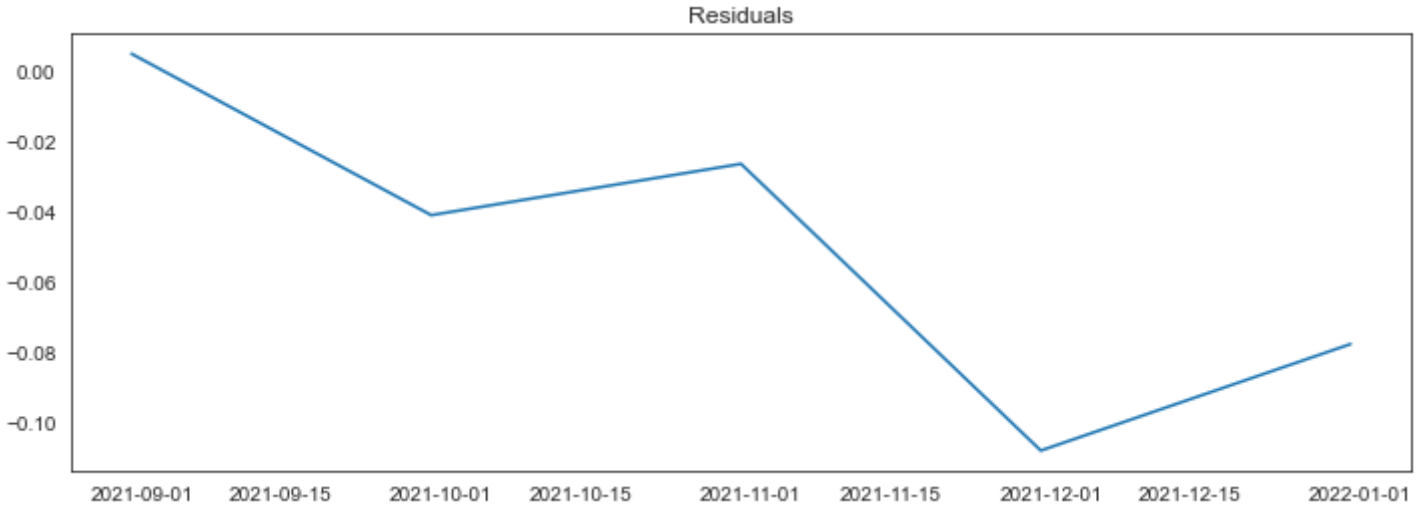
```
<AxesSubplot:xlabel='DATE_TIME'>
```



In [104]:
```
1 visualize_diagnostics(forecast_twenty_one, valid)
```

executed in 158ms, finished 10:25:39 2022-09-29

```
STANDARD DEVIATION OF RESIDUALS: 0.04416883026412912
```



Observations: this model struggled during the most during the spike of december 2021, where our model is off by about 7%. The std of the residuals has also increased by about double on the validation set of data, meaning there's a lot more variance in the residuals for the validation set.

16 Conclusion

Observations: while this model can predict an expected range, it's not precise to be useful as intended. I'd like to research if it could be use for anomaly detection, as we can see above it was able to discern that a spike was above the expected range.

Conclusion: It is not viable to use timeseries modeling alone to try and predict whether or not a stop will not lead to an arrest. Unfortunately for NYPD and the public, this model does not work as intended as is. The utility this model DOES provide the public and NYPD as is, is to anticipate a reasonable range for the rates of stops that do not lead to an arrest based on historical data. Because this is based on historical data, it could potentially be used for anomaly detection to flag if a rates are higher than they have historical precedence for.

Future work: now that using just the date has been ruled out, I'd like to try using a categorical model to use on this data, looking at precinct and suspected crime, once I can clean up those columns enough to be used. I also would like to do research about if the hour is a significant factor in the outcome of the stop and frisk.

Recommendations in the interim, before more research is done:

1) roll out the following visualizations to the public for 3rd party analysis and public understanding of data and policy. 2011 - 2021 https://public.tableau.com/views/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link (https://public.tableau.com/views/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome/2011-2021NYPDStopandFrisksbyRacePrecinctandOutcome?:language=en-US&publish=yes&:display_count=n&:origin=viz_share_link)

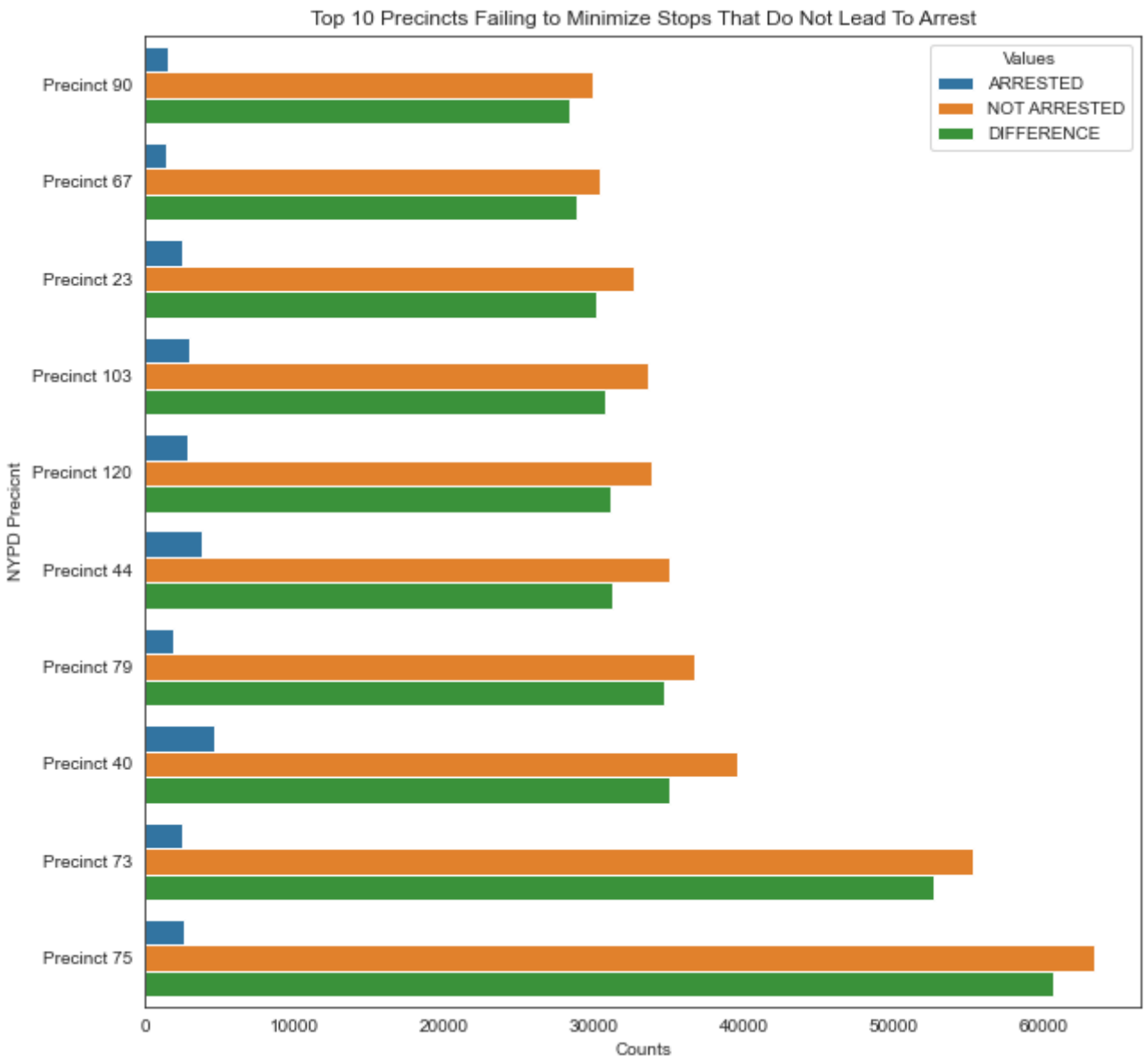
JUST 2021 (the target)
https://public.tableau.com/app/profile/louis.casanave/viz/StopandFrisk2021NYPD/StopandFrisk2021Demograp
(https://public.tableau.com/app/profile/louis.casanave/viz/StopandFrisk2021NYPD/StopandFrisk2021Demograp

2) Let's take another look at precincts where with the greatest difference between stops that have lead to an arrest and stops that have not lead to an arrest and see if we can't do more training in these struggling precincts.

In [105]:

```
plt.figure(figsize=(10, 10))
# resetting the size of the plot

sns.barplot(data = arrs_by_precinct.tail(10).melt(id_vars='index',
                                        value_name='Counts',
                                        var_name='Values'),
            y='index', x='Counts', hue='Values')
# making a plot to show the precicnts where the difference is lowest aka the
# police are doing the best job determening who to stop

plt.title("Top 10 Precincts Failing to Minimize Stops That Do Not Lead To Arrest")
plt.yticks(ticks = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
           labels=['Precinct 90', 'Precinct 67', 'Precinct 23',
                    'Precinct 103', 'Precinct 120', 'Precinct 44', 'Precinct 79',
                  'Precinct 40', 'Precinct 73', 'Precinct 75'])
plt.ylabel("NYPD Precicnt");
# setting labels and titles
```

executed in 162ms, finished 10:25:40 2022-09-29



Working backwards off this chart, it's clear that we should prioritize retraining of police in these precincts the most, starting with Precinct 75.

3) Use IMA Model retroactively for inference to detect when rates are higher than historical precedence.

4) MOST BASIC: Decrease stops if there was a spike in the not-arrested rates from last month.