Hate Crimes in America

Angela Spencer
Ayesha Patel
Clarine Esperance

Data Analysis Questions

Bias

Race

Gender

Location

What patterns do we see in the data?

What patterns in hate crime rates will be seen in relation to political climate?

What regions and states have high hate crime rates?

Does hate crime increase or decrease based on historical events?

Can we forecast how much hate crime will happen as time goes on??

Health

Religion

Politics

Justice

Data Preprocessing

Combined hate crime dataset with the political dataset

- Dropped duplicate columns
- Dropped with more than 70% missing values and replaced NaN values in ‘Offender Race’ column with ‘Unknown’
- Transformed ‘Incident_Date’ column to date time
- Feature engineering to reduce the amount of categories the following columns:
    - Victim Type
    - Bias Description
    - Location Name

Examples of Feature Engineering

```
## TRANSFORMING DATATYPES
#convert to datetime
hate_crime["INCIDENT_DATE"] = pd.to_datetime(hate_crime["INCIDENT_DATE"])

#reduce the number of categories for VICTIM_TYPES by condensing labels
replacements = {'VICTIM_TYPES':{r'.*Law Enforcement Officer.*':'Law Enforcement Officer',
                                r'.*Religious Organization.*': 'Religious Organization',
                                r'.*Business.*': 'Business',
                                r'.*Government.*': 'Government',
                                r'.*Individual.*': 'Individual',
                                r'.*Society/Public.*':'Society/Public'},
               'BIAS_DESC':{r'.*Anti-Black.*':'Anti-Black or African American',
                             r'.*Anti-Jewish.*': 'Anti-Jewish',
                             r'.*Anti-Gay.*': 'Anti-Gay (Male)',
                             r'.*Anti-Lesbian.*': 'Anti-Lesbian (Female)',
                             r'.*Anti-Islamic.*': 'Anti-Islamic (Muslim)',
                             r'.*Anti-Hispanic.*': 'Anti-Hispanic or Latino',
                             r'.*Anti-Transgender.*': 'Anti-Transgender',
                             r'.*Anti-Gender Non-Conforming.*': 'Anti-Gender Non-Conforming',
                             r'.*Anti-Asian.*': 'Anti-Asian',
                             r'.*Anti-Bisexual,*':'Anti-Bisexual',
                             r'.*Anti-American Indian.*': 'Anti-Native American',
                             r'.*Anti-Mental Disability.*': 'Anti-Mental Disability',
                             r'.*Anti-Physical Disability.*': 'Anti-Physical Disability',
                             r'.*Anti-Other Religion.*': 'Anti-Other Religion',
                             r'.*Anti-Multiple Races, Group.*': 'Anti-Multiple Races, Group',
                             r'.*Anti-Hindu.*': 'Anti-Hindu',
                             r'.*Anti-Catholic.*': 'Anti-Catholic',
                             r'.*Anti-Arab.*': 'Anti-Arab',
                             r'.*Anti-Jehovah.*': 'Anti-Jehovahs Witness',
                             r'.*Anti-White.*': 'Anti-White',
                             r'.*Anti-Multiple Religions.*': 'Anti-Multiple Religions',
                             r'.*Anti-Protestant.*': 'Anti-Protestant',
                             r'.*Anti-Native Hawaiian.*': 'Anti-Native Hawaiian or Other Pacific Islander',
                             r'.*Anti-Bisexual.*': 'Anti-Bisexual',
                             r'.*Anti-Female.*': 'Anti-Female',
                             r'.*Anti-Sikh.*': 'Anti-Sikh'},
               'LOCATION_NAME':{r'.*Highway/Road/Alley/Street/Sidewalk.*':'Highway/Road/Alley/Street/Sidewalk',
                                 r'.*College.*': 'School-College/University',
                                 r'.*Residence/Home.*': 'Residence/Home',
                                 r'.*Drug Store/Doctor.*': 'Drug Store/Doctor',
                                 r'.*Commercial/Office Building.*': 'Commercial/Office Building',
                                 r'.*Restaurant.*': 'Restaurant',
                                 r'.*Government/Public Building.*': 'Government/Public Building'
```

Combining the two dataframes

Created new column

'Middle Year' column was created in political dataframe

Merged on 'Start'

All hate crimes that occurred on a year in start All hate crimes that occurred on a year in middle

Concat

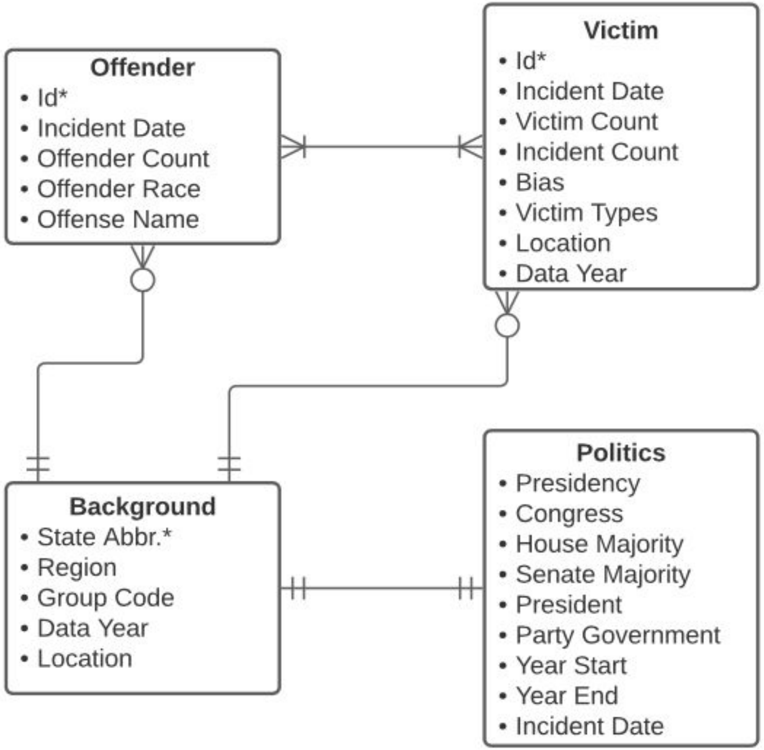
Concat on start and middle column to create new dataframe

Merge on 'Middle'

Merge dataframes for all hate crimes that occured on a year in 'Middle' column

```python
### COMBINE hate_crime and political df


#create a new column to show the middle year between start and end years
political['Middle Year'] = political['Year Start'] + 1

#merge dataframes for all hate crimes that occured on a year in 'Start'
start = hate_crime.merge(political, how='inner', left_on='DATA_YEAR', right_on='Year Start')

#merge dataframes for all hate crimes that occured on a year in 'Middle'
middle = hate_crime.merge(political, how='inner', left_on='DATA_YEAR', right_on='Middle Year')

#concat Start and middle to form new combined dataframe
#no need to include end year because end year for one presidency overlaps with start year of next
hate_crime_combined = pd.concat([start, middle])
hate_crime_combined.drop('Middle Year', axis=1, inplace=True)
```

Lucidchart: 4 tables



PostgreSQL: Dataframes and Tables

Establish Connection to Hate_Crime_Data DB

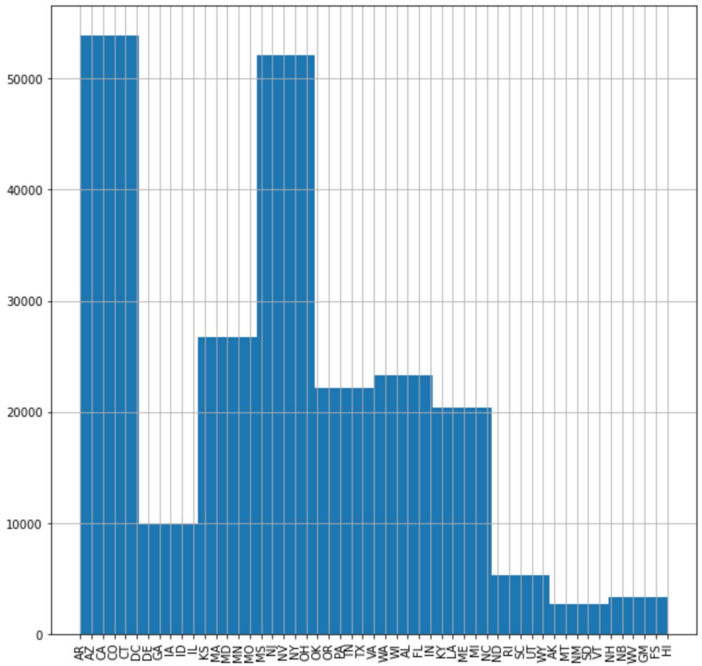
Diving Deeper

Looking at progression of, and relationships between biases, offenders, victim types and incident counts over the span of 30 years as it pertains to major events and shifts in presidency

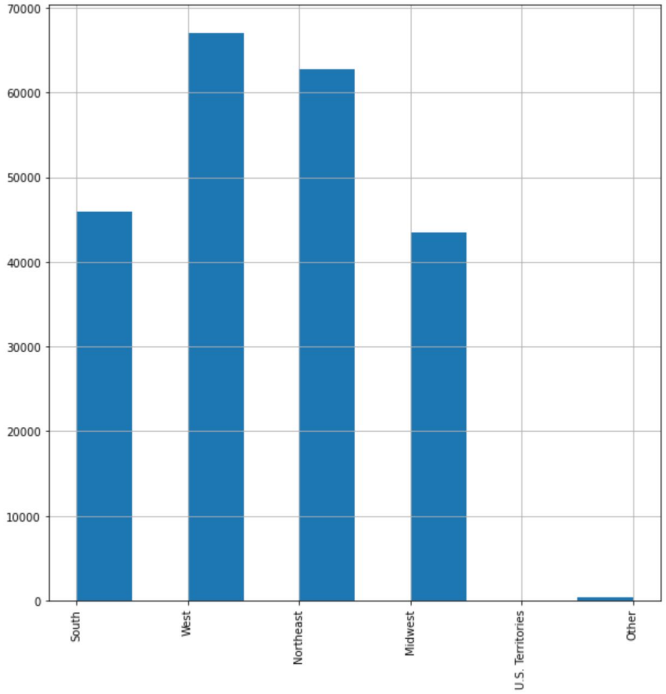
Basic Exploration

1. State and Region Population
2. Victim bias and offender counts
3. States with the highest hate crimes
4. Overall look at the progression of crimes

Data Exploration: Incidents by State and Region

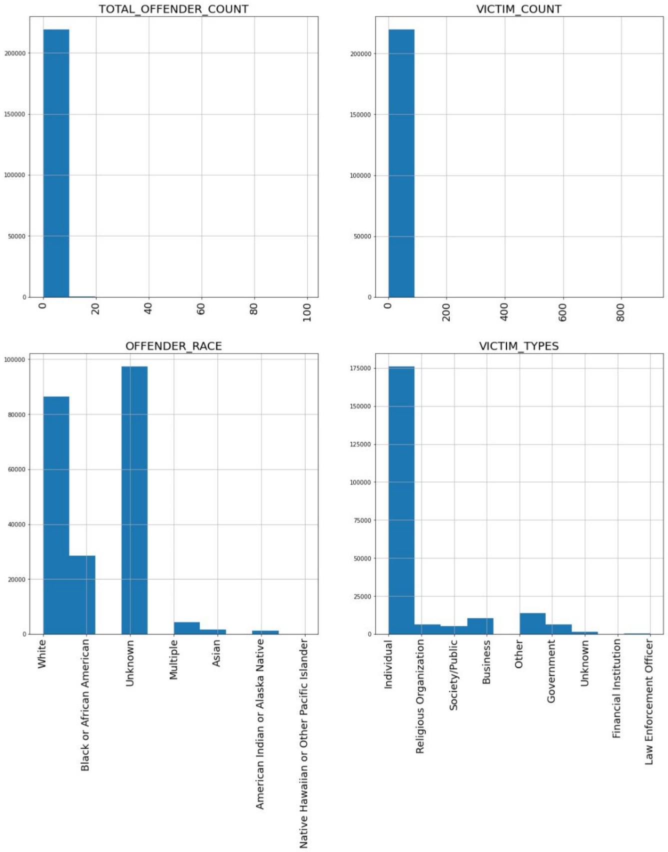


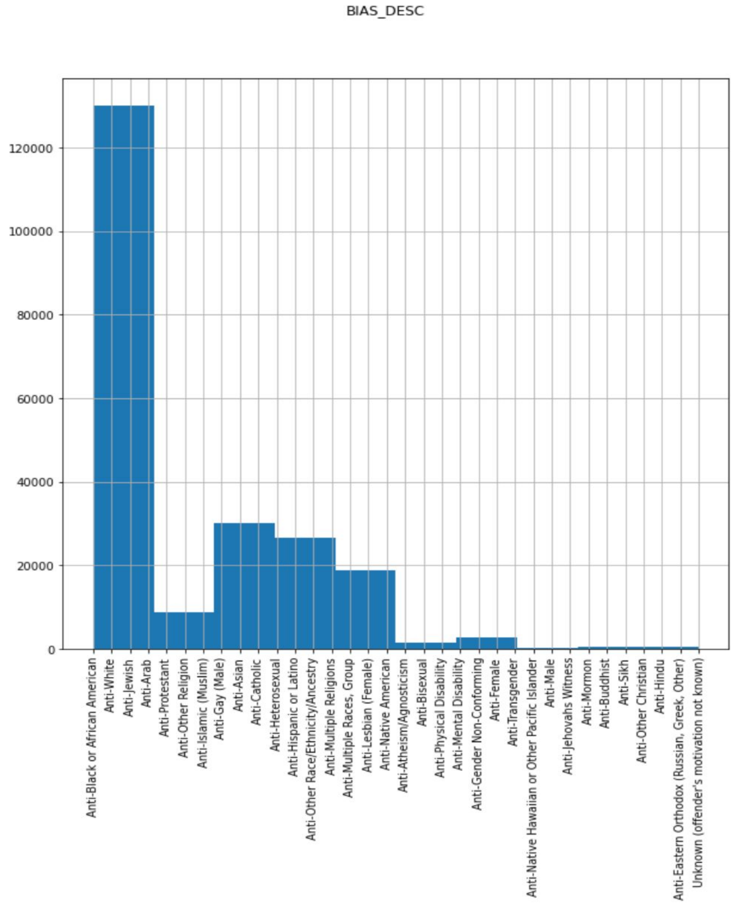
State Population



Region Population

Victims, Offenders, Bias

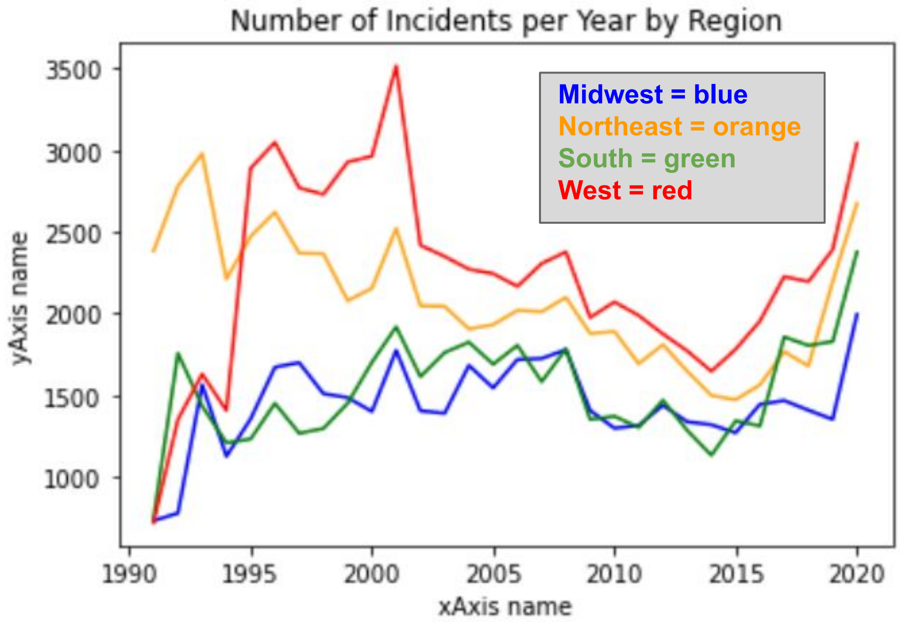


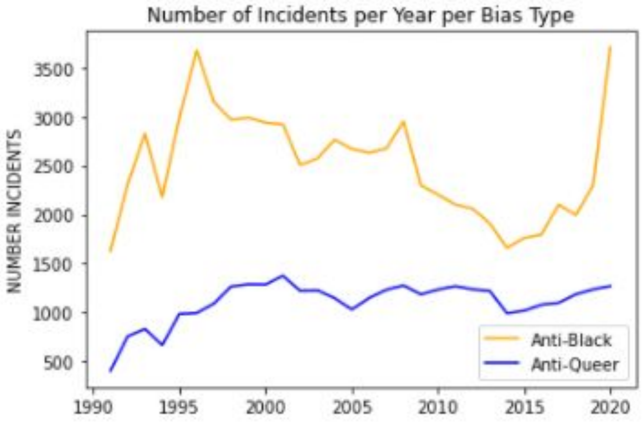


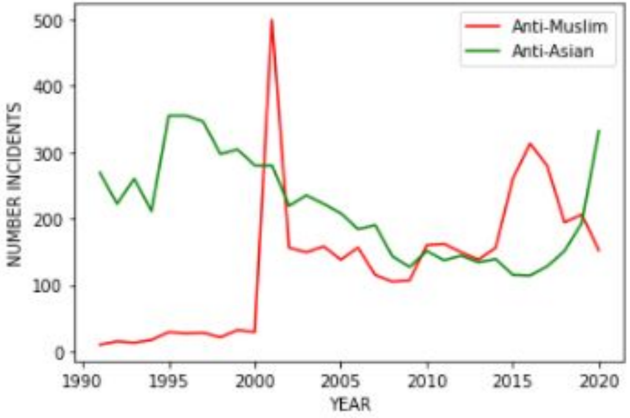
Diving Deeper by Using Queries

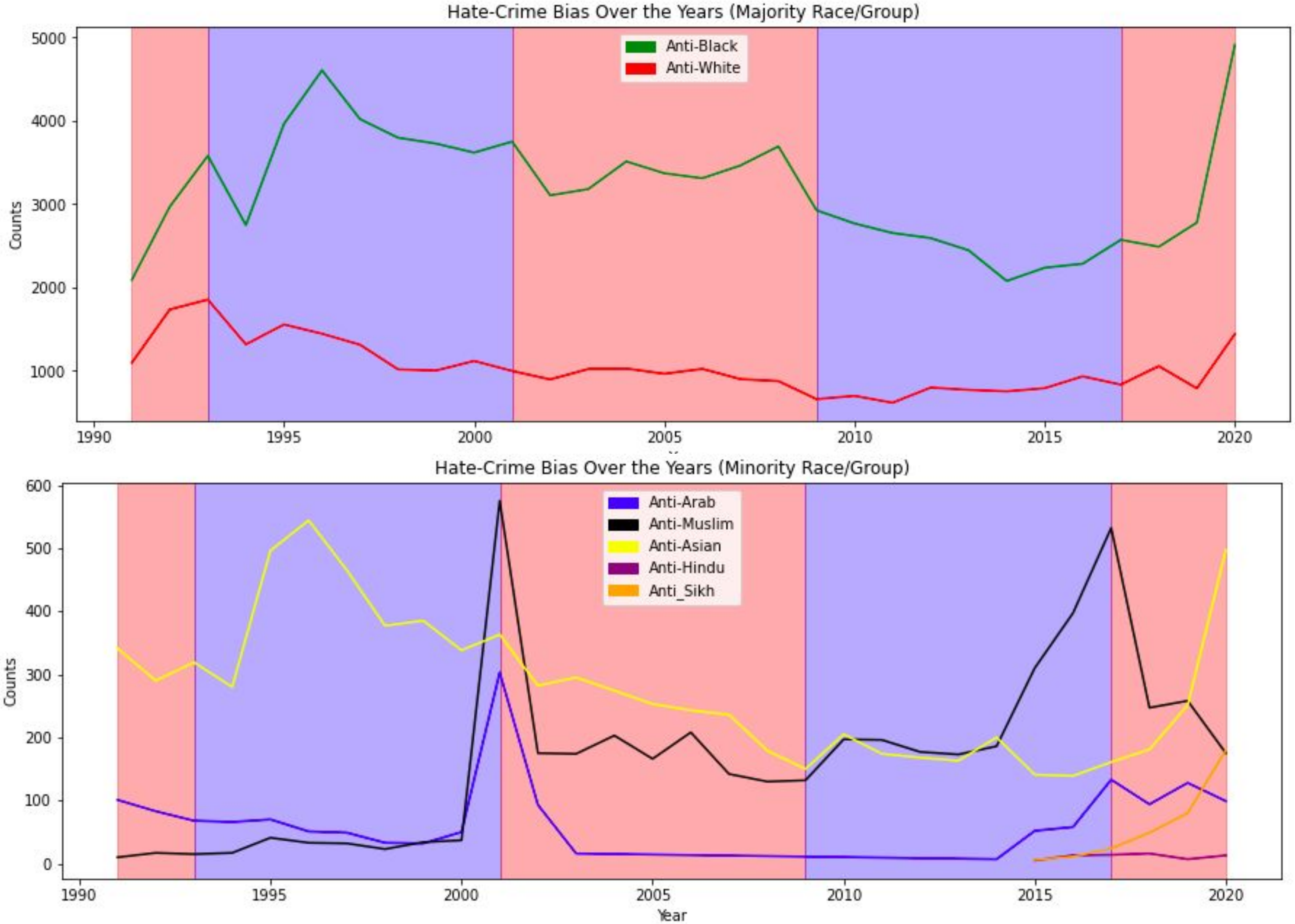
```
DMLquery1 = '''

SELECT datayear, SUM(offendercount) AS yearly_offender_count, SUM(victimcount) AS yearly_victim_count,
                SUM(incidentcount) AS yearly_incident_count
INTO hc_by_year
FROM victim
INNER JOIN offender
ON victim.id = offender.id
GROUP BY datayear;
'''

DMLquery2 = '''

SELECT DISTINCT ON (datayear) datayear, most_common_offense, most_common_vtype,
most_common_bias, most_common_presidency INTO most_common FROM (SELECT datayear,
offensename AS most_common_offense, victimtypes AS most_common_vtype, bias AS most_common_bias,
presidency AS most_common_presidency, COUNT(*) AS _count
FROM offender
INNER JOIN victim
ON offender.id = victim.id
INNER JOIN politics
ON victim.incidentdate = politics.incidentdate
GROUP BY datayear, offensename, victimtypes, bias, presidency)
A ORDER BY datayear, _count DESC;
'''

DMLquery3 = '''

SELECT most_common.datayear, yearly_offender_count, yearly_victim_count, yearly_incident_count,
most_common_offense, most_common_vtype, most_common_bias, most_common_presidency
FROM hc_by_year
INNER JOIN most_common
ON hc_by_year.datayear = most_common.datayear;
'''
```

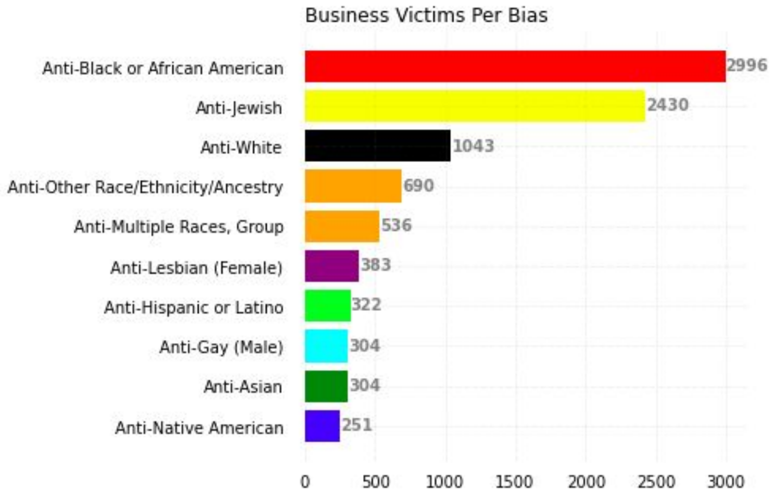
Hate Crime Incidents by Region and Bias Type

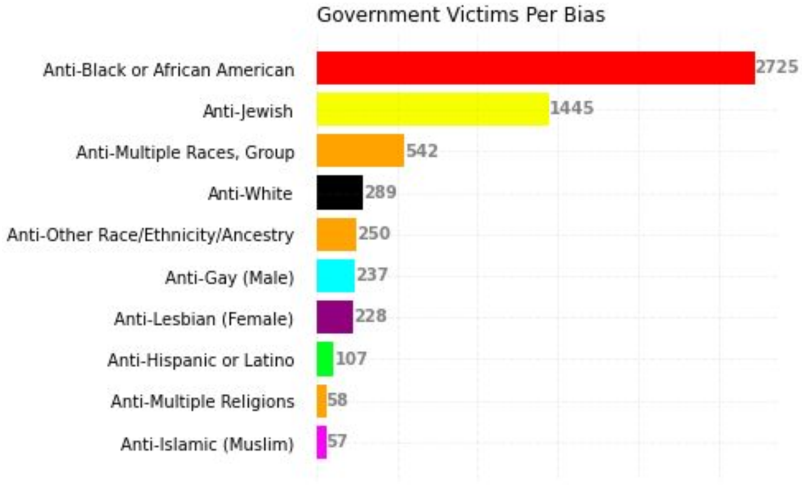


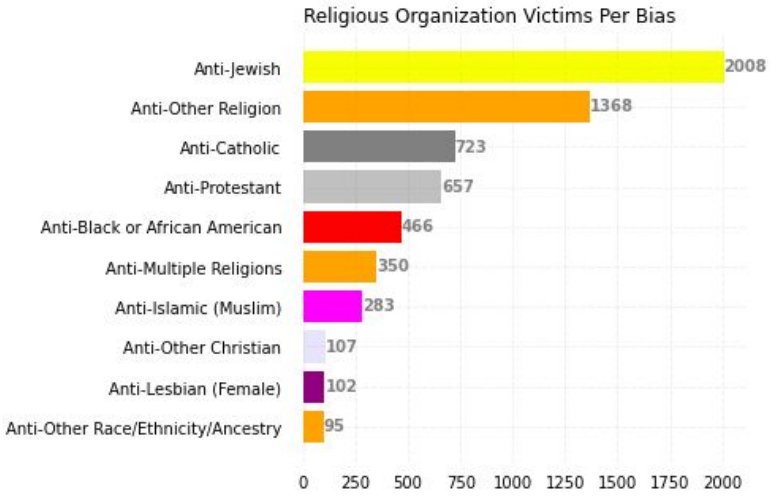


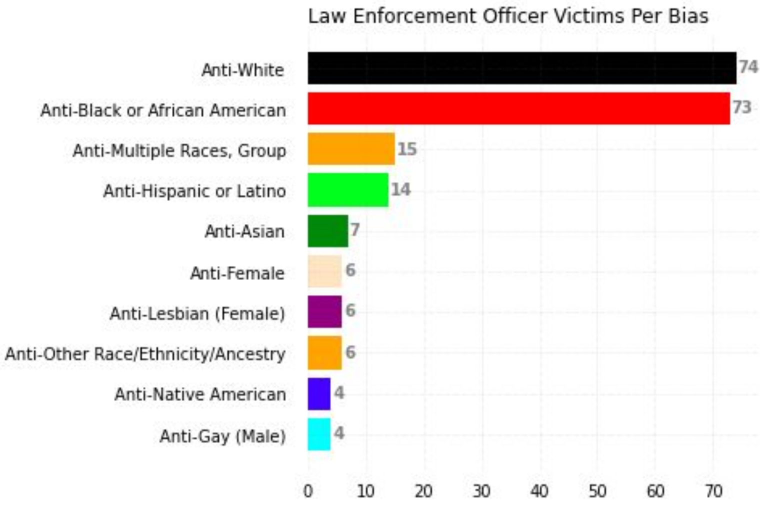



Hate-Crime Bias Over the Years (Majority Race/Group)
Anti-Black
Anti-White
Counts
5000
4000
3000
2000
1000
1990
1995
2000
2005
2010
2015
2020
Hate-Crime Bias Over the Years (Minority Race/Group)
Anti-Arab
Anti-Muslim
Anti-Asian
Anti-Hindu
Anti_Sikh
Counts
600
500
400
300
200
100
0
1990
1995
2000
2005
2010
2015
2020
Year


Business Victims Per Bias
Anti-Black or African American 2996
Anti-Jewish 2430
Anti-White 1043
Anti-Other Race/Ethnicity/Ancestry 690
Anti-Multiple Races, Group 536
Anti-Lesbian (Female) 383
Anti-Hispanic or Latino 322
Anti-Gay (Male) 304
Anti-Asian 304
Anti-Native American 251
0 500 1000 1500 2000 2500 3000


Government Victims Per Bias
Anti-Black or African American 2725
Anti-Jewish 1445
Anti-Multiple Races, Group 542
Anti-White 289
Anti-Other Race/Ethnicity/Ancestry 250
Anti-Gay (Male) 237
Anti-Lesbian (Female) 228
Anti-Hispanic or Latino 107
Anti-Multiple Religions 58
Anti-Islamic (Muslim) 57


Religious Organization Victims Per Bias
Anti-Jewish 2008
Anti-Other Religion 1368
Anti-Catholic 723
Anti-Protestant 657
Anti-Black or African American 466
Anti-Multiple Religions 350
Anti-Islamic (Muslim) 283
Anti-Other Christian 107
Anti-Lesbian (Female) 102
Anti-Other Race/Ethnicity/Ancestry 95
0 250 500 750 1000 1250 1500 1750 2000


Law Enforcement Officer Victims Per Bias
Anti-White 74
Anti-Black or African American 73
Anti-Multiple Races, Group 15
Anti-Hispanic or Latino 14
Anti-Asian 7
Anti-Female 6
Anti-Lesbian (Female) 6
Anti-Other Race/Ethnicity/Ancestry 6
Anti-Native American 4
Anti-Gay (Male) 4
0 10 20 30 40 50 60 70

Unsupervised learning

Dataframe was created to view incidents counts among states for all bias types and then for just Anti-Black bias
Columns in dataframe: State, offender count, victim count, incident count, bias count

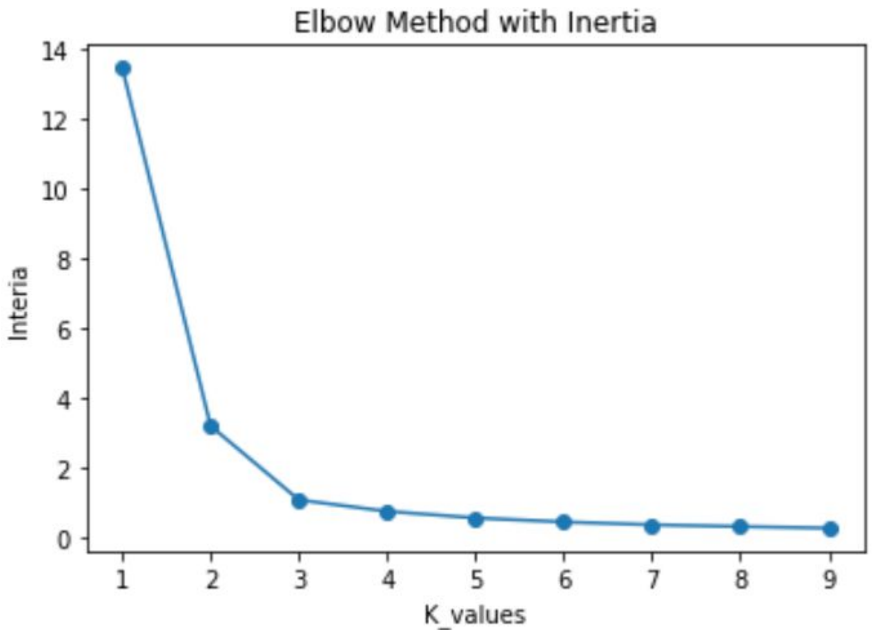
Clustering

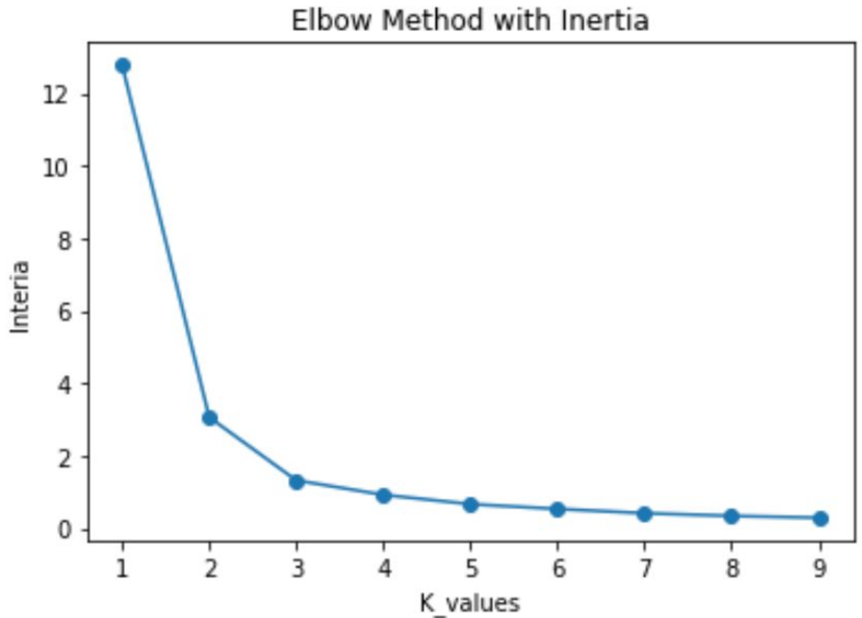
```
inertias = []

for k in range(1,10):
    #build and fit the model
    model = KMeans(n_clusters=k)
    model.fit(X_norm)
    inertia = model.inertia_
    inertias.append(inertia)
```

The inertia score when clustering for all bias types is 29%. The score for Anti-Black is 25%.

Unsupervised Learning

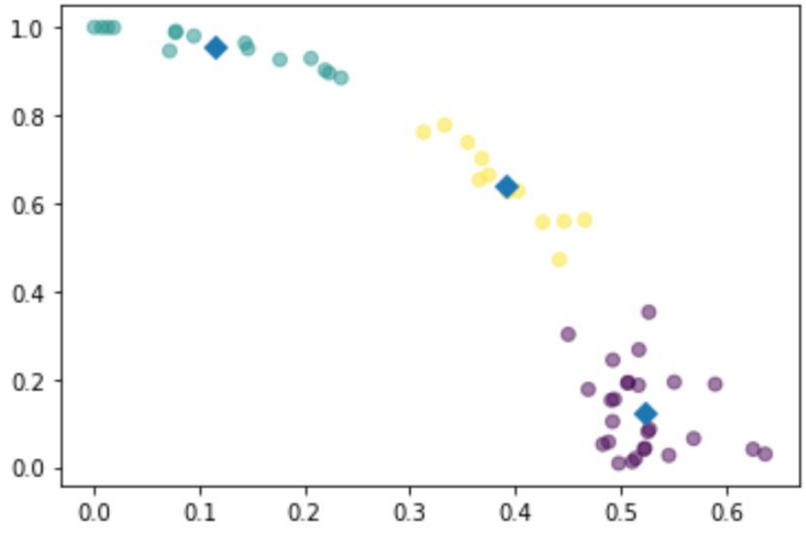
Anti-Black Bias	All Bias



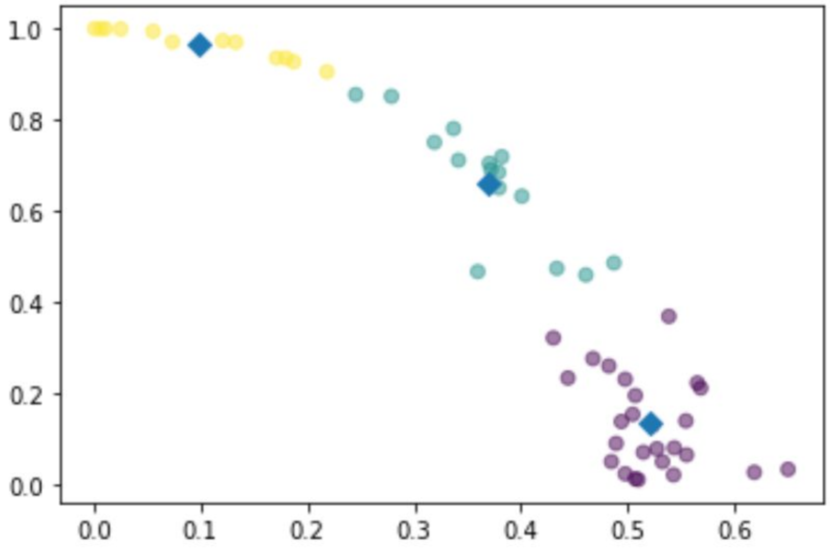


KMeans Cluster

Anti-Black Bias

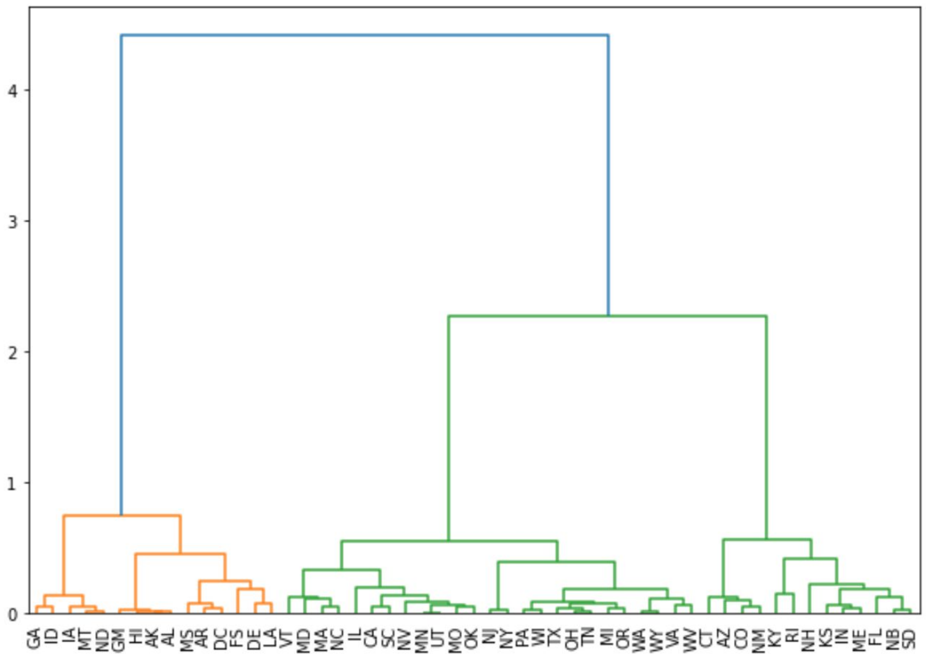


All Bias

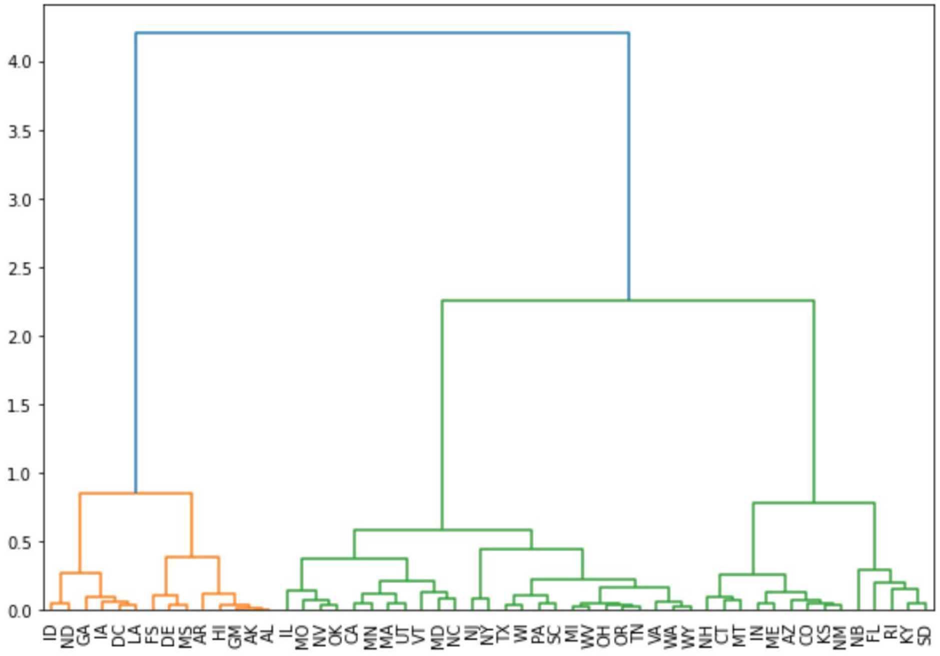


Hierarchical Cluster

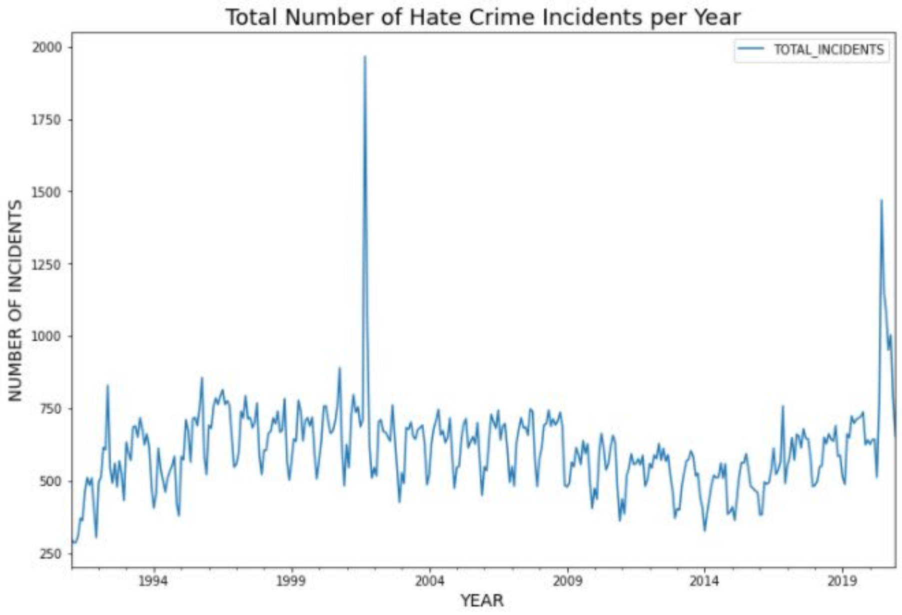
Anti-Black Bias

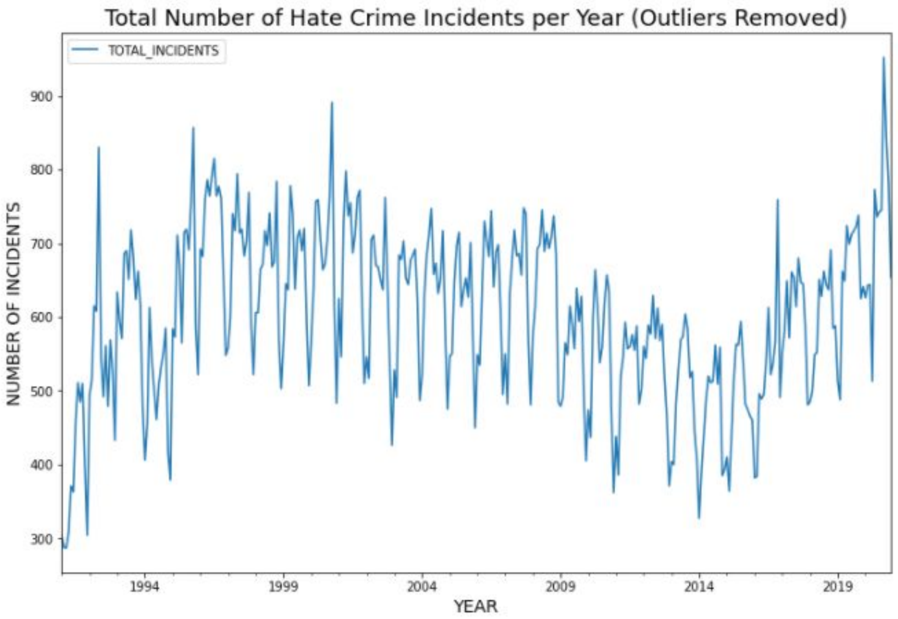


All Bias



Time Series Analysis

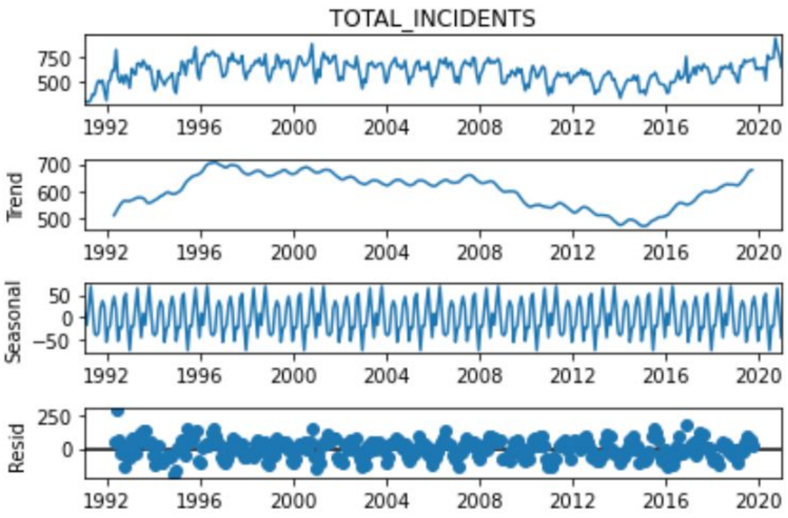




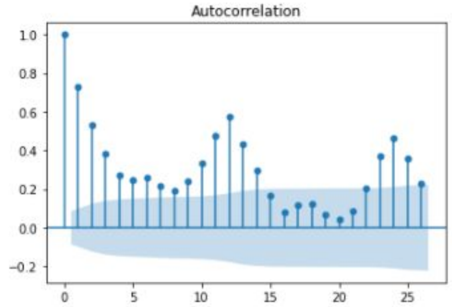
Total number of hate crime incidents per year

- Incidents month was established for each year
- Outliers removed: incidents >100

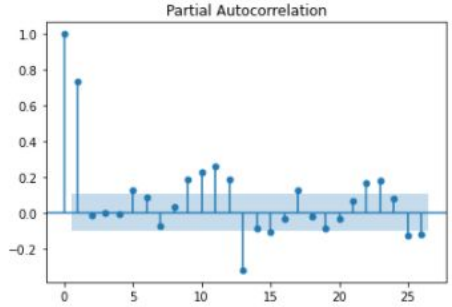
Seasonality and Stationarity




```
#ACF  (q value)
acf_array = acf(indexed_df['TOTAL_INCIDENTS'], fft=False)
plot_acf(indexed_df['TOTAL_INCIDENTS'], lags=26, alpha=ci)
plt.show()

#14 lags
```




```
#PACF (p value)
plot_pacf(indexed_df['TOTAL_INCIDENTS'], lags = 26)
plt.show()

#4 spikes
```



- Trend isn’t consistent
- Residuals account mostly for 2001 spike
- Inspect seasonality

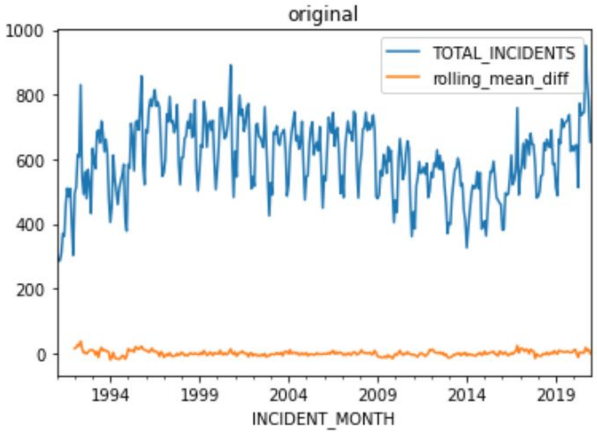
Augmented Dickey Fuller Test

```python
#Augmented Dickey Fuller test

dftest = adfuller(indexed_df.TOTAL_INCIDENTS, autolag = 'AIC')
print("ADF: ",dftest[0])
print("P-Value: ", dftest[1])
print("Num Of Lags: ", dftest[2])
print("Num Of Observations:", dftest[3])
print("Critical Values:")
for key, val in dftest[4].items():
    print("\t",key, ": ", val)

## p-value is high, well above threshold of 0.05
##dataset is non-stationary
```

```
ADF:  -2.3563481920398264
P-Value:  0.15443758406898878
Num Of Lags:  14
Num Of Observations: 345
Critical Values:
	 1% :  -3.4494474563375737
	 5% :  -2.8699542285903887
	 10% :  -2.5712527305187987
```

```python
rolling_mean = indexed_df['TOTAL_INCIDENTS'].rolling(window = 12).mean()
indexed_df['rolling_mean_diff'] = rolling_mean - rolling_mean.shift()

#plot original df and rolling mean difference together
indexed_df[['TOTAL_INCIDENTS', 'rolling_mean_diff']].plot(title='original')
```

<AxesSubplot:title={'center':'original'}, xlabel='INCIDENT_MONTH'>



ADF test on the rolling mean yields a lower p-value; lower than 0.05
Sationarity has been corrected with d=1

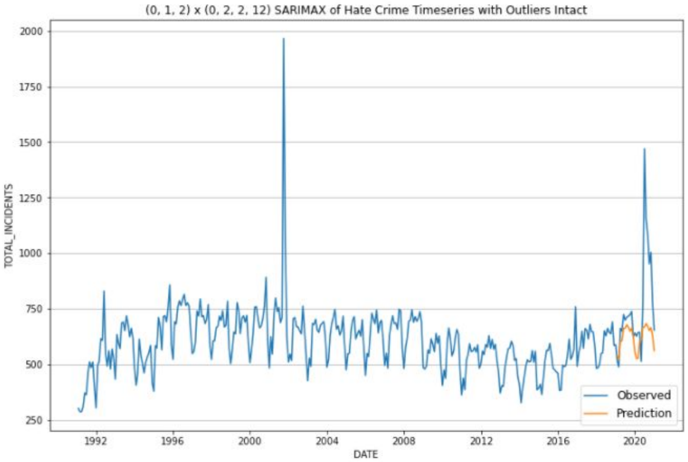
Sarimax Code

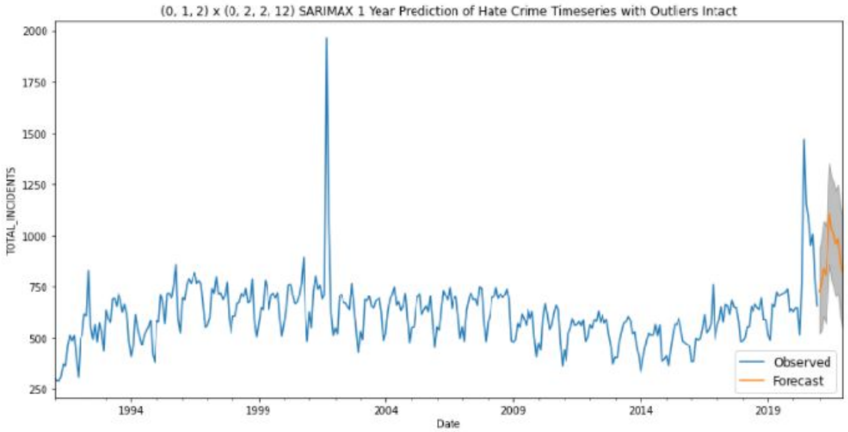
```
def sarimax_func(df, y_col, input_order, input_seasonal_order, pred_start, pred_end, title):
    model = SARIMAX(df[y_col], order=input_order, seasonal_order=input_seasonal_order)
    results = model.fit()
    print(results.summary())


    #prediction
    df['prediction']=results.predict(start=pred_start,end=pred_end,dynamic=True)

    #plotting
    plt.figure(figsize=(12,8))
    plt.grid(axis='y')
    plt.xlabel('DATE')
    plt.ylabel(y_col)
    plt.title(str(input_order) +(' x ')+ str(input_seasonal_order) + ' SARIMAX of ' + str(title))
    actual, = plt.plot(df[y_col], label='Observed')
    prediction, = plt.plot(df['prediction'], label='Prediction')
    legend = plt.legend(loc='lower right', fontsize='large')
    plt.show()

    # Forecast 1y
    years = 1
    pred_uc = results.get_forecast(steps=12*years)
    pred_ci = pred_uc.conf_int()
    ax = df[y_col].plot(label='Observed', figsize=(14, 7))
    pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
    ax.fill_between(pred_ci.index,
                    pred_ci.iloc[:, 0],
                    pred_ci.iloc[:, 1], color='k', alpha=.25)
    ax.set_xlabel('Date')
    ax.set_ylabel(y_col)
    plt.title(str(input_order) +(' x ')+ str(input_seasonal_order) + ' SARIMAX 1 Year Prediction of ' + str(title))
    plt.legend(loc='lower right', fontsize='large')
    plt.show()


    # Forecast 5y
    years = 5
    pred_uc = results.get_forecast(steps=12*years)
    pred_ci = pred_uc.conf_int()
    ax = df[y_col].plot(label='Observed', figsize=(14, 7))
    pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
    ax.fill_between(pred_ci.index,
                    pred_ci.iloc[:, 0],
                    pred_ci.iloc[:, 1], color='k', alpha=.25)
    ax.set_xlabel('Date')
    ax.set_ylabel(y_col)
    plt.title(str(input_order) +(' x ')+ str(input_seasonal_order) + ' SARIMAX 5 Year Prediction of ' + str(title))
    plt.legend(loc='lower right', fontsize='large')
    plt.show()
```

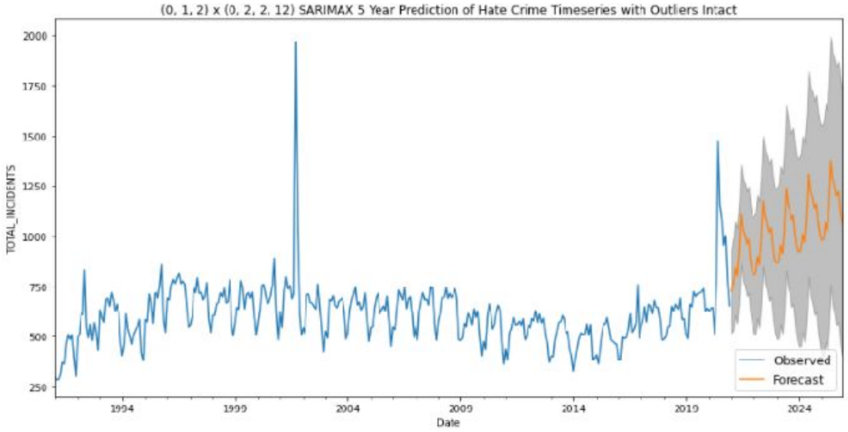
Lags for autoregressive model = 0, Differencing/integration order = 1 Moving average = 2
Seasonal Order:
P = 0,   D = 2, Q = 2, m=12
Best model from GridSearch:
(1, 0, 0) x (3, 3, 0, 12)

SARIMAX Results: Outliers Intact

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                       TOTAL_INCIDENTS   No. Observations:               360
Model:             SARIMAX(0, 1, 2)x(0, 2, 2, 12)   Log Likelihood               -1872.884
Date:                            Wed, 09 Mar 2022   AIC                           3755.769
Time:                                    00:33:17   BIC                           3774.839
Sample:                                01-31-1991   HQIC                          3763.372
                                     - 12-31-2020
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.4898      0.047    -10.393      0.000      -0.582      -0.397
ma.L2         -0.1311      0.047     -2.791      0.005      -0.223      -0.039
ma.S.L12      -1.7957      1.435     -1.251      0.211      -4.609       1.018
ma.S.L24       0.7967      1.127      0.707      0.479      -1.412       3.005
sigma2      3257.3544   4701.808      0.693      0.488   -5958.020    1.25e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.50   Jarque-Bera (JB):               152.35
Prob(Q):                              0.48   Prob(JB):                         0.00
Heteroskedasticity (H):               0.84   Skew:                             0.60
Prob(H) (two-sided):                  0.36   Kurtosis:                         6.07
===================================================================================
```

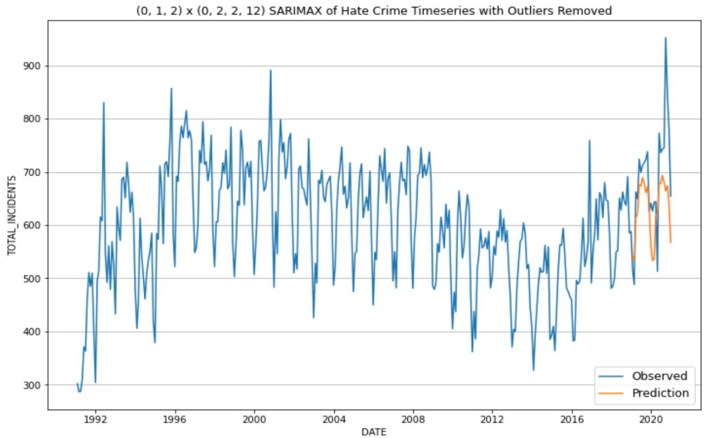


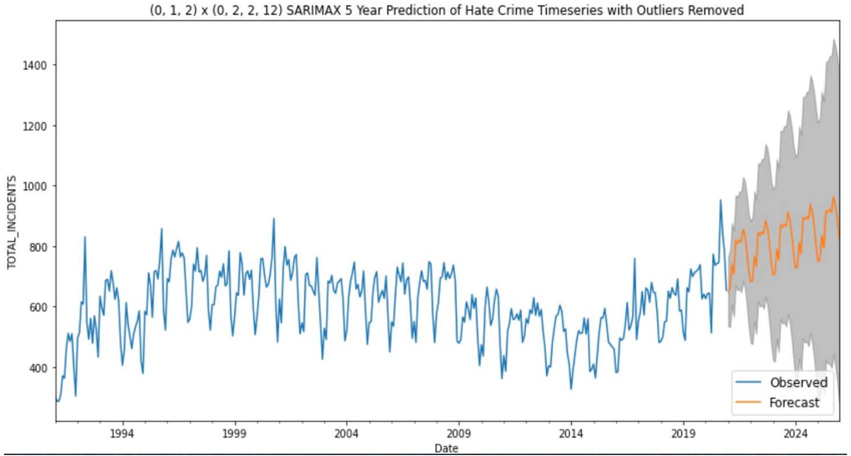


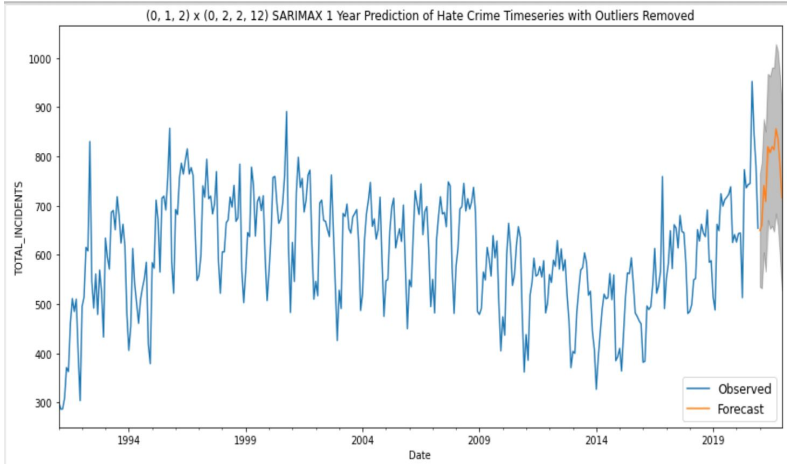


Sarimax Results: Outliers Removed

```
                                 SARIMAX Results
==========================================================================================
Dep. Variable:                    TOTAL_INCIDENTS   No. Observations:                  360
Model:             SARIMAX(0, 1, 2)x(0, 2, 2, 12)   Log Likelihood               -1872.884
Date:                            Wed, 09 Mar 2022   AIC                           3755.769
Time:                                    00:33:10   BIC                           3774.839
Sample:                                01-31-1991   HQIC                          3763.372
                                     - 12-31-2020
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.4898      0.047    -10.393      0.000      -0.582      -0.397
ma.L2         -0.1311      0.047     -2.791      0.005      -0.223      -0.039
ma.S.L12      -1.7957      1.435     -1.251      0.211      -4.609       1.018
ma.S.L24       0.7967      1.127      0.707      0.479      -1.412       3.005
sigma2      3257.3544   4701.808      0.693      0.488   -5958.020    1.25e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.50   Jarque-Bera (JB):               152.35
Prob(Q):                              0.48   Prob(JB):                         0.00
Heteroskedasticity (H):               0.84   Skew:                             0.60
Prob(H) (two-sided):                  0.36   Kurtosis:                         6.07
===================================================================================
```







Supervised Learning with Time Series

Steps taken

Preprocess

Add incidents count column
Isolate columns of interest:

- Bias description
- Region name
- Incident month
- Presidency

Categorical encoding

Time series data frame list

1. Region_Presidency Bias
2. Region Presidency TS
3. Region Bias TS
4. Region TS
5. Bias TS
6. Presidency TS

Engineering Lag Features

create lag features with shift() and diff()

```
df_list = [region_bias_presidency_ts, region_presidency_ts,
           region_bias_ts, region_ts, bias_ts, presidency_ts]

for df in df_list:
    df['MONTH-1'] = df['TOTAL_INCIDENTS'].shift()
    df['MONTH-2'] = df['MONTH-1'].shift()

    df['MONTH-1_DIFF'] = df['MONTH-1'].diff()
    df['MONTH-2_DIFF'] = df['MONTH-2'].diff()

    df.dropna(inplace=True)
    df.reset_index(inplace=True, drop=True)
    display(df.head(5))
```

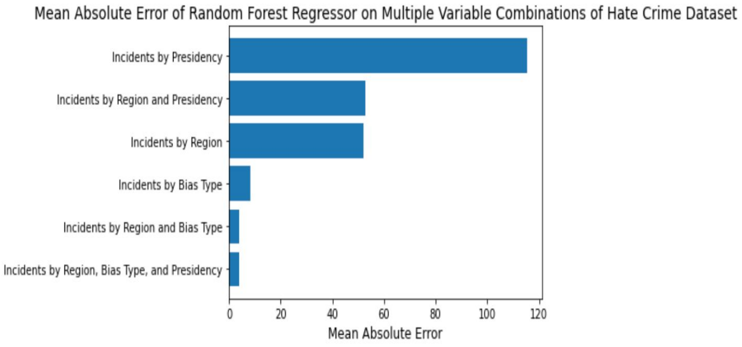
Modeling

```
def validation_split(dataset, y_column, validation_percentage):
    '''
    Input:
    dataset = dataframe processed for machine learning
    validation_percentage = percent of dataset to withold for validation, expressed as decimal

    Return:
    X_train, X_test, y_train, y_test
    '''
    #train test split
    #train test split with 85/15
    length = len(dataset)
    val_size = round(length * validation_percentage)
    cut_off = length-val_size

    X = dataset.drop([y_column], axis=1)
    y = dataset[y_column]

    X_train, X_test = X[:cut_off], X[cut_off:]
    y_train, y_test = y[:cut_off], y[cut_off:]

    return (X_train, X_test, y_train, y_test)
```

Random Forest Regressor

```python
def random_forest_func(dataset, num_estimators, dataset_string):
    '''
    Input:
    dataset = a preprocessed pandas df for machine learning
    dataset_string = the title of the dataset
    num_estimators = number of estimators for the Random Forest Regression model

    Function:
    Callss validation_split function, reserving the last 15% of the dataset for validation,
    Fits the dataset to a RandomForestRegressor and predicts results,
    Calculates MAE and RMSE for predictions compared to validation set.

    Return:
    Dataframe of dataset_string, calculated MAE, calculated RMSE
    '''
    #instantiate lists to convert to results dataframe
    MAE_list = []
    RMSE_list = []
    label_list = []

    #call validation_split function
    X_train, X_test, y_train, y_test = validation_split(dataset, 'TOTAL_INCIDENTS', 0.15)
    y_train, y_test = np.array(y_train), np.array(y_test)

    #instantiate, fit, and predict RandomForestRegressor model
    RF_model = RandomForestRegressor(n_estimators=num_estimators, n_jobs=-1, random_state=42).fit(X_train, y_trai
    y_pred = RF_model.predict(X_test)

    #calculate performance metrics
    MAE_result = MAE(y_test, y_pred)
    MAE_list.append(MAE_result)

    MSE_result = MSE(y_test, y_pred)
    RMSE_result = np.sqrt(MSE_result)
    RMSE_list.append(RMSE_result)

    label_list.append(dataset_string)

    return pd.DataFrame({'Title':label_list,
                         'MAE':MAE_list,
                         'RMSE':RMSE_list})
```

	Title	MAE	RMSE
0	Incidents by Region, Bias Type, and Presidency	3.892302	9.370970
0	Incidents by Region and Bias Type	3.893536	9.368843
0	Incidents by Bias Type	8.164020	24.381925
0	Incidents by Region	52.197874	70.015608
0	Incidents by Region and Presidency	52.831000	71.689317
0	Incidents by Presidency	115.396333	178.251851



Neural Network

```python
def neural_network_processing(dataset, y_column):
    '''
    Input:
    dataset = dataframe processed for machine learning
    y_column = column to predict as a string

    Function:
    Call validation_split function and splits into 85% training and 15% validation sets
    Converts train and validation sets into tensors and reshapes

    Return:
    X_train, y_train as reshaped tensors for PyTorch Neural Network Model
    '''
    #call validation_split function
    X_train, X_test, y_train, y_test = validation_split(dataset, y_column, 0.15)

    #convert arrays to tensors for PyTorch
    X_train_tensor = torch.FloatTensor(X_train.values)
    X_test_tensor = torch.FloatTensor(X_test.values)

    y_train_tensor = torch.FloatTensor(y_train.values)
    y_test_tensor = torch.FloatTensor(y_test.values)

    #reshape target tensors
    new_shape_train = (len(y_train_tensor), 1)
    y_train_tensor_reshape = y_train_tensor.view(new_shape_train)

    new_shape_test = (len(y_test_tensor), 1)
    y_test_tensor_reshape = y_test_tensor.view(new_shape_test)

    return X_train_tensor, y_train_tensor_reshape
```

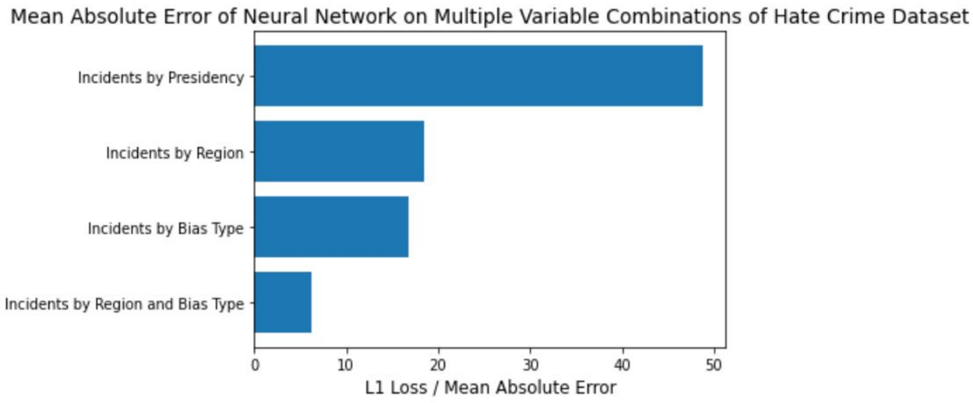
```python
neural_network_processing(region_ts, 'TOTAL_INCIDENTS')
```

```
(tensor([[   0.,    5.,   65.,  171., -106.,  134.],
         [   1.,    0.,   29.,   65.,  -36., -106.],
         [   1.,    1.,   51.,   29.,   22.,  -36.],
         ...,
         [ 313.,    3.,  156.,  107.,   49.,  -77.],
         [ 313.,    5.,  142.,  156.,  -14.,   49.],
         [ 314.,    0.,  177.,  142.,   35.,  -14.]]),
 tensor([[ 29.],
         [ 51.],
         [148.],
         ...,
         [142.],
         [177.],
         [116.]]))
```

```python
def neural_network(dataset, y_column, n_nodes, dataset_string):
    '''
    Input:
    dataset = dataframe processed for machine learning
    y_column = the column to predict as a string
    n_nodes = number of nodes in each hidden layer
    sdataset_string = title fo dataset

    Function:
    calls validation_split function, converts results into tensors and reshapes

    Return:
    dataframe containing the epoch number and loss result for all epochs divisible by 50
    '''
    X_train, y_train = neural_network_processing(dataset, y_column)

    X = dataset.drop([y_column], axis=1)

    #define PyTorch ANN model
    class ANN_Model(nn.Module):

        def __init__(self, input_features=len(X.columns),
                     hidden1=n_nodes,
                     hidden2=n_nodes,
                     hidden3=n_nodes,
                     out_features=1):
            super().__init__()
            self.layer1con = nn.Linear(input_features, hidden1)
            self.layer2con = nn.Linear(hidden1, hidden2)
            self.layer3con = nn.Linear(hidden2, hidden3)
            self.out = nn.Linear(hidden3, out_features)

        def forward(self, x):
            x = F.relu(self.layer1con(x))
            x = F.relu(self.layer2con(x))
            x = F.relu(self.layer3con(x))
            x = self.out(x)
            return(x)

    #call on model class
    ann = ANN_Model()
    loss_function = torch.nn.L1Loss()
    optimizer = torch.optim.Adam(ann.parameters(), lr=0.01)

    epoch_list = []
    loss_list = []

    for epoch in range(501):
        y_pred = ann.forward(X_train)
        loss = loss_function(y_pred, y_train)

        loss_detached = loss.detach().numpy()
        loss_list.append(loss_detached)
        epoch_list.append(epoch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Neural Network Results

	Epoch	Loss	Title
0	428	6.240892	Incidents by Region and Bias Type
0	494	16.854391	Incidents by Bias Type
0	497	18.526531	Incidents by Region
0	496	48.85291	Incidents by Presidency



Supervised Learning Parameter Tuning

```
X_train, X_test, y_train, y_test = validation_split(dataset=region_bias_ts, y_column='TOTAL_INCIDENTS',
```

```
params_dt = {'max_depth':[20, 50, 80],
             'min_samples_leaf':[0.0004,0.004],
             'max_features':[0.3, 0.5, 1]}

grid_dt = GridSearchCV(RandomForestRegressor(random_state=42, n_jobs=-1),
                       param_grid=params_dt,
                       scoring='neg_mean_squared_error')
grid_dt.fit(X_train, y_train)

#print best parameters
print('Best parameters from GridSearch CV: {:}'.format(grid_dt.best_params_))
print(' ')

#predict on test set with best estimator
y_pred_dt_tuned = grid_dt.best_estimator_.predict(X_test)

#RMSE of best estimator
MAE_dt_tuned = MAE(y_test, y_pred_dt_tuned)

print('Test MAE of tuned dt model: {:.2f}'.format(MAE_dt_tuned))
```

```
Best parameters from GridSearch CV: {'max_depth': 50, 'max_features': 0.5, 'min_samples_leaf': 0.000
4}

Test MAE of tuned dt model: 4.37
```

```
p = d = q = range(1,4)
```

```
pdq = list(itertools.product(p, d, q))
```

```
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

```
ts = indexed_df[['TOTAL_INCIDENTS']]
```

```
def sarimax_gridsearch(ts, pdq, pdqs, maxiter=50, freq='M'):
    '''
    input:
        ts : timesereis dataframe
        pdq : ARIMA combinations to test
        pdqs : seasonal ARIMA combinations from above
        maxiter : number of iterations, increase if your model isn't converging
        frequency : default='M' for month. Change to suit your time series frequency
            e.g. 'D' for day, 'H' for hour, 'Y' for year.

    Return:
        Prints out top 5 parameter combinations
        Returns dataframe of parameter combinations ranked by BIC
    '''

    # Run a grid search with pdq and seasonal pdq parameters and get the best BIC value
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(ts, # this is your time series you will input
                                                order=comb,
                                                seasonal_order=combs,
                                                freq=freq)

                output = mod.fit(maxiter=maxiter)

                if output.bic < 3400:
                    ans.append([comb, combs, output.bic])
                    print('SARIMAX {} x {} : BIC Calculated ={}'.format(comb, combs, output.bic))
                else:
                    continue
            except:
                continue

    # Find the parameters with minimal BIC value
    # Convert into dataframe
    ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'bic'])

    # Sort and return top 5 combinations
    ans_df = ans_df.sort_values(by=['bic'],ascending=True)[0:5]

    return ans_df
```

Limitations and Further Analysis

1. Include population size for each state and have the demographics
2. Due to some data exploration, it's obvious that there is under-reporting or no data was collected prior to certain years
3. Another data set with more political information would have provided more insight on if the amount and types of hate crimes had an impact on electoral outcome
4. Robust data on various types of hate crimes and counts, so this data set could be analyzed in multiple ways
5. It would be insightful to analyze how people voted in each state and see if there is a correlation between cities/states where democrats/republicans got the most votes and hate crimes
6. Increases in hate crime coincide with historical events