Predicting Future Covid-19 Cases By Growth Curve Estimation

Contributors: Davis Ulrich, Andrew Kaplan, & Victoria Austin

5/15/2020

Setup

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set(style = "whitegrid", color_codes = True, font_scale = 1.5)
```

In [2]:
```
#Import Data as DataFrame
confirmed_time_series = pd.read_csv("time_series_covid19_confirmed_US.csv")

# Column Info: https://github.com/Yu-Group/covid19-severity-prediction/blob/master/data/list_of_columns.md
county_demo = pd.read_csv("county_data_abridged.csv")
```

In [3]:
```
#Structure Analysis
#Number of Records

c_ts_shape_raw = confirmed_time_series.shape
county_shape_raw = county_demo.shape
print('Number day recorderd in confirmed_time_series:', c_ts_shape_raw[1])
print('Number of counties in confirmed_time_series', c_ts_shape_raw[0])
print('Number of counties with demographic data in county_demo:', county_shape_raw[0])
print('Number of demographic variables tracked in county_demo:', county_shape_raw[1])
```

```
Number day recorderd in confirmed_time_series: 118
Number of counties in confirmed_time_series 3261
Number of counties with demographic data in county_demo: 3244
Number of demographic variables tracked in county_demo: 87
```

Data Cleaning Part 1: County Data

In [4]:
```
county_demo.head()
```

Out[4]:

	countyFIPS	STATEFP	COUNTYFP	CountyName	StateName	State	lat	lon	POP_LATITUDE	POP_LONGITUDE	...	>500 gatherings	public schools	restaurant dine-in	enterta
0	01001	1.0	1.0	Autauga	AL	Alabama	32.540091	-86.645649	32.500389	-86.494165	...	737497.0	737500.0	737503.0	
1	01003	1.0	3.0	Baldwin	AL	Alabama	30.738314	-87.726272	30.548923	-87.762381	...	737497.0	737500.0	737503.0	
2	01005	1.0	5.0	Barbour	AL	Alabama	31.874030	-85.397327	31.844036	-85.310038	...	737497.0	737500.0	737503.0	
3	01007	1.0	7.0	Bibb	AL	Alabama	32.999024	-87.125260	33.030921	-87.127659	...	737497.0	737500.0	737503.0	
4	01009	1.0	9.0	Blount	AL	Alabama	33.990440	-86.562711	33.955243	-86.591491	...	737497.0	737500.0	737503.0	

5 rows × 87 columns

In [5]:
```
# Counties without COUNTYFP codes overwhelmingly have missing data values, and will be dropped

county_demo = county_demo[~county_demo.COUNTYFP.isna()]
#Change 'countyFIPS' from str to int
county_demo['countyFIPS'] = county_demo['countyFIPS'].astype(int)
# Puerto Rican municipalities dropped to maintain consistency of scope of analysis
county_demo = county_demo[county_demo.StateName != 'PR']
```

In [6]:
```
# Alaska, Hawaii, and Virginia counties often have no records in following columns: State', 'lat', 'lon'
# Impute 'lat'/'lon' using 'POP_LATITUDE'/'POP_LONGUITUDE'

county_demo.lat = county_demo.lat.fillna(county_demo.POP_LATITUDE)
county_demo.lon = county_demo.lon.fillna(county_demo.POP_LONGITUDE)
# Impute 'State' colums using corresponding 'StateName' data for Alaska, Hawaii, and Virginia
county_demo.loc[(county_demo.StateName == 'AK'), 'State'] = 'Alaska'
county_demo.loc[(county_demo.StateName == 'HI'), 'State'] = 'Hawaii'
county_demo.loc[(county_demo.StateName == 'VA'), 'State'] = 'Virginia'
```

In [7]:
```
# Fill in NaN values, convert Gregorian Ordinal time to date time

def ordinal_to_datetime(data, column):
    data[column] = data[column].fillna('No Order Issued')
    arr = data[column].to_numpy()
    for i in range(len(arr)):
        if isinstance(arr[i], float):
            arr[i] = (pd.Timestamp.fromordinal(int(arr[i]))).date()
    data[column] = arr
    return data
```

In [8]:
```
# Reformatting Date into a recognizable format

county_demo = ordinal_to_datetime(county_demo, 'stay at home')
county_demo = ordinal_to_datetime(county_demo, '>50 gatherings')
county_demo = ordinal_to_datetime(county_demo, '>500 gatherings')
county_demo = ordinal_to_datetime(county_demo, 'public schools')
county_demo = ordinal_to_datetime(county_demo, 'restaurant dine-in')
county_demo = ordinal_to_datetime(county_demo, 'entertainment/gym')
county_demo = ordinal_to_datetime(county_demo, 'federal guidelines')
county_demo = ordinal_to_datetime(county_demo, 'foreign travel ban')
```

In [9]:
```
# The latest CDC data shows that that fataility rate of disgnosed covid patients between the ages of 20-54 is <1%,
# No confirmed patients age 19 and under have died.
# Taking this into consideration, the 3 Year mortality rates for ages 0-54, all of which have less than 68%
# rate of data collection for counties being analayzed, thus we will drop the columns

county_demo = county_demo.drop('3-YrMortalityAge<1Year2015-17', axis = 1)
county_demo = county_demo.drop('3-YrMortalityAge1-4Years2015-17', axis = 1)
county_demo = county_demo.drop('3-YrMortalityAge5-14Years2015-17', axis = 1)
county_demo = county_demo.drop('3-YrMortalityAge15-24Years2015-17', axis = 1)
county_demo = county_demo.drop('3-YrMortalityAge25-34Years2015-17', axis = 1)
county_demo = county_demo.drop('3-YrMortalityAge35-44Years2015-17', axis = 1)
county_demo = county_demo.drop('3-YrMortalityAge45-54Years2015-17', axis = 1)

# 'mortality2015-17Estimated' is collected in <4% of counties
county_demo = county_demo.drop('mortality2015-17Estimated', axis = 1)

# '3-YrDiabetes2015-17' is collected in <50% of counties, while diabeties rate is collected in >99% of counties
# and should
# serve as a suitable metric for diabetes impact on virus outcome
county_demo = county_demo.drop('3-YrDiabetes2015-17', axis = 1)
```

In [10]:
```
# Oglala Lakota County, South Dakota (formely Shannon County), and Bedford City, VA have very low
# rates of data collection.
# Oglala Lakota county is entirely within the Pine Ridge Indian Reservation, and remains 'unorganized'
# Source: https://en.wikipedia.org/wiki/Oglala_Lakota_County,_South_Dakota

county_demo = county_demo[county_demo.countyFIPS != 46113]
county_demo = county_demo[county_demo.countyFIPS != 51515]
```

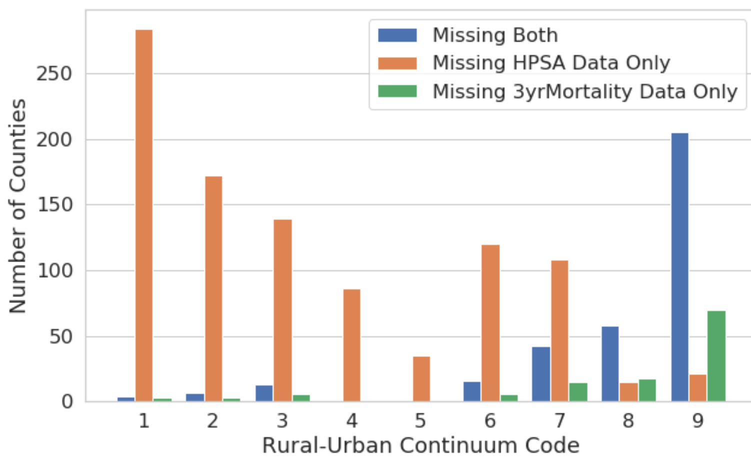
In [11]:
```
# Impute StrokeMortality data into the 8 Counties with NaN values using the mean
county_codes = county_demo[county_demo.StrokeMortality.isnull()]['countyFIPS'].to_list()
for county_code in county_codes:
    county_demo.loc[(county_demo.countyFIPS == county_code), 'StrokeMortality'] = county_demo.StrokeMortality.mean()

# Impute HeartDiseaseMortality data into the 8 counties with NaN values (all in Alaska)
# According to latest CDC statistics, avg rate for alaska is 129.7 [https://www.cdc.gov/nchs/pressroom/sosmap/heart_disease_mortali
ty/heart_disease.htm]
county_codes = county_demo[county_demo['HeartDiseaseMortality'].isnull()]['countyFIPS'].to_list()
for county_code in county_codes:
    county_demo.loc[(county_demo.countyFIPS == county_code), 'HeartDiseaseMortality'] = 129.7

#Impute DiabetesPercentage data into the 1 county with NaN values (in Alaska) using Alaska mean rate
county_codes = county_demo[county_demo['DiabetesPercentage'].isnull()]['countyFIPS'].to_list()
for county_code in county_codes:
    county_demo.loc[(county_demo.countyFIPS == county_code), 'DiabetesPercentage'] = county_demo[county_demo.StateName == 'AK']['Di
abetesPercentage'].mean()
```

In [12]:
```
# Impute 28 total counties with missing medicare
# eligibility rate measures average percentage increase in number of eligible medicare patients in 2017 enrollment

eligibility_rate = county_demo['#EligibleforMedicare2018'].mean() / county_demo['MedicareEnrollment,AgedTot2017'].mean()

# 21 counties with missing 'MedicareEnrollment,AgedTot2017' data
# Nebraska: 9, Texas: 6, Idaho: 1, Montana: 1, South Dakota: 1, Colorado: 1

missing_enrollment = county_demo[county_demo['MedicareEnrollment,AgedTot2017'].isnull()]['countyFIPS'].to_list()
for county_code in missing_enrollment:
    eligible = county_demo.loc[(county_demo.countyFIPS == county_code), '#EligibleforMedicare2018']
    county_demo.loc[(county_demo.countyFIPS == county_code), 'MedicareEnrollment,AgedTot2017'] = round(eligible/eligibility_rate)

# 8 counties with missing '#EligibleforMedicare2018' data
# Alaska: 7, Hawaii: 1

missing_eligibility = county_demo[county_demo['#EligibleforMedicare2018'].isnull()]['countyFIPS'].to_list()
for county_code in missing_eligibility:
    enrolled = county_demo.loc[(county_demo.countyFIPS == county_code), 'MedicareEnrollment,AgedTot2017']
    county_demo.loc[(county_demo.countyFIPS == county_code), '#EligibleforMedicare2018'] = round(enrolled*eligibility_rate)
```

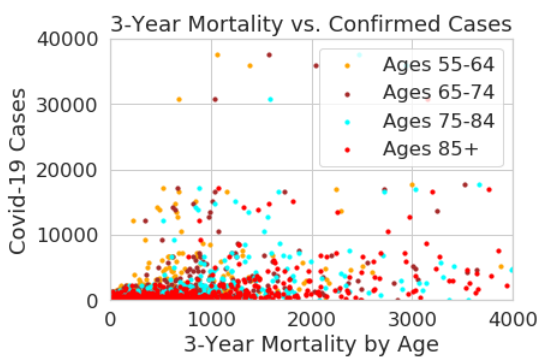
In [13]:
```
# The portion of rows in each column that are not null
# These are the ten highest null value columns

(~county_demo.isna()).mean().sort_values().head(10)
```

Out[13]:
```
HPSAUnderservedPop                   0.649475
HPSAServedPop                        0.649475
HPSAShortage                         0.649475
3-YrMortalityAge55-64Years2015-17    0.851640
3-YrMortalityAge65-74Years2015-17    0.902260
3-YrMortalityAge75-84Years2015-17    0.931869
3-YrMortalityAge85+Years2015-17      0.949698
dem_to_rep_ratio                     0.991404
SVIPercentile                        0.999682
PopMale30-342010                     1.000000
dtype: float64
```

In [14]:
```
# Dropping counties that do no have 3 year mortaility rate information for elder ages disportionalty
# filters rural communities

rural_urban = county_demo['Rural-UrbanContinuumCode2013']
Null_Both = county_demo[county_demo['3-YrMortalityAge55-64Years2015-17'].isnull()]
Null_Both = Null_Both[Null_Both['HPSAShortage'].isnull()]
Null_3yrMortality_Only = county_demo[county_demo['3-YrMortalityAge55-64Years2015-17'].isnull()]
Null_3yrMortality_Only = Null_3yrMortality_Only[~Null_3yrMortality_Only.countyFIPS.isin(Null_Both.countyFIPS)]
Null_HPSA_Only = county_demo[county_demo['HPSAShortage'].isnull()]
Null_HPSA_Only = Null_HPSA_Only[~Null_HPSA_Only.countyFIPS.isin(Null_Both.countyFIPS)]
fig, ax = plt.subplots(figsize=(10,6))
ax.grid(b=None, axis = 'x')
ax.set_xticks(np.arange(1,10))
ax.set_xlabel('Rural-Urban Continuum Code')
ax.set_ylabel('Number of Counties')
ax.hist((Null_3yrMortality_Only['Rural-UrbanContinuumCode2013'],
Null_HPSA_Only['Rural-UrbanContinuumCode2013'],
Null_Both['Rural-UrbanContinuumCode2013']),
bins = np.arange(min(rural_urban)-.5, max(rural_urban) + 1.5,1),
label = {'Missing 3yrMortality Data Only','Missing HPSA Data Only','Missing Both'});
ax.legend();
```



In [15]:
```
# Observations: More counties lack HPSA data than 3yrMortalityRate data (ages 55 and up) in total, but the
# distribution of county type by rural-urban continum code is different.

# Counties without HPSA data skew more urban while those without 3yrMortalityRate data are overwhelmingly rural,
# which is also the case for counties missing both data.

# Counties without HPSA data make up ~35% of the all counties, while those without 3yrMortalityRate make up ~15%
# of all counties.

# Decision: Drop the HPSA columns, keep all rows

# Rational: If HPSA data became part of the predictive model, the scope of our analysis would decrease significantly
# and disproportionaly impact analysis of urban areas, which are likely to be hardest hit.

# If 3-YrMOrtalityRate becomes part of the predictive model, we can use progressivly older age bins that have more
# higher rates of data collection than the 55-64 bin. Choosing to drop counties without this data would hurt our
# models ability to predict outcomes for rural counties.

# GRAPH: this graph is showing that 3-year mortality and number of Covid-19 cases has no correlation
# and is therefore admissable to exclude from our model. Other features that have many null values and
# do not improve our model are also dropped, such as HPSA columns.

county_health = county_demo.copy()
confirmed_copy = confirmed_time_series.copy().dropna()
int_v = [int(i) for i in confirmed_copy['FIPS']]
confirmed_copy['FIPS'] = [str(i) for i in int_v]

int_v2 = [int(i) for i in county_demo["countyFIPS"]]
county_health['FIPS'] = [str(i) for i in int_v2]
mortality_is_a_sham = county_health.merge(confirmed_copy, on = 'FIPS', how = 'left')[[
        '3-YrMortalityAge55-64Years2015-17','3-YrMortalityAge65-74Years2015-17', '3-YrMortalityAge75-84Years2015-17',
            '3-YrMortalityAge85+Years2015-17', '5/7/20']]

%matplotlib inline
fig = plt.figure()
ax2 = fig.add_subplot(111)

mortality_columns = ['3-YrMortalityAge55-64Years2015-17', '3-YrMortalityAge65-74Years2015-17',
                     '3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17']

ax2.scatter(mortality_is_a_sham[mortality_columns[0]], mortality_is_a_sham['5/7/20'], s=10, c='orange',
                            marker="o", label= 'Ages 55-64')
ax2.scatter(mortality_is_a_sham[mortality_columns[1]], mortality_is_a_sham['5/7/20'], s=10, c='brown',
                            marker="o", label= 'Ages 65-74')
ax2.scatter(mortality_is_a_sham[mortality_columns[2]], mortality_is_a_sham['5/7/20'], s=10, c='cyan',
                            marker="o", label= 'Ages 75-84')
ax2.scatter(mortality_is_a_sham[mortality_columns[3]], mortality_is_a_sham['5/7/20'], s=10, c='red',
                            marker="o", label= 'Ages 85+')
plt.ylim(0, 40000)
plt.xlim(0, 4000)
plt.legend(loc='upper right')
plt.xlabel('3-Year Mortality by Age')
plt.ylabel('Covid-19 Cases')
plt.title('3-Year Mortality vs. Confirmed Cases')
plt.show()
```



In [16]:
```
# 1 county exists in the county_demo dataset that is not in time_series dataset
# A little digging shows that this is Wade Hampton county in Alaska, this row is dropped from county_demo
county_demo = county_demo.drop(90)
```

In [17]:
```
county_demo = county_demo.drop(['HPSAShortage', 'HPSAServedPop', 'HPSAUnderservedPop'],axis=1)
county_demo = county_demo.dropna()
```

Data Cleaning Part 2: Confirmed Cases Data

In [18]:
```
confirmed_time_series.head()
```

Out[18]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_	...	4/28/20	4/29/20	4/30/20	5/1/20	5/2/20	5/3/20	5/4/20	5/5/20	5/6/20	5/7
0	16	AS	ASM	16	60.0	NaN	American Samoa	US	-14.2710	-170.1320	...	0	0	0	0	0	0	0	0	0	
1	316	GU	GUM	316	66.0	NaN	Guam	US	13.4443	144.7937	...	141	141	145	145	145	145	145	145	149	
2	580	MP	MNP	580	69.0	NaN	Northern Mariana Islands	US	15.0979	145.6739	...	14	14	14	14	14	14	14	14	15	
3	630	PR	PRI	630	72.0	NaN	Puerto Rico	US	18.2208	-66.5901	...	1400	1433	1539	1575	1757	1808	1843	1924	1968	2
4	850	VI	VIR	850	78.0	NaN	Virgin Islands	US	18.3358	-64.8963	...	57	57	66	66	66	66	66	66	66	

5 rows × 118 columns

In [19]:
```
# Primary Key: FIPS <--> countyFIPS

confirmed_time_series = confirmed_time_series[~confirmed_time_series.FIPS.isnull()]
confirmed_time_series['FIPS'] = (confirmed_time_series['FIPS']).astype(int)
confirmed_time_series = confirmed_time_series[confirmed_time_series.FIPS.isin(county_demo.countyFIPS)]
```

In [20]:
```
def select_columns(data, *columns):
    """Select only columns passed as arguments."""
    return data.loc[:, columns]
```

In [21]:
```
# All boroughs in New York have confirmed cases coded into New York county, New York
# New York County is in Reality only Manhattan, this needs to be accounted for.
# If we don't correct this, when similar counties chooses a New York county, it will show up as zero cases.

boroughs = ['Bronx' , 'Kings', 'New York', 'Queens', 'Richmond']
ny_confirmed = confirmed_time_series[confirmed_time_series.Province_State == 'New York']
nyc_confirmed = ny_confirmed[ny_confirmed.Admin2.isin(boroughs)]
nyc_confirmed
```

Out[21]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	Lat	Long_	...	4/28/20	4/29/20	4/30/20	5/1/20	5/2/20	5/3/20	5/4/20	5
1835	84036005	US	USA	840	36005	Bronx	New York	US	40.852093	-73.862828	...	0	0	0	0	0	0	0	
1856	84036047	US	USA	840	36047	Kings	New York	US	40.636182	-73.949356	...	0	0	0	0	0	0	0	
1863	84036061	US	USA	840	36061	New York	New York	US	40.767273	-73.971526	...	162338	164841	167478	169690	172354	174331	175651	17
1873	84036081	US	USA	840	36081	Queens	New York	US	40.710881	-73.816847	...	0	0	0	0	0	0	0	
1875	84036085	US	USA	840	36085	Richmond	New York	US	40.585822	-74.148086	...	0	0	0	0	0	0	0	

5 rows × 118 columns

In [22]:
```
# Condense County Demographics data for 5 counties into 1 single series

ny_demo = county_demo.loc[county_demo.State == 'New York']
nyc_demo = ny_demo.loc[ny_demo.CountyName.isin(boroughs)]
numeric_nyc = select_columns(nyc_demo, 'PopulationEstimate2018', 'PopTotalMale2017', 'PopTotalFemale2017',
'FracMale2017', 'PopulationEstimate65+2017',
'PopulationDensityperSqMile2010', 'CensusPopulation2010',
'MedianAge2010', '#EligibleforMedicare2018',
'MedicareEnrollment,AgedTot2017', 'DiabetesPercentage',
'HeartDiseaseMortality', 'StrokeMortality', 'Smokers_Percentage',
'RespMortalityRate2014', '#FTEHospitalTotal2017',
"TotalM.D.'s,TotNon-FedandFed2017", '#HospParticipatinginNetwork2017',
'#Hospitals', '#ICU_beds', 'dem_to_rep_ratio', 'PopMale<52010',
'PopFmle<52010', 'PopMale5-92010', 'PopFmle5-92010', 'PopMale10-142010',
'PopFmle10-142010', 'PopMale15-192010', 'PopFmle15-192010',
'PopMale20-242010', 'PopFmle20-242010', 'PopMale25-292010',
'PopFmle25-292010', 'PopMale30-342010', 'PopFmle30-342010',
'PopMale35-442010', 'PopFmle35-442010', 'PopMale45-542010',
'PopFmle45-542010', 'PopMale55-592010', 'PopFmle55-592010',
'PopMale60-642010', 'PopFmle60-642010', 'PopMale65-742010',
'PopFmle65-742010', 'PopMale75-842010', 'PopFmle75-842010',
'PopMale>842010', 'PopFmle>842010', '3-YrMortalityAge55-64Years2015-17',
'3-YrMortalityAge65-74Years2015-17',
'3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17')

nyc_as_one_county = numeric_nyc.cumsum()

nyc_as_one_county[['FracMale2017','PopulationDensityperSqMile2010','MedianAge2010','DiabetesPercentage',
'HeartDiseaseMortality', 'StrokeMortality', 'Smokers_Percentage','RespMortalityRate2014',
'dem_to_rep_ratio', '3-YrMortalityAge55-64Years2015-17','3-YrMortalityAge65-74Years2015-17',
'3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17']] = nyc_as_one_county[['FracMale2017',
'PopulationDensityperSqMile2010','MedianAge2010','DiabetesPercentage',
'HeartDiseaseMortality', 'StrokeMortality', 'Smokers_Percentage','RespMortalityRate2014',
'dem_to_rep_ratio', '3-YrMortalityAge55-64Years2015-17','3-YrMortalityAge65-74Years2015-17',
'3-YrMortalityAge75-84Years2015-17', '3-YrMortalityAge85+Years2015-17']] / 5

nyc_as_one_county = nyc_as_one_county.loc[nyc_as_one_county.PopulationEstimate2018 == 8398748.0]
```

In [23]:
```
#Assign relevant data from this series to New York County, New York in county_demo

county_demo.loc[county_demo.CountyName == 'New York', 'PopulationEstimate2018']= nyc_as_one_county['PopulationEstimate2018']
county_demo.loc[county_demo.CountyName == 'New York', 'PopTotalMale2017'] = nyc_as_one_county['PopTotalMale2017']
county_demo.loc[county_demo.CountyName == 'New York', 'PopTotalFemale2017'] = nyc_as_one_county['PopTotalFemale2017']
county_demo.loc[county_demo.CountyName == 'New York', 'FracMale2017'] = nyc_as_one_county['FracMale2017']
county_demo.loc[county_demo.CountyName == 'New York', 'PopulationEstimate65+2017'] = nyc_as_one_county['PopulationEstimate65+2017']
county_demo.loc[county_demo.CountyName == 'New York', 'PopulationDensityperSqMile2010'] = nyc_as_one_county['PopulationDensityperSq
Mile2010']
county_demo.loc[county_demo.CountyName == 'New York', 'CensusPopulation2010'] = nyc_as_one_county['CensusPopulation2010']
county_demo.loc[county_demo.CountyName == 'New York', 'MedianAge2010'] = nyc_as_one_county['MedianAge2010']
county_demo.loc[county_demo.CountyName == 'New York', '#EligibleforMedicare2018'] = nyc_as_one_county['#EligibleforMedicare2018']
county_demo.loc[county_demo.CountyName == 'New York', 'MedicareEnrollment,AgedTot2017'] = nyc_as_one_county['MedicareEnrollment,Age
dTot2017']
county_demo.loc[county_demo.CountyName == 'New York','DiabetesPercentage'] = nyc_as_one_county['DiabetesPercentage']
county_demo.loc[county_demo.CountyName == 'New York', 'HeartDiseaseMortality'] = nyc_as_one_county['HeartDiseaseMortality']
county_demo.loc[county_demo.CountyName == 'New York', 'StrokeMortality'] = nyc_as_one_county['StrokeMortality']
county_demo.loc[county_demo.CountyName == 'New York', 'Smokers_Percentage'] = nyc_as_one_county['Smokers_Percentage']
county_demo.loc[county_demo.CountyName == 'New York', 'RespMortalityRate2014'] = nyc_as_one_county['RespMortalityRate2014']
county_demo.loc[county_demo.CountyName == 'New York', '#FTEHospitalTotal2017'] = nyc_as_one_county['#FTEHospitalTotal2017']
county_demo.loc[county_demo.CountyName == 'New York', "TotalM.D.'s,TotNon-FedandFed2017"] = nyc_as_one_county["TotalM.D.'s,TotNon-F
edandFed2017"]
county_demo.loc[county_demo.CountyName == 'New York', '#HospParticipatinginNetwork2017'] = nyc_as_one_county['#HospParticipatinginN
etwork2017']
county_demo.loc[county_demo.CountyName == 'New York', '#Hospitals'] = nyc_as_one_county['#Hospitals']
county_demo.loc[county_demo.CountyName == 'New York', '#ICU_beds'] = nyc_as_one_county['#ICU_beds']
county_demo.loc[county_demo.CountyName == 'New York', 'dem_to_rep_ratio'] = nyc_as_one_county['dem_to_rep_ratio']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale<52010'] = nyc_as_one_county['PopMale<52010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle<52010'] = nyc_as_one_county['PopFmle<52010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale5-92010'] = nyc_as_one_county['PopMale5-92010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle5-92010'] = nyc_as_one_county['PopFmle5-92010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale10-142010'] = nyc_as_one_county['PopMale10-142010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle10-142010'] = nyc_as_one_county['PopFmle10-142010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale15-192010'] = nyc_as_one_county['PopMale15-192010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle15-192010'] = nyc_as_one_county['PopFmle15-192010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale20-242010'] = nyc_as_one_county['PopMale20-242010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle20-242010'] = nyc_as_one_county['PopFmle20-242010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale25-292010'] = nyc_as_one_county['PopMale25-292010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle25-292010'] = nyc_as_one_county['PopFmle25-292010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale30-342010'] = nyc_as_one_county['PopMale30-342010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle30-342010'] = nyc_as_one_county['PopFmle30-342010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale35-442010'] = nyc_as_one_county['PopMale35-442010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle35-442010'] = nyc_as_one_county['PopFmle35-442010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle35-442010'] = nyc_as_one_county['PopFmle35-442010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale45-542010'] = nyc_as_one_county['PopMale45-542010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale45-542010'] = nyc_as_one_county['PopMale45-542010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle45-542010'] = nyc_as_one_county['PopFmle45-542010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale55-592010'] = nyc_as_one_county['PopMale55-592010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle55-592010'] = nyc_as_one_county['PopFmle55-592010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale60-642010'] = nyc_as_one_county['PopMale60-642010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle60-642010'] = nyc_as_one_county['PopFmle60-642010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale65-742010'] = nyc_as_one_county['PopMale65-742010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle65-742010'] = nyc_as_one_county['PopFmle65-742010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale75-842010'] = nyc_as_one_county['PopMale75-842010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle75-842010'] = nyc_as_one_county['PopFmle75-842010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopMale>842010'] = nyc_as_one_county['PopMale>842010']
county_demo.loc[county_demo.CountyName == 'New York', 'PopFmle>842010'] = nyc_as_one_county['PopFmle>842010']
county_demo.loc[county_demo.CountyName == 'New York', '3-YrMortalityAge55-64Years2015-17'] = nyc_as_one_county['3-YrMortalityAge55-
64Years2015-17']
county_demo.loc[county_demo.CountyName == 'New York', '3-YrMortalityAge65-74Years2015-17'] = nyc_as_one_county['3-YrMortalityAge65-
74Years2015-17']
county_demo.loc[county_demo.CountyName == 'New York', '3-YrMortalityAge75-84Years2015-17'] = nyc_as_one_county['3-YrMortalityAge75-
84Years2015-17']
county_demo.loc[county_demo.CountyName == 'New York', '3-YrMortalityAge85+Years2015-17'] = nyc_as_one_county['3-YrMortalityAge85+Ye
ars2015-17']
```

In [24]:
```
#Drop the other borough counties
county_demo = county_demo.drop(1827)
county_demo = county_demo.drop(1848)
county_demo = county_demo.drop(1865)
county_demo = county_demo.drop(1867)
```

In [25]:
```
confirmed_time_series = confirmed_time_series[confirmed_time_series.FIPS.isin(county_demo.countyFIPS)]
```

In [26]:
```
# Data Cleaning: Conclusion
print('Number of counties removed from county_demo:', (county_shape_raw[0] - county_demo.shape[0]))
print('Number of demographic variables removed from county_demo:', (county_shape_raw[1] - county_demo.shape[1]))
print('Number of counties removed from confirmed_time_series:', (c_ts_shape_raw[0] - confirmed_time_series.shape[0]))
```

```
Number of counties removed from county_demo: 590
Number of demographic variables removed from county_demo: 12
Number of counties removed from confirmed_time_series: 607
```

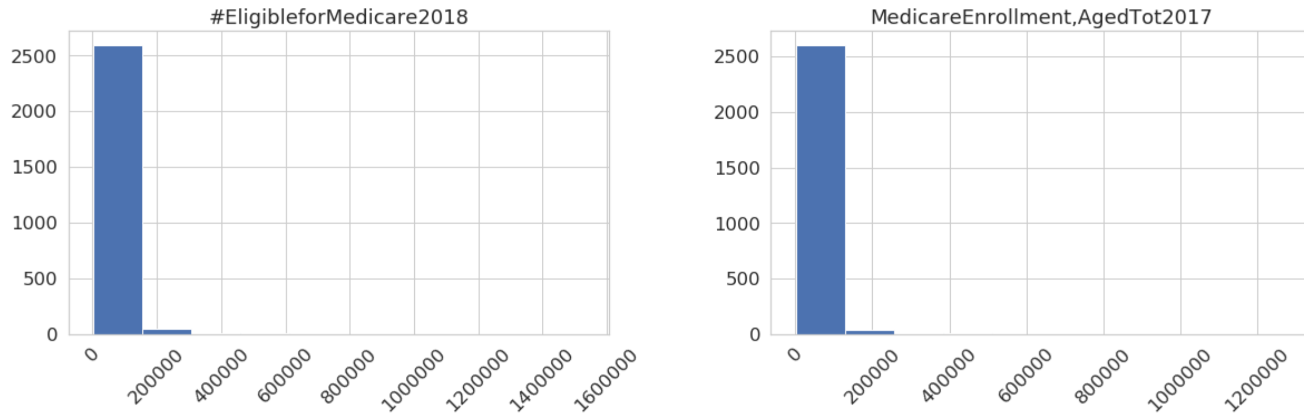
Methods:

In [27]:
```
#Data Pipeline

def pipeline(train_t_series, test_t_series, train_counties, test_counties):

    """Prepare time series and county demographic data for modeling."""
    # save original populaitons for weighting later in the model
    train_county_pops, test_county_pops = populations(train_counties, test_counties)
    train_t_series = time_series_transformation(train_t_series)
    test_t_series = time_series_transformation(test_t_series)
    train_counties = numeric_transformation(train_counties)
    test_counties = numeric_transformation(test_counties)
    train_counties, means, sds = standardize_train(train_counties)
    test_counties = standardize_test(test_counties, means, sds)

    return train_t_series, test_t_series, train_counties, test_counties, train_county_pops, test_county_pops
```

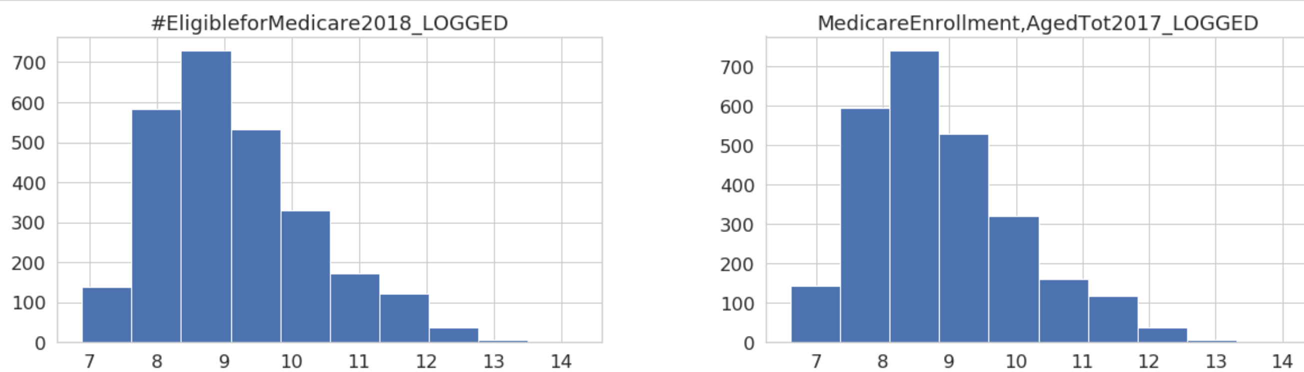
In [28]:
```
# Retrieve train and test county populations.

def populations(train_counties, test_counties):

    """Retrieve train and test county populations."""
    train_county_pops = train_counties.set_index('countyFIPS')['PopulationEstimate2018']
    test_county_pops = test_counties.set_index('countyFIPS')['PopulationEstimate2018']

    return train_county_pops, test_county_pops
```

In [29]:
```
# Transformations needed for fitting the model

def time_series_transformation(time_series):

    """Drop unused columns and transform time series data to usable format."""

    time_series = select_columns(time_series,'FIPS','1/22/20', '1/23/20',
    '1/24/20', '1/25/20', '1/26/20', '1/27/20', '1/28/20', '1/29/20',
    '1/30/20', '1/31/20', '2/1/20', '2/2/20', '2/3/20', '2/4/20','2/5/20',
    '2/6/20', '2/7/20', '2/8/20', '2/9/20', '2/10/20', '2/11/20','2/12/20',
    '2/13/20', '2/14/20', '2/15/20', '2/16/20', '2/17/20', '2/18/20',
    '2/19/20', '2/20/20', '2/21/20', '2/22/20', '2/23/20', '2/24/20',
    '2/25/20', '2/26/20', '2/27/20', '2/28/20', '2/29/20', '3/1/20',
    '3/2/20', '3/3/20', '3/4/20', '3/5/20', '3/6/20', '3/7/20', '3/8/20',
    '3/9/20', '3/10/20', '3/11/20', '3/12/20', '3/13/20', '3/14/20',
    '3/15/20', '3/16/20', '3/17/20', '3/18/20', '3/19/20', '3/20/20',
    '3/21/20', '3/22/20', '3/23/20', '3/24/20', '3/25/20', '3/26/20',
    '3/27/20', '3/28/20', '3/29/20', '3/30/20', '3/31/20', '4/1/20',
    '4/2/20', '4/3/20', '4/4/20', '4/5/20', '4/6/20', '4/7/20', '4/8/20',
    '4/9/20', '4/10/20', '4/11/20', '4/12/20', '4/13/20', '4/14/20',
    '4/15/20', '4/16/20', '4/17/20', '4/18/20', '4/19/20', '4/20/20',
    '4/21/20','4/22/20', '4/23/20', '4/24/20', '4/25/20', '4/26/20',
    '4/27/20', '4/28/20','4/29/20', '4/30/20', '5/1/20', '5/2/20',
    '5/3/20', '5/4/20', '5/5/20','5/6/20', '5/7/20')

    time_series = time_series.set_index('FIPS').transpose().cumsum()
    time_series = time_series.transpose().reset_index()

    return time_series

def numeric_transformation(data):

    """Extract predictive numerical indicators from county data."""
    #possibly remove population size? because its used later in weighting but, its probably fine

    data = select_columns(data, 'countyFIPS','lat','lon','PopulationEstimate2018','FracMale2017',
    'PopulationEstimate65+2017','PopulationDensityperSqMile2010','MedianAge2010',
    'DiabetesPercentage','HeartDiseaseMortality','StrokeMortality',
    'Smokers_Percentage', 'RespMortalityRate2014','SVIPercentile',
    '#Hospitals', '#FTEHospitalTotal2017','#HospParticipatinginNetwork2017',
    'MedicareEnrollment,AgedTot2017','#EligibleforMedicare2018', '#ICU_beds')

    #Log data with significant spread and skew

    data[['MedicareEnrollment,AgedTot2017', '#EligibleforMedicare2018']] = np.log(data[['MedicareEnrollment,AgedTot2017',
    '#EligibleforMedicare2018']])

    data = data.set_index('countyFIPS')

    return data

def standardize_train(numeric_data):

    """Return mean and sd numeric columns in train data to assign z-score to test data."""

    #preserve means, sds to calculate
    means, sds = numeric_data.mean(), numeric_data.std()

    #preserve lat, lon for reinsertion
    lat, lon = numeric_data['lat'], numeric_data['lon']
    numeric_data = (numeric_data - numeric_data.mean()) / numeric_data.std()
    numeric_data['lat'], numeric_data['lon'] = lat, lon

    return numeric_data, means, sds

def standardize_test(numeric_data, means, sds):

    """Use column means and sds to caculate test county z-scores."""
    #preserve lat, lon for reinsertion
    lat, lon = numeric_data['lat'], numeric_data['lon']
    numeric_data = (numeric_data - means) / sds
    numeric_data['lat'], numeric_data['lon'] = lat, lon

    return numeric_data
```

In [30]:
```
# Justification for logging 'MedicareEnrollment,AgedTot2017' and '#EligibleforMedicare2018' columns:
# First two graphs show the initial distributions of both features,
# Second two graphs show the distributions of the logged features.

# These first two graphs are not extremely useful for our model due to their concentrated distribution.

county_copy = county_demo.copy()

county_copy[['MedicareEnrollment,AgedTot2017', '#EligibleforMedicare2018']].hist(figsize = (20, 5), xrot = 45);
```



In [31]:
```
# These second two graphs are normalized by the log and become much more useful for our model.

county_copy[['MedicareEnrollment,AgedTot2017_LOGGED', '#EligibleforMedicare2018_LOGGED']] = np.log(county_copy[[
    'MedicareEnrollment,AgedTot2017', '#EligibleforMedicare2018']])
county_copy[['MedicareEnrollment,AgedTot2017_LOGGED', '#EligibleforMedicare2018_LOGGED']].hist(figsize = (20, 5));
```



In [32]:
```
# A function that finds the n most similar counties to each test county

def similar_counties(train_counties, test_counties, n):

    """Return dictionary of n most similar counties to each test county"""

    train_geo = train_counties[['lat', 'lon']]
    test_geo = test_counties[['lat', 'lon']]
    train_no_geo = train_counties.drop(['lat', 'lon'], axis = 1)
    test_no_geo = test_counties.drop(['lat', 'lon'], axis = 1)

    # matrix of haversine distance between geographical locations
    spherical_dist_matrix = dist_matrix(train_geo, test_geo, geo = True)

    # matrix of euclidian distance between standardized values of numeric columns
    euclid_matrix = dist_matrix(train_no_geo, test_no_geo)

    # matrix of similarity scores - best tradeoff between counties demographically similar and are
    # geopraphically close

    similarity_matrix = (euclid_matrix)**2 + np.sqrt(spherical_dist_matrix)
    similarity_dict = most_similar(similarity_matrix, n)

    return similarity_dict
```

In [33]:
```
# Two functions for creating a distance matrix: Euclidean and Haversine matrices

def dist_matrix(train, test, geo = False):

    """Create matrix of distance values between each county in test and train set."""
    """geo = False: calculate the euclidian distance between standardized values of numeric columns."""
    """geo = True: calculate the haversine distance between geographical distance."""

    train_t = train.transpose()
    test_t = test.transpose()
    arr = np.zeros(shape = (len(train_t.columns),len(test_t.columns)))

    for i, train_county in enumerate(train_t.columns):
        for j, test_county in enumerate(test_t.columns):
            if geo == True:
                arr[i][j] = spherical_dist(train_t[train_county].lat, test_t[test_county].lat,
                                        train_t[train_county].lon, test_t[test_county].lon)
            else:
                arr[i][j] = np.sqrt(sum((train_t[train_county] - test_t[test_county])**2))
    dist_matrix = pd.DataFrame(arr, columns = test_t.columns, index = train_t.columns)

    return dist_matrix

def spherical_dist(lat1, lat2, lon1, lon2):

    """Calculate haversine distance in miles between two locations"""

    R = 3958.8 #spherical radius of earth, in miles
    phi_1, phi_2 = lat1 * np.pi/180, lat2 * np.pi/180
    del_phi = (lat1 - lat2) * np.pi/180
    del_lam = (lon1 - lon2) * np.pi/180
    a = np.sin(del_phi/2) * np.sin(del_phi/2) + np.cos(phi_1) * np.cos(phi_2) * np.sin(del_lam/2) * np.sin(del_lam/2)
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    d = R * c
    #possible return log of distance if overcompensating

    return d

def most_similar(similarity_m, n):

    """Return the n most similar counties to each test county from similarity matrix."""

    county_dict = {}
    for test_county in similarity_m.columns:
        county_dict[test_county] = similarity_m[test_county].nsmallest(n).index.tolist()

    return county_dict
```

In [34]:
```
# Use similar counties to find best logistic growth model parameters for each test county

def best_params(similar_dict, train_t_series, n):

    """Use similar counties to find best logistic growth model parameters for each test county"""

    test_counties = similar_dict.keys()
    sum_sq_errors = []
    parameters = []
    y_hat = []
    model_param_dict = {}

    for fips in test_counties:
        similar_fips = similar_dict.get(fips)
        for N in np.arange(1, len(similar_fips) + 1):
            n_sim_fips = similar_fips[:N]
            if isinstance(n_sim_fips, int):
                n_sim_flips = [n_sim_fips]

            w = weights(fips, n_sim_fips)
            # n similar counties in n_sim_fips, 107 days in the time series data

            # y : n X 108 (fips included) sub data frame of train_t_series
            y = train_t_series.loc[train_t_series['FIPS'].isin(n_sim_fips)]
            # drop the fips, now y is n X 107
            y = y.drop(columns = ['FIPS'])
            # time series from 1/22/20 - 5/7/20, with 1/22/20 zerod to facilate numpy operations

            x = np.arange(len(y.columns))
            # multiply the confirmed cases each day by the weights
            y = w @ y.to_numpy()
            y_hat.append(y)

            # using similar counties to as starting parameter estimates
            a, b, c = logistic_growth_model(x, y)

            #err = cross_validate(x, y)
            err = sum_sqr_residuals(y, logistic_func(x, a, b, c))
            sum_sq_errors.append(err)
            parameters.append((a, b, c))

        min_index = np.argmin(sum_sq_errors)
        best_similar = min_index + 1
        best_err = sum_sq_errors[min_index]
        best_params = parameters[min_index]
        best_y_hat = y_hat[min_index]
        model_param_dict[fips] = (best_params, best_err, best_y_hat, best_similar)
        sum_sq_errors.clear()
        parameters.clear()
        y_hat.clear()

    return model_param_dict
```

In [35]:
```
def weights(fips, similar_fips):

    """Use similarity rank and population difference to determine weighting of each similar county."""

    num_similar = len(similar_fips)
    raw_weights = np.zeros(num_similar)
    population_weights = np.zeros(num_similar)

    for i in np.arange(num_similar):
        raw_weights[i] = 1 / (i + 1)
        population_weights[i] = abs(train_county_pops.loc[similar_fips[i]] - test_county_pops.loc[fips])

    weights = raw_weights / (population_weights + 1)
    weights = weights / sum(weights)
    return weights
```

In [36]:
```
# Initialize and train model

import scipy.optimize as optim

def logistic_growth_model(x, y):

    """Model logistic growth of coronavirus cases."""

    # parameters:
        # a: constant
        # b: rate of transmission
        # c: maximum capacity for y
        # p0: (a, b ,c)

    # constant, here as the mean number of cases on day 0

    a = y[0]

    # Early studies show the median R0 value of coronavirus as 5.7, with 95% CI 3.8-8.9
    # Source: https://wwwnc.cdc.gov/eid/article/26/7/20-0282_article
    b = .057

    # maximum capacity, here as maximum of test county populations
    c = test_county_pops.max()

    p0 = (a, b, c)

    # lower bound 0, corresponding upper bounds for a, b, c
    bounds = (0, [100000000., 5., 1000000000000000.])

    #scipy non linear least squares optimization gives values for a, b, & c that minimize the least square errors
    (a, b, c), cov = optim.curve_fit(logistic_func, x, y, bounds=bounds, p0=p0)

    return a, b, c
```

In [37]:
```
# Logistic function to fit model.

def logistic_func(t, a, b, c):

    """Logistic function to fit model."""

    return c / (1 + a * np.exp(-b * t))
```

In [38]:
```
# Error function for logistic growth model.

def sum_sqr_residuals(y, yhat):

    """Error function for logistic growth model."""

    return np.sum((y - yhat)**2)
```

In [39]:
```
# Train / test split to get our training and test series data

from sklearn.model_selection import train_test_split

train_t_series, test_t_series = train_test_split(confirmed_time_series, test_size=0.2, random_state = 83)

test_t_series = test_t_series.sample(n = 40, random_state = 301)

train_counties = county_demo.loc[county_demo['countyFIPS'].isin(train_t_series['FIPS'])]
test_counties = county_demo.loc[county_demo['countyFIPS'].isin(test_t_series['FIPS'])]
```

In [40]:
```
# Run our train / test series through our pipeline

train_t_series, test_t_series, train_counties, test_counties, train_county_pops, test_county_pops = pipeline(

train_t_series, test_t_series, train_counties, test_counties)
```

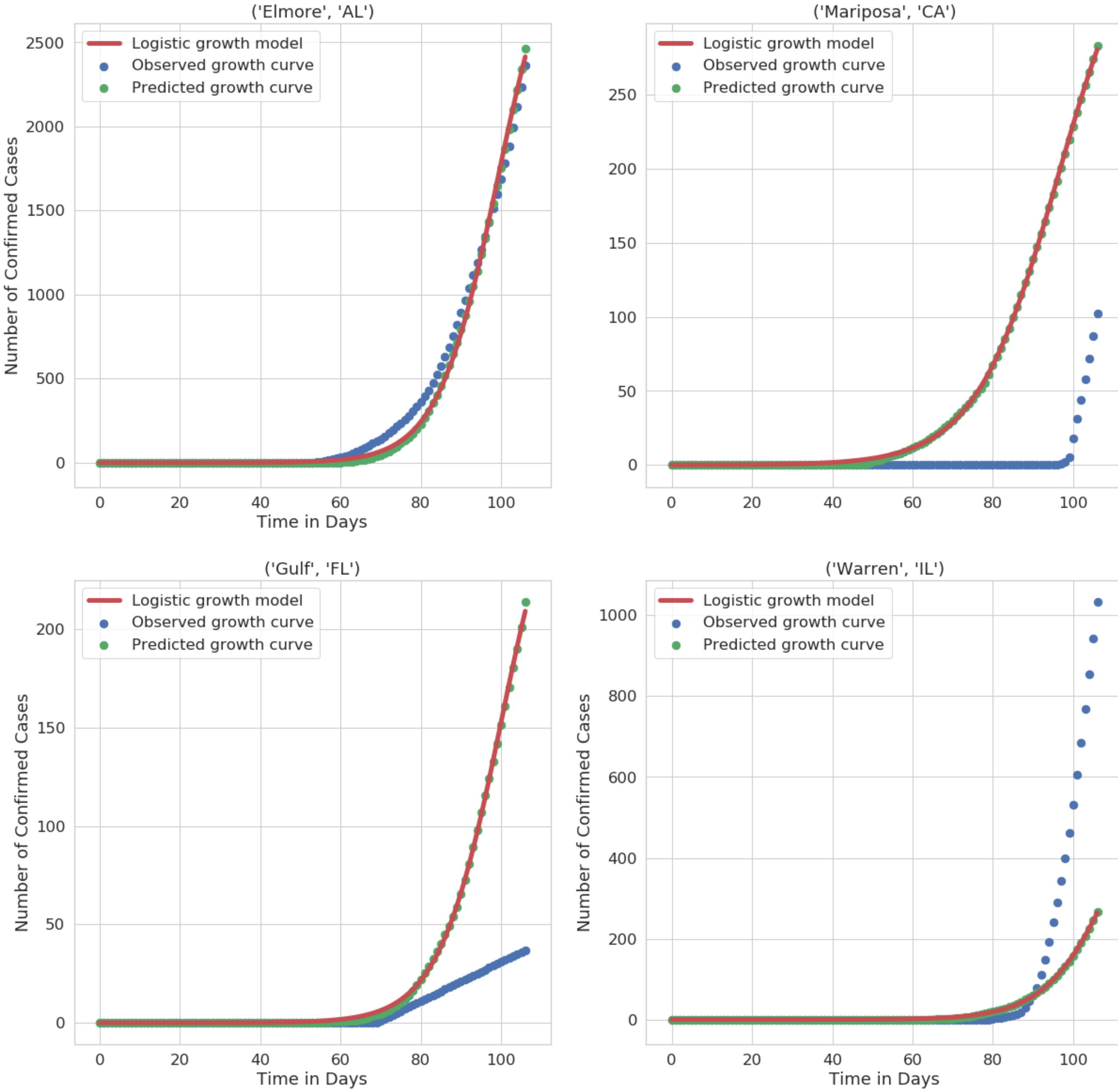
In [41]:
```
# Calculate most similar counties and find the optimal model parameters

test_index = test_counties.index.tolist()

sim_county_dict = similar_counties(train_counties, test_counties, 10)

models = best_params(sim_county_dict, train_t_series, 10)
```

In [42]:
```
# Calculate mean least squared error

sum_least_sq_error = 0

for fips in models.keys():
    sum_least_sq_error += models.get(fips)[1]

mean_least_sq_error = sum_least_sq_error/len(models.keys())
mean_least_sq_error
```

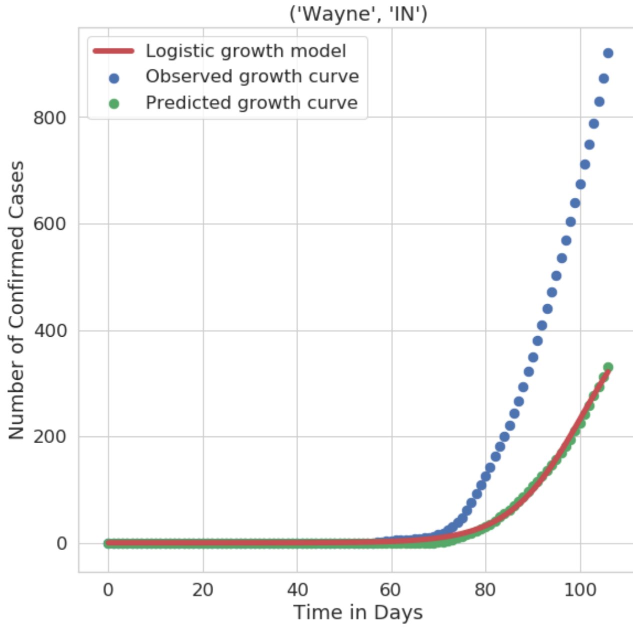
Out[42]:
```
391692.39310323267
```

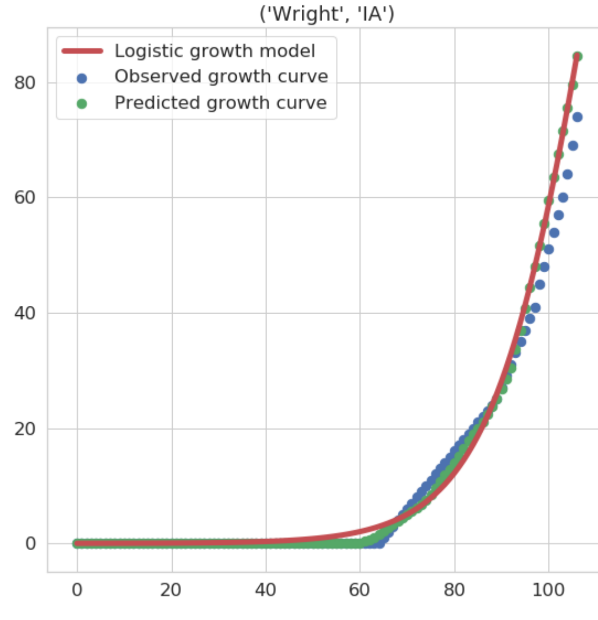
Model Testing

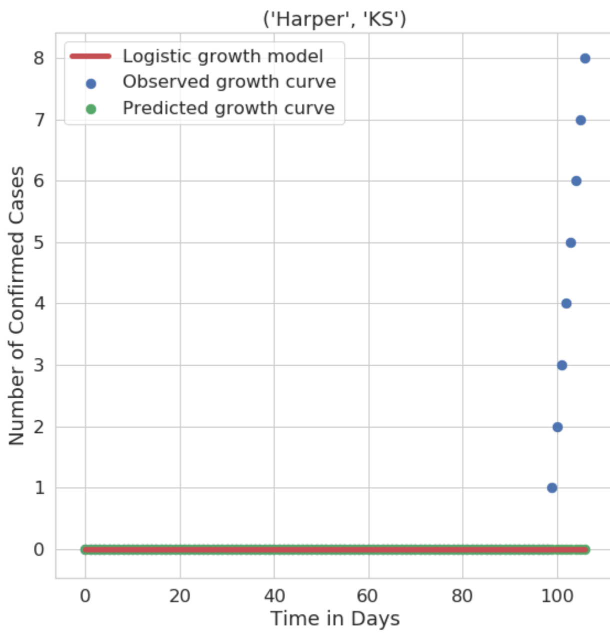
In [43]:
```
# This is what we used to investigate how our model was performing on all of our test counties

# Legend:
    # Logistic growth model: this is the curve that we are using as our model's estimate of the real curve
    # Observed growth curve: the county's actual confirmed time series
    # Predicted growth curve: our estimate of the actual cureve using similar counties

# First 4 of the test counties:

countyFIPS = [test_index[0], test_index[1], test_index[2], test_index[3]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

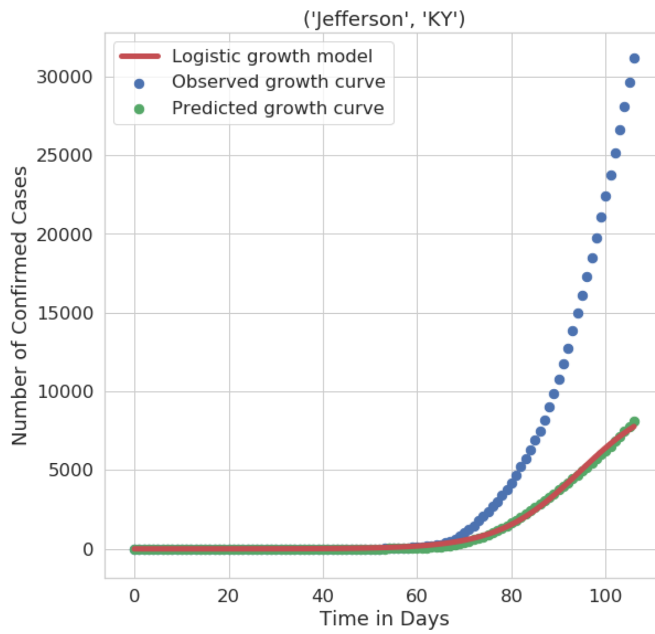

('Elmore', 'AL')
('Mariposa', 'CA')
('Gulf', 'FL')
('Warren', 'IL')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

In [44]:
```
# Second 4 of the test counties:

countyFIPS = [test_index[4], test_index[5], test_index[6], test_index[7]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

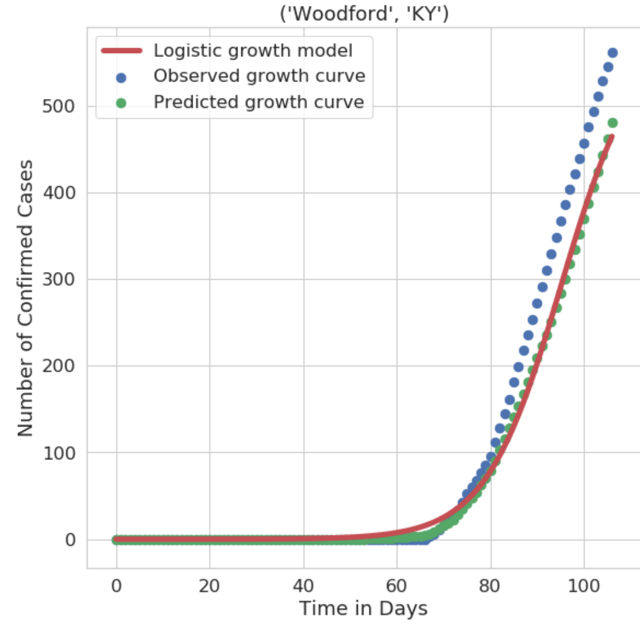

('Wayne', 'IN')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

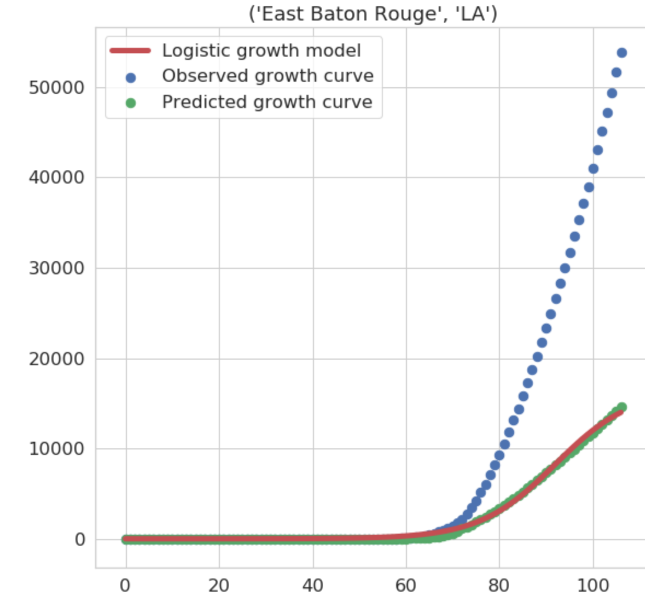

('Wright', 'IA')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days

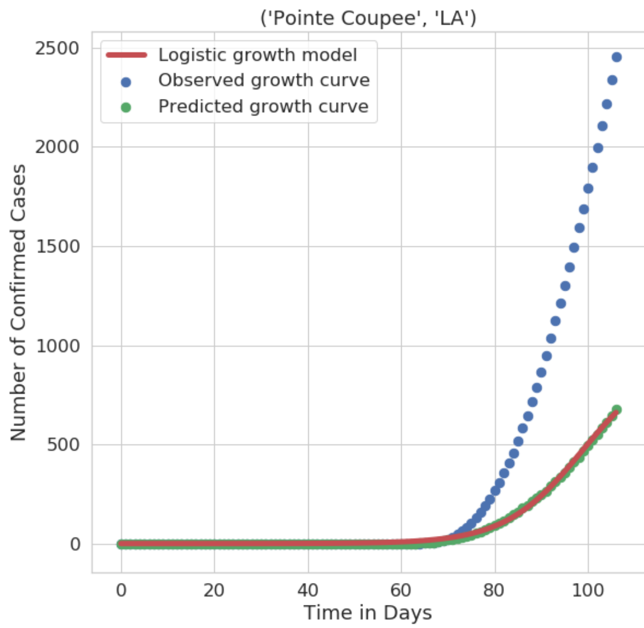

('Harper', 'KS')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

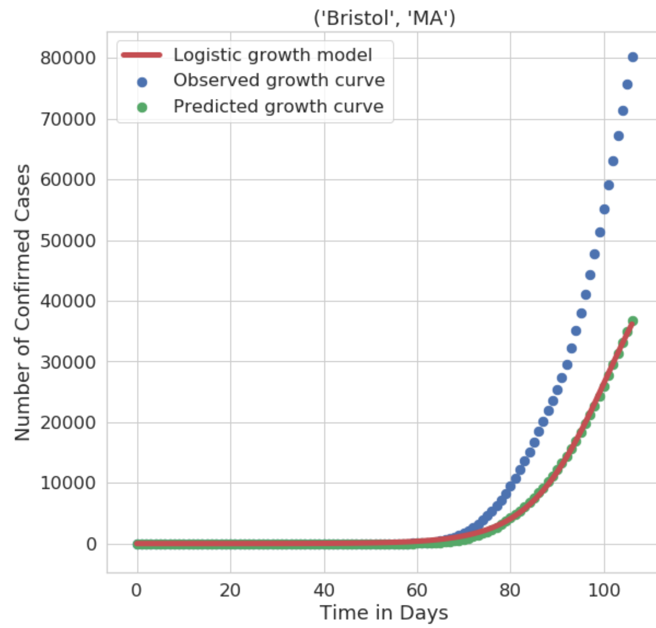

('Jefferson', 'KY')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

In [45]:
```
# Third 4 of the test counties:

countyFIPS = [test_index[8], test_index[9], test_index[10], test_index[11]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

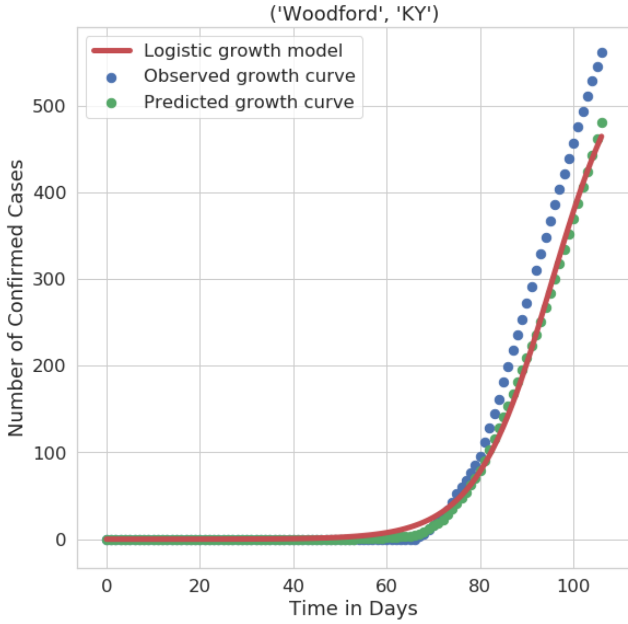

('Woodford', 'KY')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

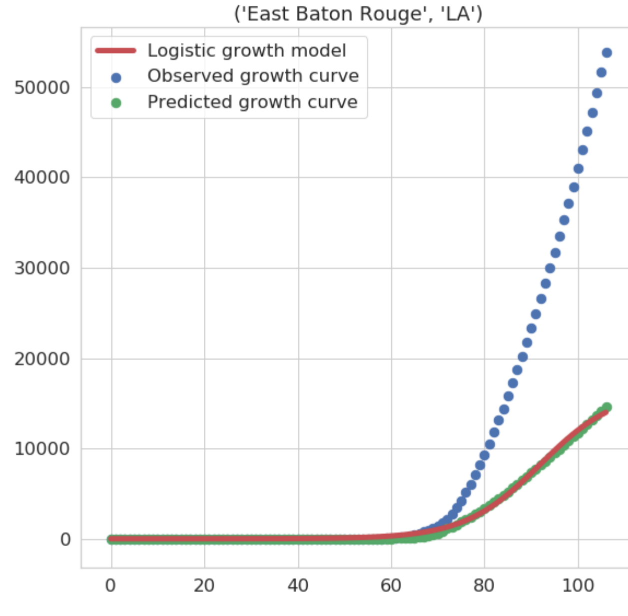

('East Baton Rouge', 'LA')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days

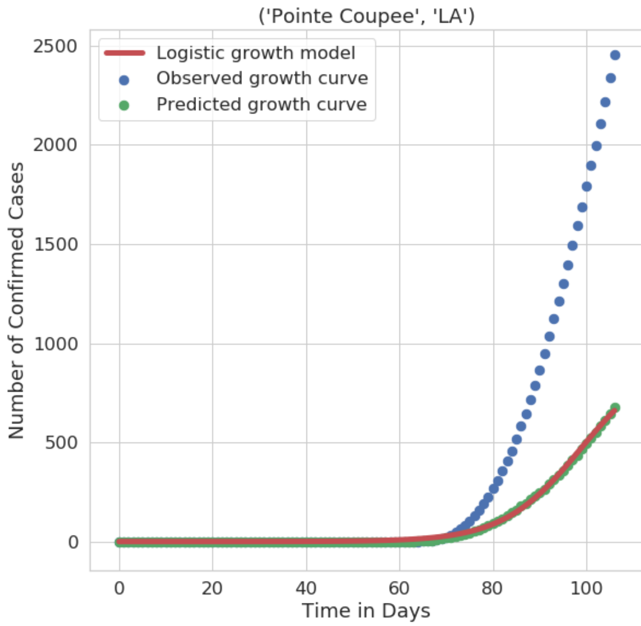

('Pointe Coupee', 'LA')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

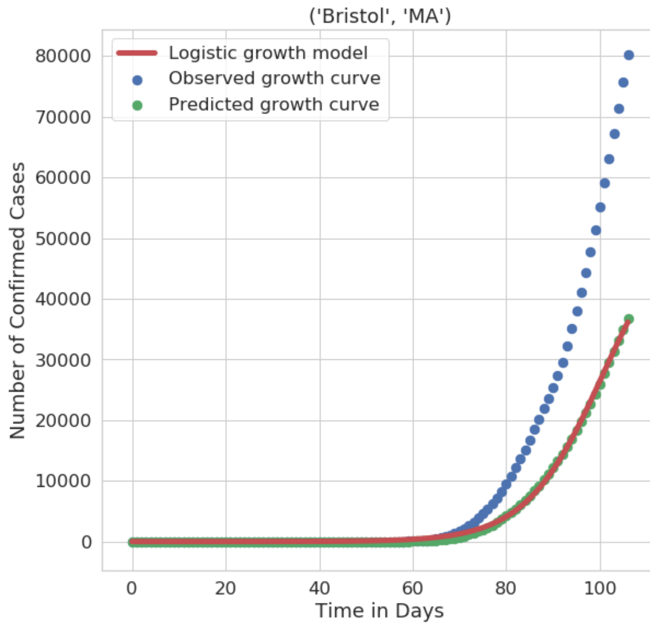

('Bristol', 'MA')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

In [46]:
```
# and so on...

countyFIPS = [test_index[8], test_index[9], test_index[10], test_index[11]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

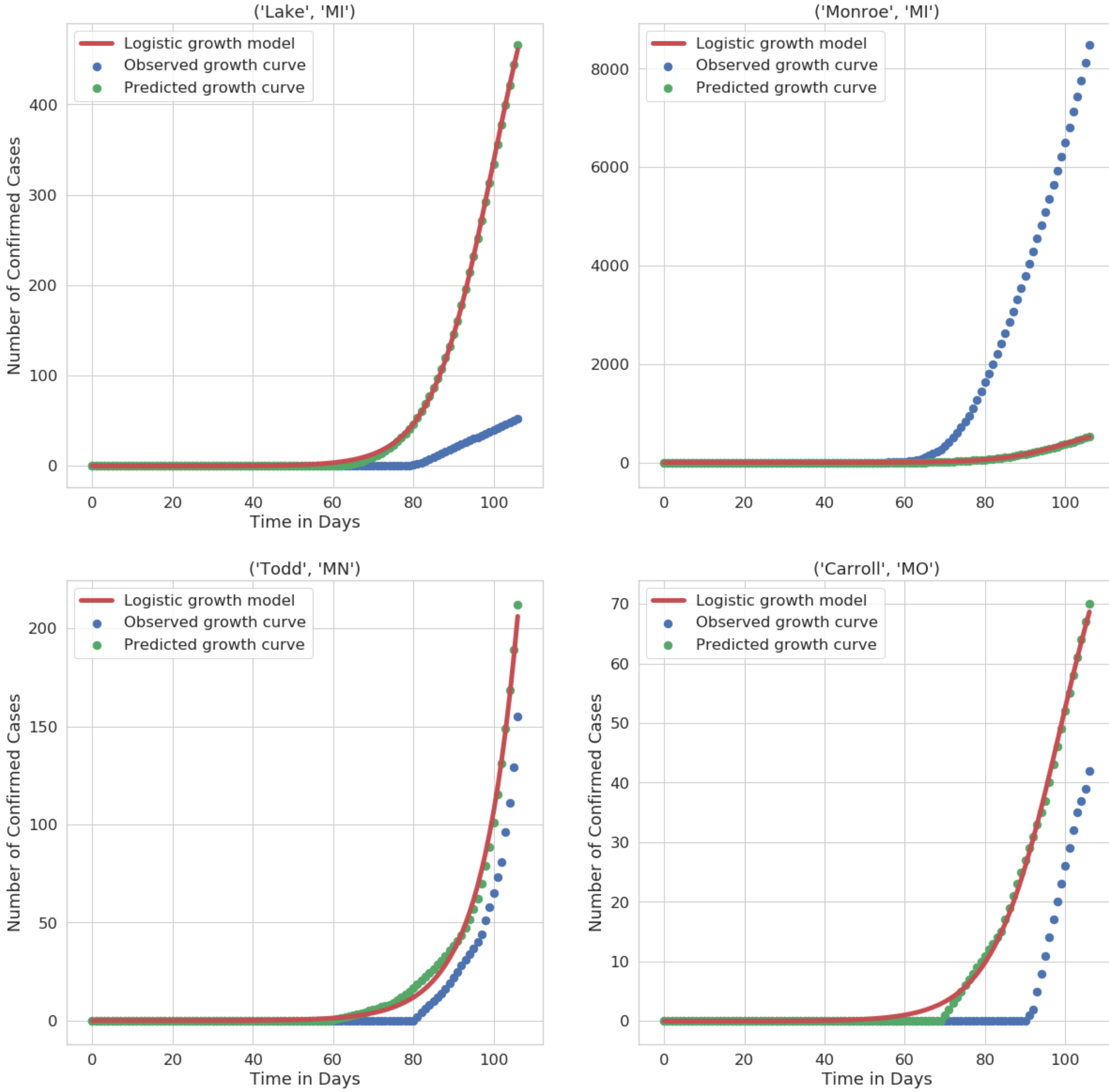

('Woodford', 'KY')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days


('East Baton Rouge', 'LA')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days

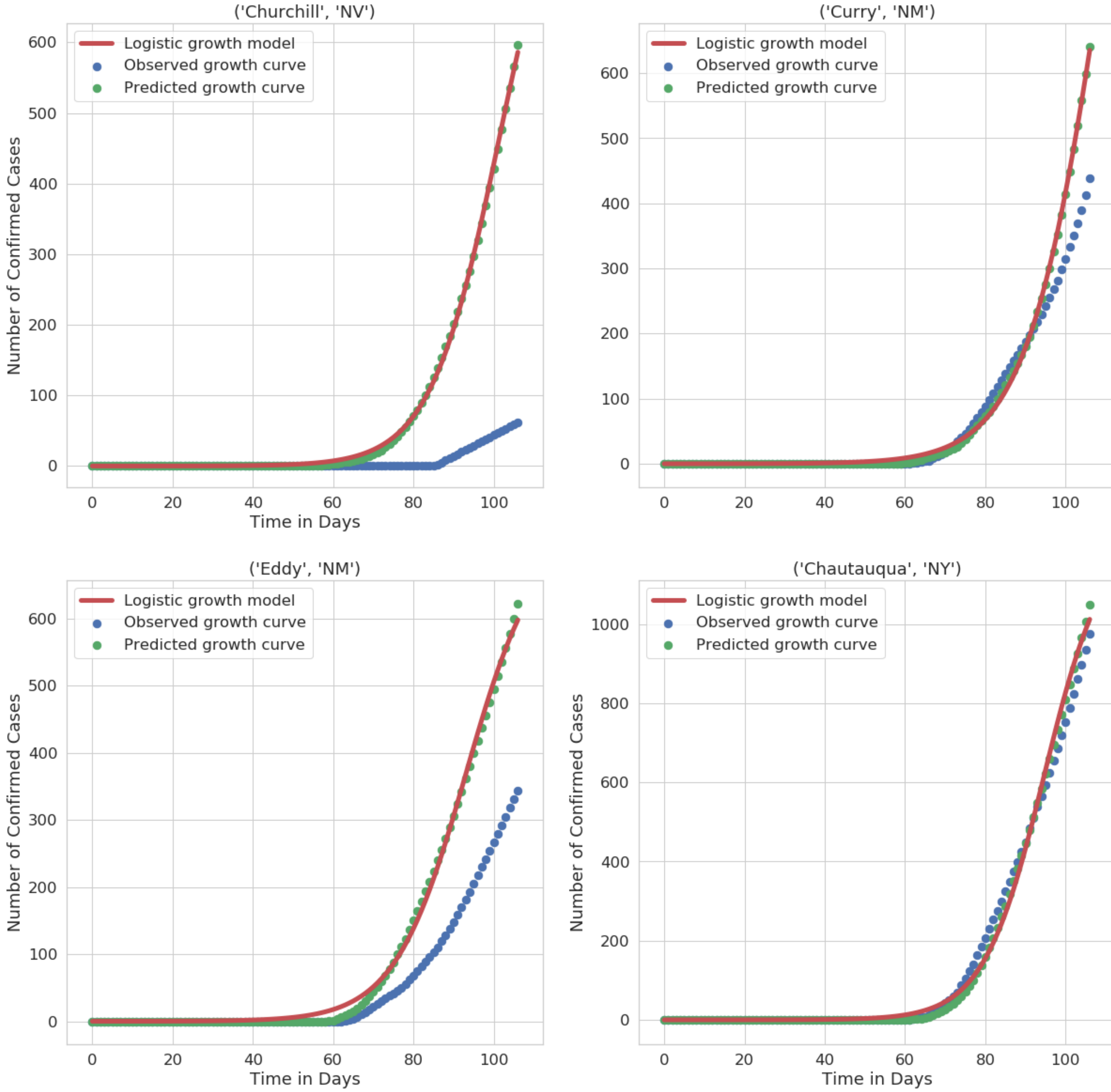

('Pointe Coupee', 'LA')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days


('Bristol', 'MA')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

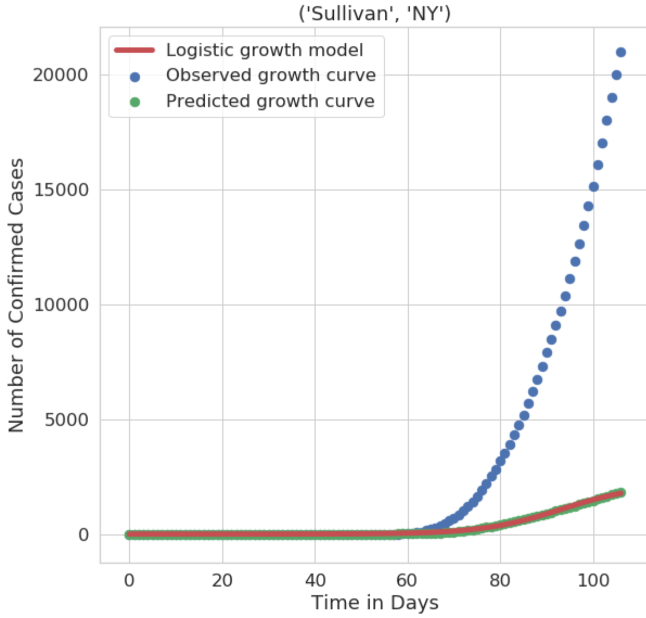
In [47]:
```
# and so on...

countyFIPS = [test_index[12], test_index[13], test_index[14], test_index[15]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

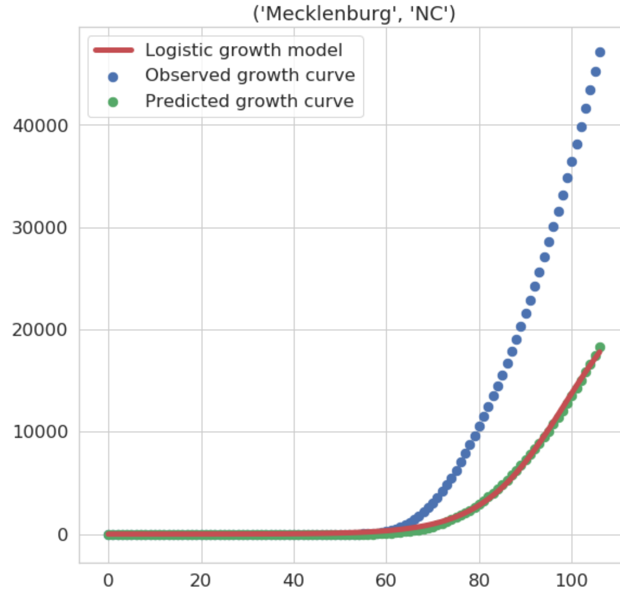

('Lake', 'MI')
('Monroe', 'MI')
('Todd', 'MN')
('Carroll', 'MO')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

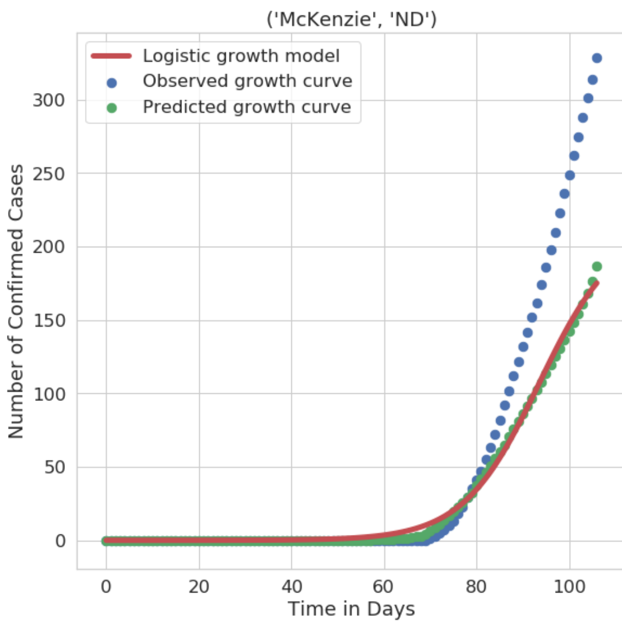
In [48]:
```
# and so on...

countyFIPS = [test_index[16], test_index[17], test_index[18], test_index[19]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

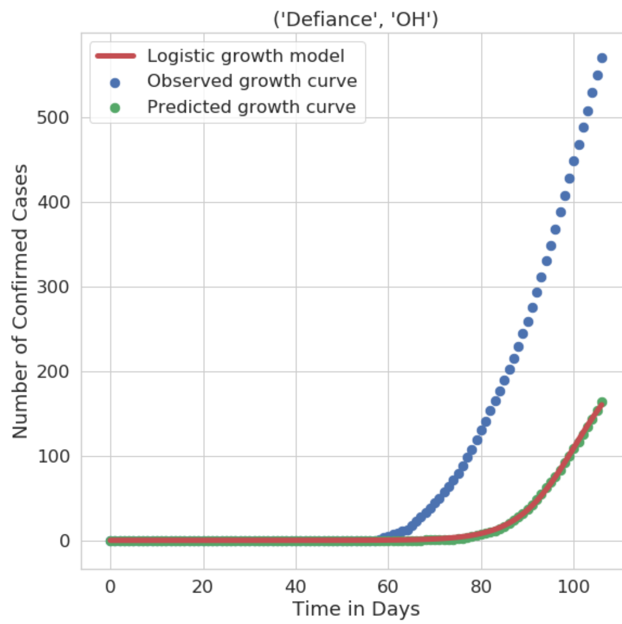

('Churchill', 'NV')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
('Curry', 'NM')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days
('Eddy', 'NM')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
('Chautauqua', 'NY')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

In [49]:
```
# and so on...

countyFIPS = [test_index[20], test_index[21], test_index[22], test_index[23]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

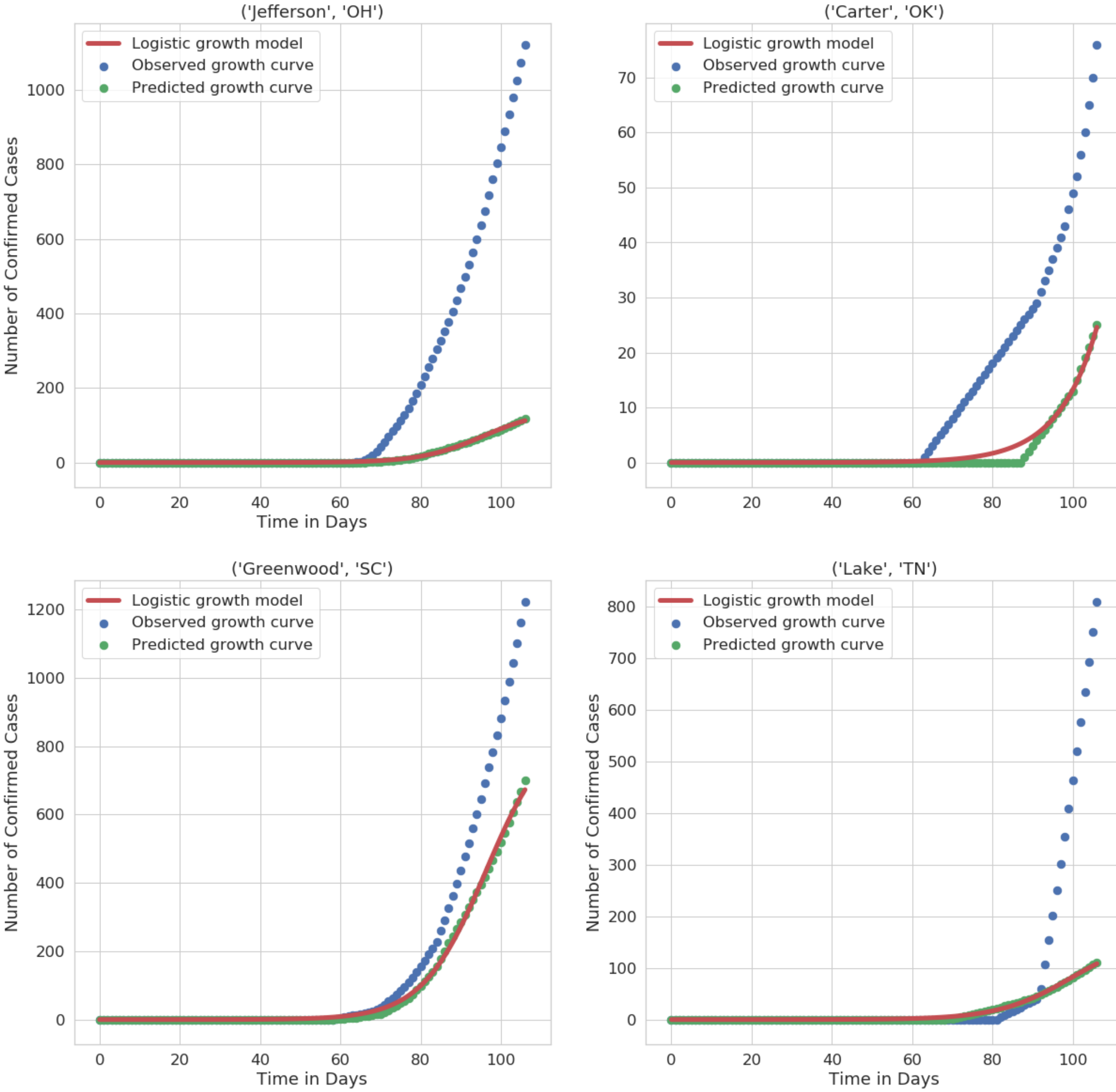

('Sullivan', 'NY')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days


('Mecklenburg', 'NC')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days

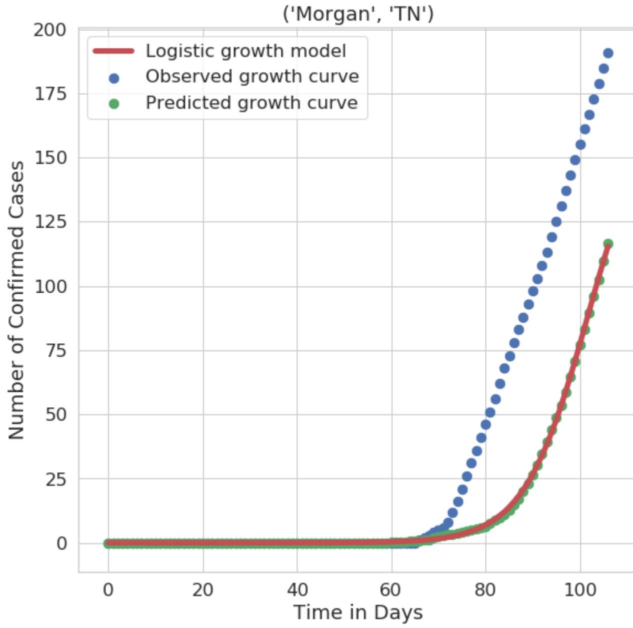

('McKenzie', 'ND')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

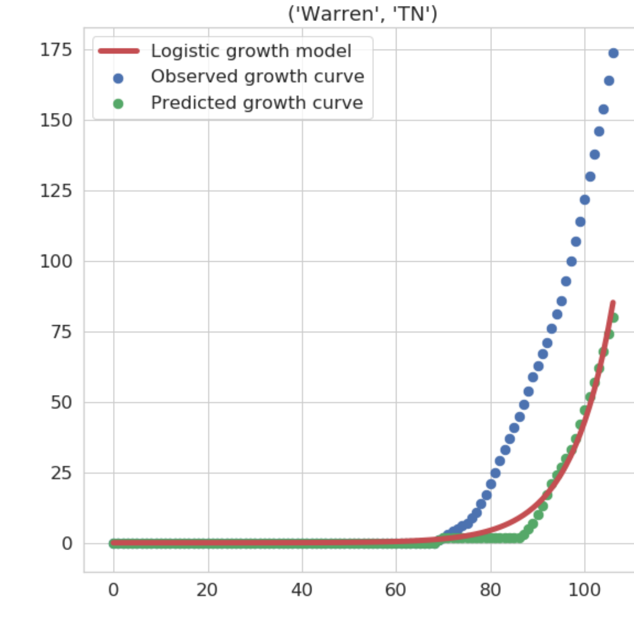

('Defiance', 'OH')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

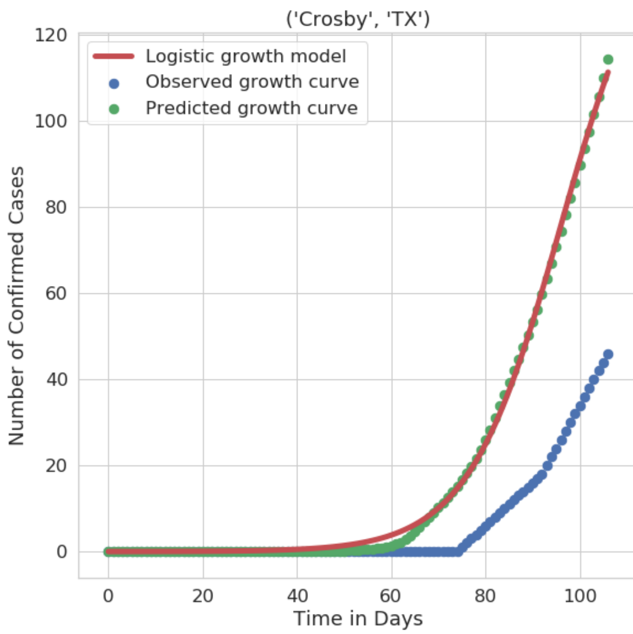
In [50]:
```
# and so on...

countyFIPS = [test_index[24], test_index[25], test_index[26], test_index[27]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

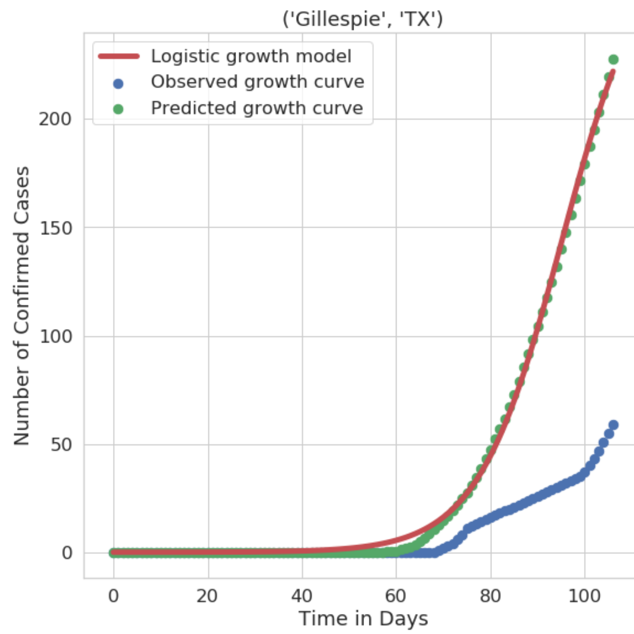

('Jefferson', 'OH')
('Carter', 'OK')
('Greenwood', 'SC')
('Lake', 'TN')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

In [51]:
```
# and so on...

countyFIPS = [test_index[28], test_index[29], test_index[30], test_index[31]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

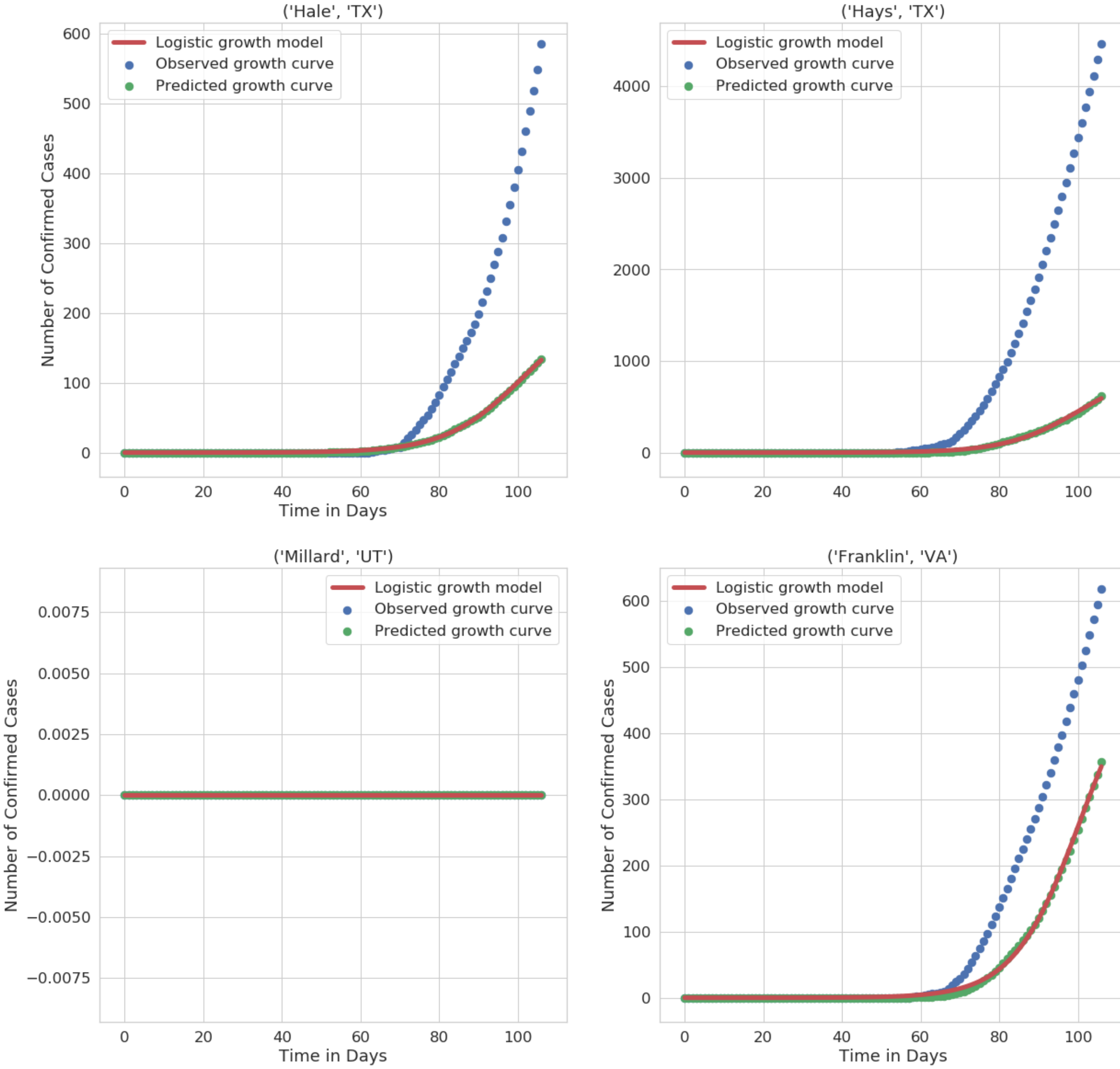

('Morgan', 'TN')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
0
25
50
75
100
125
150
175
200
0
20
40
60
80
100


('Warren', 'TN')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days
0
25
50
75
100
125
150
175
0
20
40
60
80
100

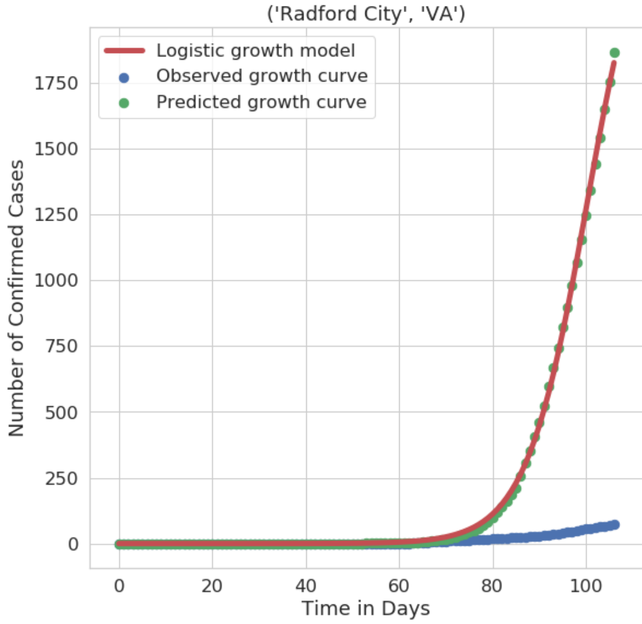

('Crosby', 'TX')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
0
20
40
60
80
100
120
0
20
40
60
80
100

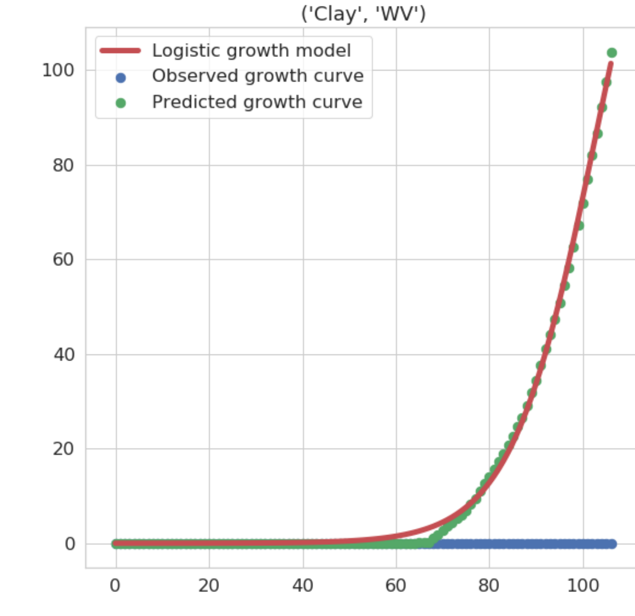

('Gillespie', 'TX')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
0
50
100
150
200
0
20
40
60
80
100

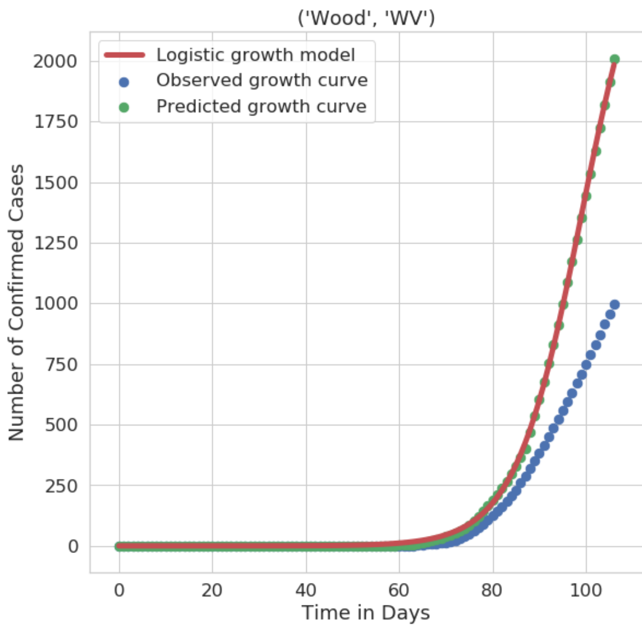
In [52]:
```
# and so on...

countyFIPS = [test_index[32], test_index[33], test_index[34], test_index[35]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```

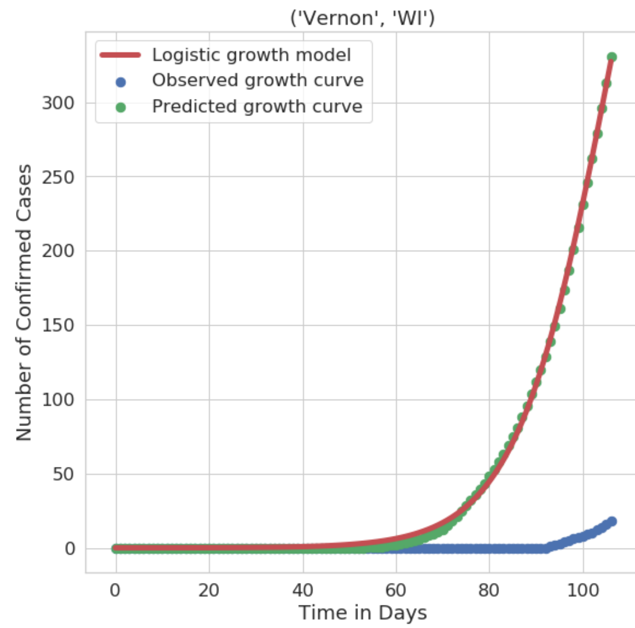

('Hale', 'TX')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
('Hays', 'TX')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days
('Millard', 'UT')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days
('Franklin', 'VA')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days

In [53]:
```
# and so on...

countyFIPS = [test_index[36], test_index[37], test_index[38], test_index[39]]

fig, axs = plt.subplots(2, 2, figsize = (20, 20))

county0 = county_demo[county_demo['countyFIPS'] == countyFIPS[0]].iloc[0]
y_0 = test_t_series[test_t_series['FIPS'] == countyFIPS[0]]
y_0 = y_0.drop(columns=['FIPS'])
x_0 = np.arange(len(y_0.columns))
y_0 = y_0.to_numpy()
params_0 = models.get(countyFIPS[0])[0]
err_0 = models.get(countyFIPS[0])[1]
y_hat_0 = models.get(countyFIPS[0])[2]

county1 = county_demo[county_demo['countyFIPS'] == countyFIPS[1]].iloc[0]
y_1 = test_t_series[test_t_series['FIPS'] == countyFIPS[1]]
y_1 = y_1.drop(columns=['FIPS'])
x_1 = np.arange(len(y_1.columns))
y_1 = y_1.to_numpy()
params_1 = models.get(countyFIPS[1])[0]
err_1 = models.get(countyFIPS[1])[1]
y_hat_1 = models.get(countyFIPS[1])[2]

county2 = county_demo[county_demo['countyFIPS'] == countyFIPS[2]].iloc[0]
y_2 = test_t_series[test_t_series['FIPS'] == countyFIPS[2]]
y_2 = y_2.drop(columns=['FIPS'])
x_2 = np.arange(len(y_2.columns))
y_2 = y_2.to_numpy()
params_2 = models.get(countyFIPS[2])[0]
err_2 = models.get(countyFIPS[2])[1]
y_hat_2 = models.get(countyFIPS[2])[2]

county3 = county_demo[county_demo['countyFIPS'] == countyFIPS[3]].iloc[0]
y_3 = test_t_series[test_t_series['FIPS'] == countyFIPS[3]]
y_3 = y_3.drop(columns=['FIPS'])
x_3 = np.arange(len(y_3.columns))
y_3 = y_3.to_numpy()
params_3 = models.get(countyFIPS[3])[0]
err_3 = models.get(countyFIPS[3])[1]
y_hat_3 = models.get(countyFIPS[3])[2]

# Plot Top Left
axs[0, 0].scatter(x_0, y_0, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 0].scatter(x_0, y_hat_0, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 0].plot(x_0, logistic_func(x_0, params_0[0], params_0[1], params_0[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 0].set_title((county0['CountyName'], county0['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 0].legend();

# Plot Top Right
axs[0, 1].scatter(x_1, y_1, c = 'b', s = 70, label = 'Observed growth curve')
axs[0, 1].scatter(x_1, y_hat_1, c = 'g', s = 70, label = 'Predicted growth curve')
axs[0, 1].plot(x_1, logistic_func(x_1, params_1[0], params_1[1], params_1[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[0, 1].set_title((county1['CountyName'], county1['StateName']))
axs[0, 0].set_xlabel('Time in Days')
axs[0, 0].set_ylabel('Number of Confirmed Cases')
axs[0, 1].legend();

# Plot Bottom Left
axs[1, 0].scatter(x_2, y_2, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 0].scatter(x_2, y_hat_2, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 0].plot(x_2, logistic_func(x_2, params_2[0], params_2[1], params_2[2]), c = 'r',  lw = 5,
               label = 'Logistic growth model')
axs[1, 0].set_title((county2['CountyName'], county2['StateName']))
axs[1, 0].set_xlabel('Time in Days')
axs[1, 0].set_ylabel('Number of Confirmed Cases')
axs[1, 0].legend();

# Plot Bottom Right
axs[1, 1].scatter(x_3, y_3, c = 'b', s = 70, label = 'Observed growth curve')
axs[1, 1].scatter(x_3, y_hat_3, c = 'g', s = 70, label = 'Predicted growth curve')
axs[1, 1].plot(x_3, logistic_func(x_3, params_3[0], params_3[1], params_3[2]), c = 'r', lw = 5,
               label = 'Logistic growth model')
axs[1, 1].set_title((county3['CountyName'], county3['StateName']))
axs[1, 1].set_xlabel('Time in Days')
axs[1, 1].set_ylabel('Number of Confirmed Cases')
axs[1, 1].legend();
```


('Radford City', 'VA')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days


('Clay', 'WV')
Logistic growth model
Observed growth curve
Predicted growth curve
Time in Days


('Wood', 'WV')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days


('Vernon', 'WI')
Logistic growth model
Observed growth curve
Predicted growth curve
Number of Confirmed Cases
Time in Days