COVID-19 Project

June 26, 2020

1 COVID-19 Notebook- Tracking The Virus

1.1 By David R McKenna

Because the COVID crisis is affecting the entire world and I want to be aware of what is happening without having to rely on the data being filtered through a political lens, I created this notebook. I hope you enjoy it and feel free to modify the code at the bottom to investigate your own states of interest.

1.1.1 Import the necessary libraries, packages and functions to be used.

[1]:
```
import numpy as np
import pandas as pd
from datetime import date
import matplotlib.pyplot as plt
import seaborn as sns
import requests, json
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from sklearn.linear_model import LinearRegression

import warnings
warnings.filterwarnings('ignore')
```

1.2 Import the latest data

1.2.1 The data is imported from the covidtracking.com website.

I create columns in the data frame for the Deaths per Case (DperP) and the Positive per Test as a percentage (PosPerTest). These will be used latter. I set the index to be the date as a datetime object. I then drop all the columns that are not used to speed things up a bit.

[2]:
```
url = 'https://covidtracking.com/api/v1/states/daily.json'

r = requests.get(url)

json_data = r.json()

df = pd.json_normalize(json_data)
```

```
### Add calculate various relations of the data and add them as columns

df['DperP'] = df['death']/df['positive']  # Add columns Deaths per Positive␣
↪case (DperP), Positive per Test (PosPerTest),

df['PosPerTest']= df['positiveIncrease']/df['totalTestResultsIncrease']*100  #␣
↪Positives per test in percentage

df['date'] = pd.to_datetime(df['date'], format = '%Y%m%d') #convert date to␣
↪datetime object and set as index
df.set_index('date')

# Drop all unused columns.

drop = ['pending',
        'hospitalizedCurrently', 'hospitalizedCumulative',␣
↪'onVentilatorCurrently',
        'onVentilatorCumulative','recovered', 'dataQualityGrade',␣
↪'lastUpdateEt',
        'dateModified','checkTimeEt', 'dateChecked','totalTestsViral',␣
↪'positiveTestsViral',
        'negativeTestsViral','positiveCasesViral', 'fips','posNeg','hash',␣
↪'commercialScore',
        'negativeRegularScore', 'negativeScore', 'positiveScore',␣
↪'score','grade']

df.drop(columns=drop, inplace=True)

#create date_ordinal column because it is sometimes earsier to refer to dates␣
↪as ordinals
df['date_ordinal'] = pd.to_datetime(df['date']).apply(lambda date: date.
↪toordinal())
#print('done')
```

1.3 Subset and clean data:

I selected UT first because that is where I live. There were some clear outliers so I replaced them with the average over the two days. It appears that there was a correction for an overcount or overreported value so I took the sum and divided by two = 12. This may not be the exact values for each day but preserves the integrity of the average over the corrected dates.

[3]:
```
UT = df[df['state']=='UT']
UT= UT[UT['date'] >= '2020-03-15' ]

A = UT['hospitalizedIncrease'].loc[lambda x: x==389].index
```

```
B = UT['hospitalizedIncrease'].loc[lambda x: x== -365].index
C = UT['positiveIncrease'].loc[lambda x: x== 0].index

UT.loc[A,'hospitalizedIncrease'] = 12
UT.loc[B,'hospitalizedIncrease'] = 12
UT.drop(C,inplace=True)
```

1.4 Setup global characteristics for plots.

Plots will have the basic seaborn appearance and will have a watermark with attributions. date_ticks overcomes the issues of plotting datetime indexed datasets, especially if there is analysis, fitting, or averaging, this also allows quick scaling of the date interval presented. This is useful as the duration of the virus increases rescaling the interval provides a fast and easy way to keep the plots readable.

[4]:
```
sns.set()

# create a watermark for the plots that declares my ownership and
# attributes where I got the data
def watermark(loc_x = 0.92, loc_y =0.15):
    """ Puts Property Of David McKenna on LRH corner of plots"""

    fig.text(loc_x,loc_y, 'Property of David McKenna \n data from:␣
 ↪covidtracking.com',
         fontsize=10, color='gray', rotation=270,
         ha='right', va='bottom', alpha=0.75)

    # Format the datetime x-axis tick marks
def date_ticks(interval = 7) :
    # Define the date format
    date_form = DateFormatter("%m-%d")
    ax.xaxis.set_major_formatter(date_form)

    # Ensure a major tick for each week using (interval=_)
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=interval))
    plt.xticks(rotation=30)

# Calculate the rolling averages of selected columns
def Roll_Avg(df, col, interval) :
    """Calculate the rolling averages of interval duration of columns in df"""
    length = len(interval)

    for n in range(length) :
        # make new col called roll_col_interval
        new_col = 'roll_' + str(col) + '_'+ str(interval[n])
```

```
        df[new_col] = df.loc[:,col].rolling(interval[n]).mean().shift(periods=␣
↪- interval[n])
```

1.5 Define the dates where UT changes their COVID status' from Red to Orange and Orange to Yellow.

I added Memorial Day and the start of Protests as well as 7-days after each (hashed-line), to visualize possible effects these events might have on case increases. my_annotate allows for flexible location and removal of text labels.

[5]:
```
# Define the dates of transitions by their ordinal date
OrangeDate = UT[UT['date']=='2020-04-28']['date_ordinal']
OrangeDate = int(OrangeDate)

YellowDate = UT[UT['date']=='2020-05-14']['date_ordinal']
YellowDate = int(YellowDate)

MemorialDay = UT[UT['date']=='2020-05-25']['date_ordinal']
MemorialDay = int(MemorialDay)

ProtestDate = UT[UT['date']=='2020-05-29']['date_ordinal']
ProtestDate = int(ProtestDate)

#Define a function that adds the vertical lines at the relevent dates
def my_annotate(text_loc, text=True) :
    """Setup the anotations of the plots- marking of significant dates"""

    ax.axvline(x=OrangeDate, color='tab:orange', linewidth=2,alpha=0.5)
    ax.axvline( x=YellowDate, color='gold', linewidth=2,alpha=0.5)
    ax.axvline( x=MemorialDay, color='tab:red', linewidth=1.5,alpha=0.5)
    ax.axvline( x=ProtestDate, color='tab:purple', linewidth=1.5,alpha=0.5)
    ax.axvline(x=OrangeDate + 7, color='tab:orange', linewidth=2, linestyle =␣
↪'--',alpha=0.5)
    ax.axvline( x=YellowDate + 7, color='gold', linewidth=2, linestyle =␣
↪'--',alpha=0.5)
    ax.axvline( x=ProtestDate + 7, color='tab:purple', linewidth=2, linestyle =␣
↪'--',alpha=0.5)

    if text==True:
        ax.annotate('Code Orange Date', (OrangeDate -␣
↪2,text_loc),color='black',rotation=90,fontsize=13,alpha=0.5)
        ax.annotate('Code Yellow Date', (YellowDate -␣
↪2,text_loc),color='black',rotation=90,fontsize=13,alpha=0.5)
        ax.annotate('Memorial Day', (MemorialDay - 2␣
↪,text_loc),color='black',rotation=90,fontsize=13,alpha=0.5)
        ax.annotate('Protest Start Date', (ProtestDate - 2␣
↪,text_loc),color='black',rotation=90,fontsize=13,alpha=0.5)
```

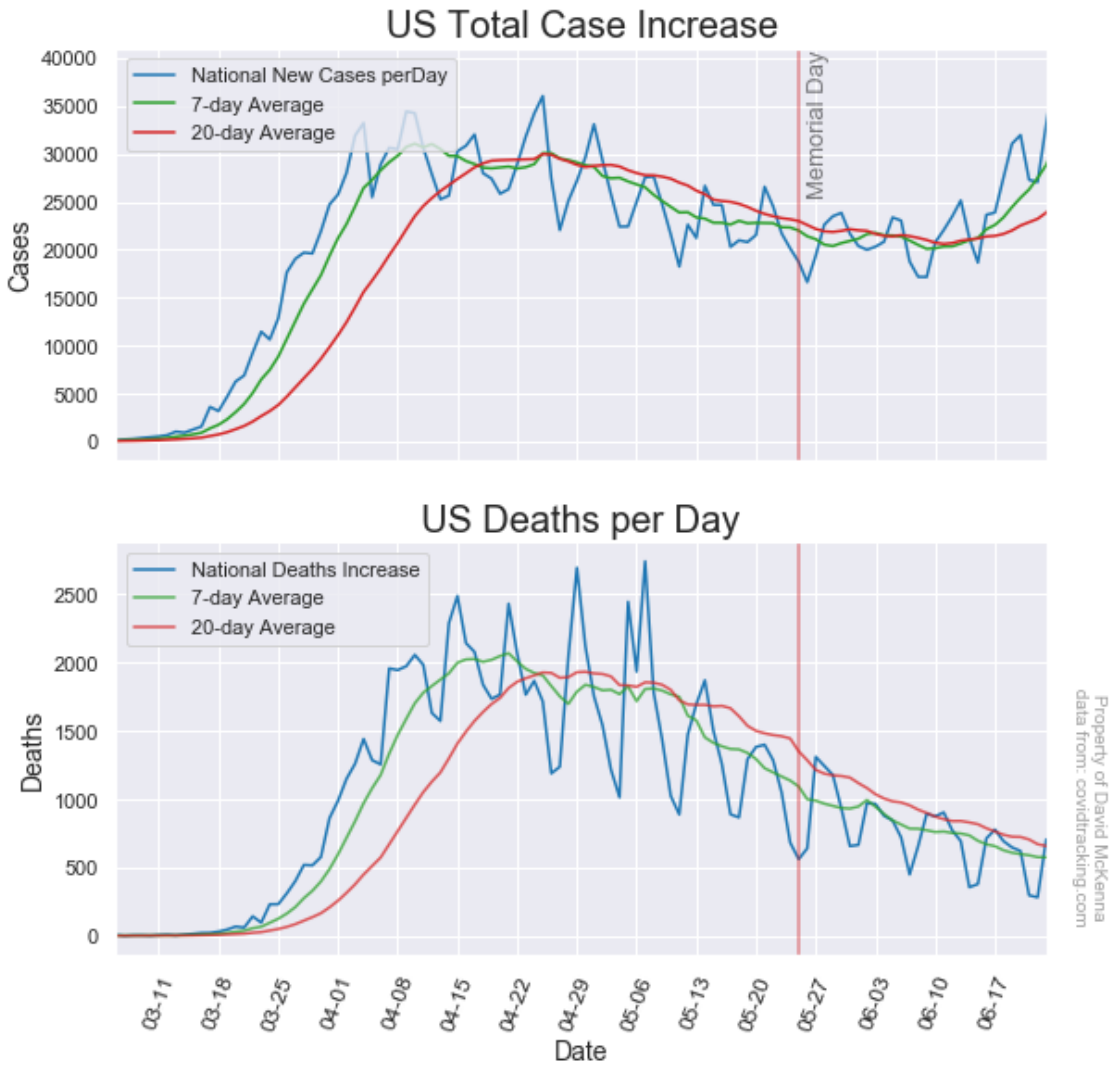
```
        ax.annotate('Code Orange + 7-Days', (OrangeDate␣
↪+5,text_loc),color='black',rotation=90,fontsize=13,alpha=0.5)
```

1.6 The US total Case and Deaths per day:

For a quick look at what is happening nationwide.

[6]:
```
# Plot the national case increase with 20-day average
sns.set()


interval = 7
fig, ax = plt.subplots(2,1,sharex=True,figsize = (9,9))
end = df.date.max() + pd.DateOffset(-2)
watermark(loc_x = 0.95)

df.groupby('date')['positiveIncrease'].sum().plot(ax=ax[0],label='National New␣
↪Cases perDay',
                                                  color='tab:blue')
df.groupby('date')['positiveIncrease'].sum().rolling(7).mean().
↪plot(ax=ax[0],label='7-day Average',
                                                                  ␣
↪color='tab:green')
df.groupby('date')['positiveIncrease'].sum().rolling(20).mean().
↪plot(ax=ax[0],label='20-day Average',
                                                                  ␣
↪color='tab:red')

ax[0].axvline(x='2020-05-25', color='tab:red', linewidth=1.5, alpha=0.5)
ax[0].annotate('Memorial Day', ('2020-05-26' ␣
↪,26000),color='black',rotation=90,fontsize=13,alpha=0.5)
date_form = DateFormatter("%m-%d")
ax[0].xaxis.set_major_formatter(date_form)

# Ensure a major tick for each week using (interval=_)
ax[0].xaxis.set_major_locator(mdates.DayLocator(interval=interval))
ax[0].legend(loc='upper left')
ax[0].set_title('US Total Case Increase', fontdict={'fontsize':20})
ax[0].set_xticklabels(ax[0].get_xticklabels(),rotation=30)
ax[0].set_ylabel('Cases', fontdict={'fontsize':14})
today = max(df['date'])

df.groupby('date')['deathIncrease'].sum().plot(ax=ax[1],label='National Deaths␣
↪Increase',color='tab:blue')
df.groupby('date')['deathIncrease'].sum().rolling(7).mean().
↪plot(ax=ax[1],label='7-day Average',
```

```
 ↪color='tab:green',alpha=0.75)
df.groupby('date')['deathIncrease'].sum().rolling(20).mean().
 ↪plot(ax=ax[1],label='20-day Average',
                                                                color='tab:
 ↪red',alpha=0.75)

ax[1].xaxis.set_major_formatter(date_form)

# Ensure a major tick for each week using (interval=_)
ax[1].xaxis.set_major_locator(mdates.DayLocator(interval=interval))

ax[1].axvline(x='2020-05-25', color='tab:red', linewidth=1.5, alpha=0.5)

ax[1].legend(loc='upper left')
ax[0].set_xlim('2020-03-06',end)
ax[1].set_title('US Deaths per Day', fontdict={'fontsize':20})
plt.setp( ax[1].xaxis.get_majorticklabels(), rotation=70,fontsize=12 )
ax[1].set_ylabel('Deaths', fontdict={'fontsize':14})
ax[1].set_xlabel('Date', fontdict={'fontsize':14})
plt.savefig('national.png')

plt.show()
```

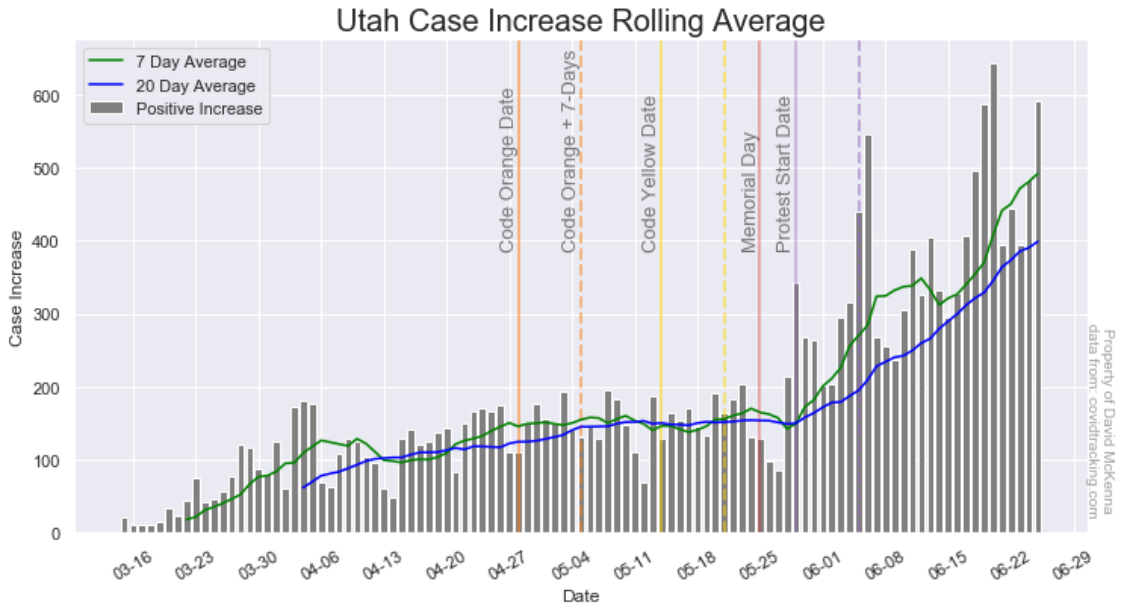


1.7 Plot the Daily Case Increase with 7 & 20 day rolling average

I chose to do a rolling average at 7 & 20 days to smooth out the natural oscillations in the data that may caused by difficulties of reporting during the weekend and subsequently over reporting during the following couple days. The oscillations are on the order of 3 days so the 7 day average is sufficient to smooth them out as well as be sensitive enough to react quickly to sudden spikes. The 20 day average gives a more general shape of the curve.

[7]:
```
# Calculate the 7 & 20 day rolling averages:
Roll_Avg(UT, 'positiveIncrease', [7,20])

#define location for the text labels on annotations
text_loc = max(UT['positiveIncrease'])-250
```

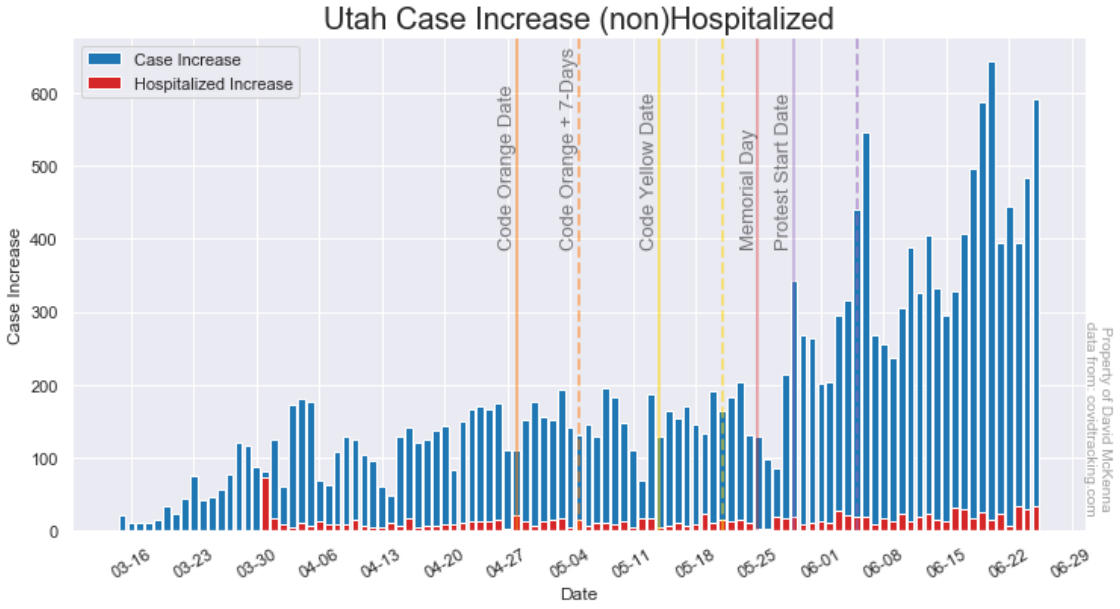
```
# Create barplot of the case increase with rolling averages overlayed.
fig, ax = plt.subplots(figsize = (12,6))
watermark() #add watermark with attributions
my_annotate(text_loc) #add annotated lines to mark relevent dates
plt.bar(UT['date'], UT['positiveIncrease'], label='Positive␣
 ↪Increase',color='grey')
plt.plot(UT['date'], UT.roll_positiveIncrease_7,
         label='7 Day Average', color='green')
plt.plot(UT['date'], UT.roll_positiveIncrease_20,
         label='20 Day Average', color='blue')
date_ticks() #format the dates on x-axis
plt.legend(loc='upper left')
plt.title('Utah Case Increase Rolling Average', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Case Increase', fontdict={'fontsize':12})
plt.axis('auto')

#plt.savefig('Utah_Increase_Rolling_Avg.png')
plt.show()
```



1.8 Plot the Daily Increase in cases as well as hospitalizations

The daily increase in cases doesn't tell the whole story. The true danger of the pandemic is when the number of hospitalizations exceeds the capacity of the local healthcare system. I add to the case increase plot the fraction of those that are hospitalized. Because hospitalization often occurs after diagnosis, the daily hospitalizations may lag the case increase.

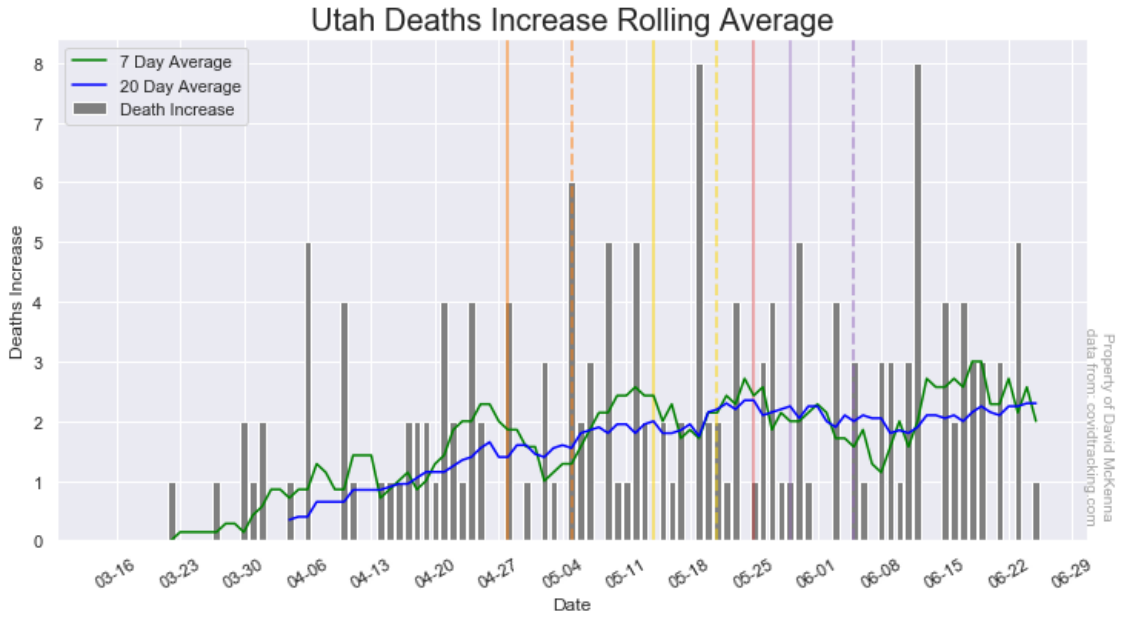
[8]:
```
#Create overlapping bar plots of case increase and hospitalizations
fig, ax = plt.subplots(figsize = (12,6))
watermark() #add atribution watermark

ax.bar(UT['date'], UT['positiveIncrease'], label='Case Increase',color='tab:
 ↪blue')
ax.bar(UT['date'], UT['hospitalizedIncrease'],
       label='Hospitalized Increase',color='tab:red')

ax.legend(loc='upper left')
my_annotate(text_loc) # add annotated lines of relevent dates

date_ticks() #format the dates on x-axis

plt.title('Utah Case Increase (non)Hospitalized', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Case Increase', fontdict={'fontsize':12})
plt.axis('tight')

#plt.savefig('Utah_Increase_Hospitalized.png')
plt.show()
```



1.9 Plot the Daily Deaths with 7 & 20 day rolling average

Ultimately, the metric of interest is the number of deaths. As long as hospitalizations don't overwhelm the system the number of deaths should stay relatively low. If we see a sudden and sustained increase in deaths it may indicate the overwhelming of the hospitals.

[9]:
```
# Calculate the 7 & 20 day rolling averages:
Roll_Avg(UT,'deathIncrease', [7,20])

fig, ax = plt.subplots(figsize = (12,6))
watermark()
my_annotate(text_loc,text=False)
plt.bar(UT['date'], UT['deathIncrease'], label='Death Increase',color='grey')
plt.plot(UT['date'], UT.roll_deathIncrease_7, label='7 Day Average',␣
 ↪color='green')
plt.plot(UT['date'], UT.roll_deathIncrease_20, label='20 Day Average',␣
 ↪color='blue')
plt.legend(loc='upper left')

date_ticks()

plt.title('Utah Deaths Increase Rolling Average', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Deaths Increase', fontdict={'fontsize':12})
plt.axis('tight')

#plt.savefig('Utah_Death_Rolling_Avg.png')
plt.show()
```

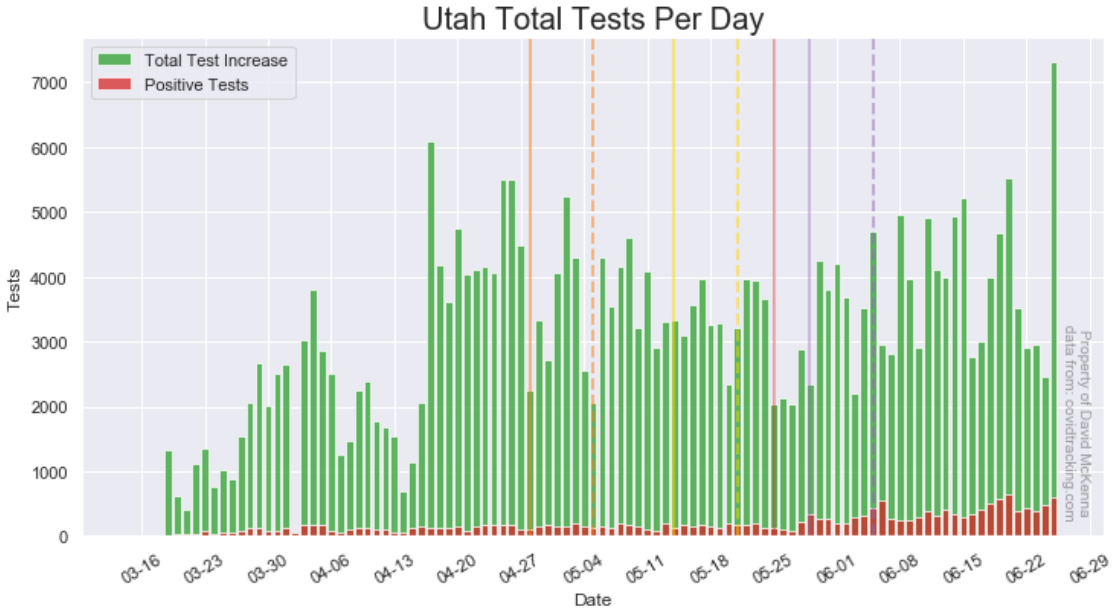


1.10 Plot the Total tests and which of those are positive

I now turn my attention to the testing in Utah. Here I plot the daily test increase and the fraction of those that return positive. Again, the positive test may lag the total because it takes up to 48

hrs to receive results.

[10]:
```
sns.set()

fig, ax = plt.subplots(figsize = (12,6))
watermark(loc_x=0.89)
my_annotate(text_loc,text=False) #add annotation lines without text

ax.bar(UT['date'], UT['totalTestResultsIncrease'],
       label='Total Test Increase',color='tab:green',alpha=0.75)
ax.bar(UT['date'], UT['positiveIncrease'],
       label='Positive Tests',color='tab:red',alpha=0.75)
date_ticks()

ax.legend(loc='upper left')
plt.title('Utah Total Tests Per Day', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Tests', fontdict={'fontsize':12})
plt.axis('tight')
#plt.savefig('Utah_Increase_Test.png')

plt.show()
```

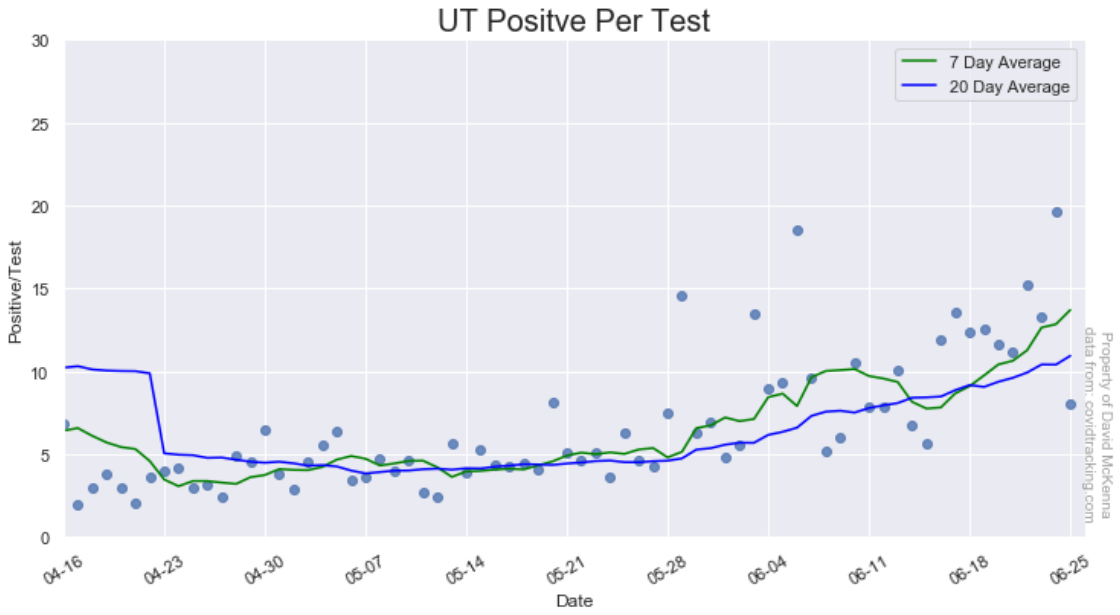


1.11 For clarity I plot the positive test rate as a percentage of total tests

The positive test rate can help determine if we are testing enough. A low positive test rate could indicate that the tests represent a good sample of the population. A large positive rate could indicate 1) there are not enough tests being performed for screening, 2) there is a spike in the local

outbreak, 3) the virus is extremely contagious. In this case we see it level off at about 5% for a while, this gives me confidence that at that time there were sufficient tests performed and the actual infection rate was about 5%. About 5-29 the rate starts increasing which might indicate a new spike in cases, or that we have dropped below the threshold of tests per day to provide an accurate picture.

[11]:
```
# calculate the 7 & 20 day rolling averages
Roll_Avg(UT,'PosPerTest', [7,20])
#define dates of interest in terms of ordinal
start = UT[UT['date'] == '2020-04-16']['date']
end = UT.date.max() + pd.DateOffset(1)

fig, ax = plt.subplots(figsize = (12,6))
watermark()

fig = sns.regplot(x = 'date', y = 'PosPerTest', data = UT, fit_reg=False)
plt.plot(UT['date'], UT.roll_PosPerTest_7, label='7 Day Average', color='green')
plt.plot(UT['date'], UT.roll_PosPerTest_20, label='20 Day Average',␣
 ↪color='blue')

date_ticks(7)

plt.legend(loc='upper right')
ax.set_xlim(start,end)
ax.set_ylim(0,30)
plt.title('UT Positve Per Test', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Positive/Test', fontdict={'fontsize':12})

#plt.savefig('UT_Positive_Per_Test.png')
plt.show()
```



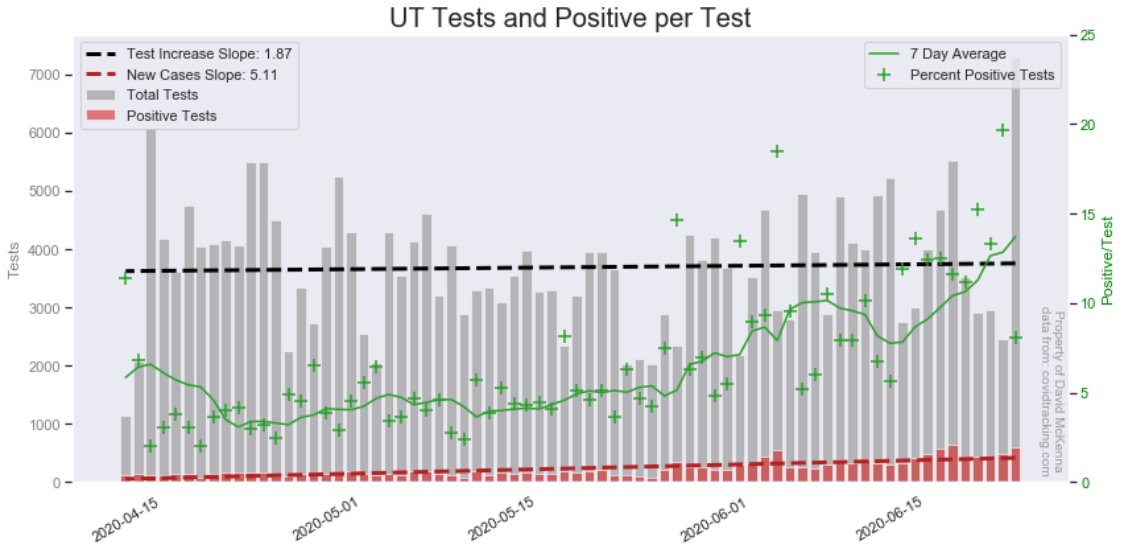
1.12 It is easier to see the Total Tests, Positive Tests, and the Positive per Test

From the linear fit it is clear to see that the Total Tests per day is decreasing while the Positive Tests per day is increasing, this yields a proportional increase in Positives per Test. This contradicts the claim of President of testing bias, ie. we have more cases just because we are testing more. This is the case only when not enough tests are being performed. It is also important to understand that tests IDENTIFY cases, they do NOT CREATE cases. If there are insufficient tests being done to create a representation of the population (sufficient randomized sample) then if testing is increased it will identify more cases until the sampling threshold is reached, once reached the case rate will level off at the 'true' value.

[12]:
```
UT_1 = UT[UT['date'] >= '2020-04-15'] # select the data after the rampup of␣
 ↪testing
#define start and end dates for x limits- thus we can average
#over all data but only plot the interesing part
start = UT_1[UT_1['date'] == '2020-04-15']['date']
end = UT_1.date.max() + pd.DateOffset(1)
Roll_Avg(UT,'PosPerTest', [7])

# I use the scikit LinearRegression to perfrom the least squares linear fit of␣
 ↪the data

#setup the x values to be fit as ordinal because LinerRegressor doesnt like␣
 ↪datetime objs
X_i = UT_1['date_ordinal']
X_i = np.array(X_i).reshape(-1,1)
X_t = X_i
```

```
# Create the Y values for the two fits we will do
Y_i = UT_1['totalTestResultsIncrease']
Y_t = UT_1['positiveIncrease']

linear_regressor_i = LinearRegression()  # create object for the class
linear_regressor_i.fit(X_t, Y_i)  # perform linear regression
Y_i_pred = linear_regressor_i.predict(X_i)  # make predictions

linear_regressor_t = LinearRegression()
linear_regressor_t.fit(X_t, Y_t)
Y_t_pred = linear_regressor_t.predict(X_t)

sns.set_style("dark")  # to remove grid lines because of the 2 y axis they␣
 ↪overlap and are confusing
fig, ax1 = plt.subplots(figsize = (12,6))

plt.title('UT Tests and Positive per Test', fontdict={'fontsize':20})
date_ticks()
watermark(loc_x=0.94)

color = 'gray'

ax1.bar(UT_1['date'], UT_1['totalTestResultsIncrease'],
        label='Total Tests',color='gray',alpha=0.5)
ax1.bar(UT_1['date'], UT_1['positiveIncrease'],
        label='Positive Tests',color='tab:red',alpha=0.6)
ax1.plot(UT_1['date'],Y_i_pred, color='black',
         linestyle = '--', linewidth = 3,
         label = 'Test Increase Slope: {:.2f}'.format(linear_regressor_i.
 ↪coef_[0]))
ax1.plot(UT_1['date'],Y_t_pred, color='firebrick',
         linestyle = '--', linewidth = 3,
         label = 'New Cases Slope: {:.2f}'.format(linear_regressor_t.coef_[0]))
ax1.tick_params(axis='y', labelcolor=color)
ax1.legend(loc='upper left')
fig = plt.ylabel('Tests', fontdict={'fontsize':12},color=color)

ax2 = ax1.twinx()

color = 'green'
fig = plt.scatter(UT_1['date'], UT_1['PosPerTest'],
                  marker='+',s=100,color='tab:green' ,label='Percent Positive␣
 ↪Tests')
ax2 = plt.plot(UT_1['date'], UT_1.roll_PosPerTest_7, label='7 Day Average',␣
 ↪color='tab:green')
ax2 = plt.ylabel('Positive/Test', fontdict={'fontsize':12},color=color)
ax2 = plt.tick_params(axis='y', labelcolor=color)
```

```
ax2 = plt.ylim(0,25)
ax2 = plt.legend(loc='upper right')

fig = plt.tight_layout()

#plt.savefig('UT_Test_+_Positive_Per_Test.png')
plt.show()
```

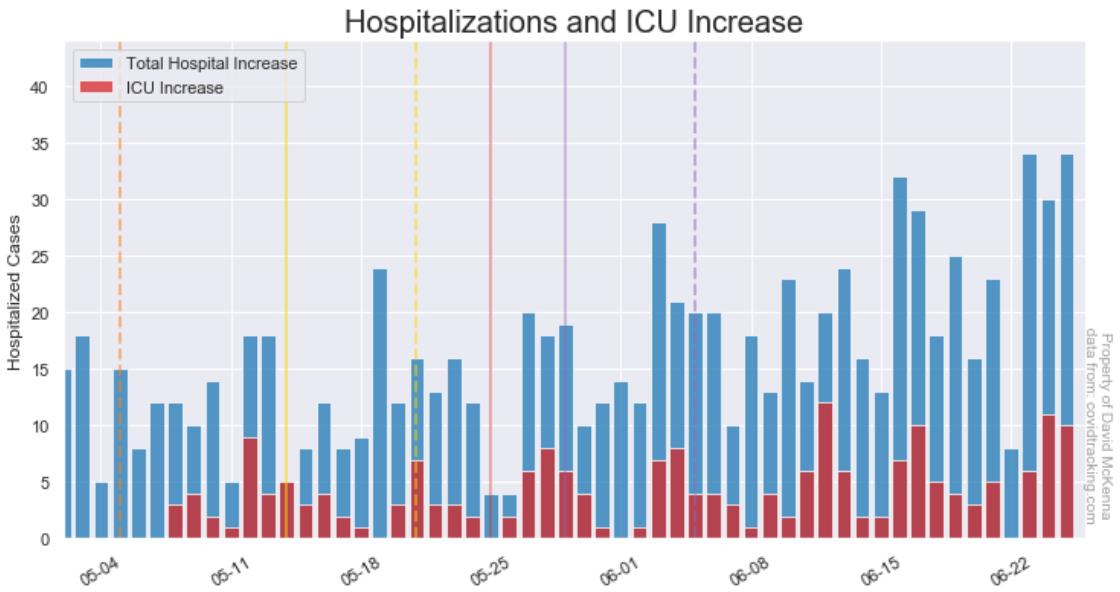


1.13 We can look closer at the Hospital situation

Here I plot the hospitalizations by day and the fraction of those that are admitted to the ICU. Again, the ICU data may lag the hospitalizations as patients are usually admitted to the hospital and then admitted to the ICU at a later date if their symptoms worse. Since space in the ICU is limited, when they become full is where we should best define the capacity of the healthcare system. Deaths that otherwise may have been prevented will occur when ICU is over capacity and triage would need to begin.

[13]:
```
# again using the truncated set because ICU data wasn't available before 5-8
UT.sort_index() #sort by date

#create shifted column that I can then calculate the daily increase
UT['PreviousDayICU'] = UT['inIcuCumulative'].shift(-1)

# iterate through the rows to calculate daily increase in ICU cases
for row in UT.iterrows() :
    UT['icuIncrease'] = UT['inIcuCumulative'] - UT['PreviousDayICU']

#drop the added column from data frame
UT.drop(columns = 'PreviousDayICU',inplace=True)
```

```
#define x limits to frame the timeframe of interest
start = UT[UT['date'] == '2020-05-02']['date']
end = UT.date.max() + pd.DateOffset(1)
sns.set()# reset the plot style
fig, ax = plt.subplots(figsize = (12,6))
watermark()
my_annotate(text_loc = 18, text=False)
ax.bar(UT['date'], UT['hospitalizedIncrease'],
       label='Total Hospital Increase',color='tab:blue',alpha=0.75)
ax.bar(UT['date'], UT['icuIncrease'],
       label='ICU Increase',color='tab:red',alpha=0.75)
ax.legend(loc='upper left')
ax.set_xlim(start,end)
#set max y-value to be the max in window of interest + 10
ax.set_ylim(0,max(UT[UT['date']>= '2020-05-02']['hospitalizedIncrease'])+10)
plt.title('Hospitalizations and ICU Increase', fontdict={'fontsize':20})
plt.ylabel('Hospitalized Cases', fontdict={'fontsize':12})
date_ticks()
#plt.savefig('ICU.png')
plt.show()
```



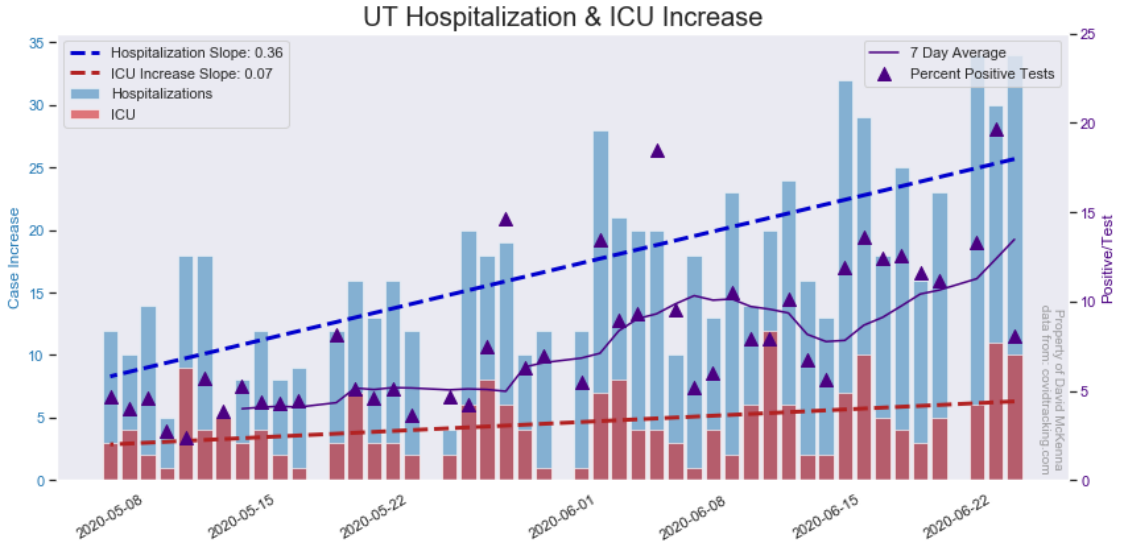
1.14 I do a similar overlay of Hospitalizations and ICU as I did with Tests:

I add linear regression for the hospitalizations and the ICU case increase. I also plot the positive per test percentage with its 7-day average. This gives a good picture of the critical, decision making, trends. Ultimately, to make sure there are no otherwise preventable deaths we need to keep the ICU below capacity (more importantly below ventilator capacity) so as not to overwhelm the health

care system. As the hospitalizations and ICU cases trend upward mitigation strategies should be put back in place.

[14]:
```
UT_1 = UT[UT['date'] >= '2020-04-20'] # select the data after the rampup of␣
 ↪testing
#define start and end dates for x limits- thus we can average
#over all data but only plot the interesing part
start = UT_1[UT_1['date'] == '2020-05-11']['date']
end = UT_1.date.max() + pd.DateOffset(1)
UT_1 = UT_1[(UT_1['icuIncrease']<1000) & (UT_1['icuIncrease']>0)]
Roll_Avg(UT_1,'PosPerTest', [7])
# I use the scikit LinearRegression to perfrom the least squares linear fit of␣
 ↪the data

#setup the x values to be fit as ordinal because LinerRegressor doesnt like␣
 ↪datetime objs
X_i = UT_1['date_ordinal']
X_i = np.array(X_i).reshape(-1,1)
X_t = X_i
# Create the Y values for the two fits we will do
Y_i = UT_1['hospitalizedIncrease']
Y_t = UT_1['icuIncrease']

linear_regressor_i = LinearRegression()  # create object for the class
linear_regressor_i.fit(X_t, Y_i)  # perform linear regression
Y_i_pred = linear_regressor_i.predict(X_i)  # make predictions

linear_regressor_t = LinearRegression()
linear_regressor_t.fit(X_t, Y_t)
Y_t_pred = linear_regressor_t.predict(X_t)

sns.set_style("dark")  # to remove grid lines because of the 2 y axis they␣
 ↪overlap and are confusing
fig, ax1 = plt.subplots(figsize = (12,6))

plt.title('UT Hospitalization & ICU Increase', fontdict={'fontsize':20})
watermark(loc_x=0.94)
date_ticks()
color = 'tab:blue'

ax1.bar(UT_1['date'], UT_1['hospitalizedIncrease'],
        label='Hospitalizations',color='tab:blue',alpha=0.5)
ax1.bar(UT_1['date'], UT_1['icuIncrease'],
        label='ICU',color='tab:red',alpha=0.6)
ax1.plot(UT_1['date'],Y_i_pred, color='mediumblue',
         linestyle = '--', linewidth = 3,
```

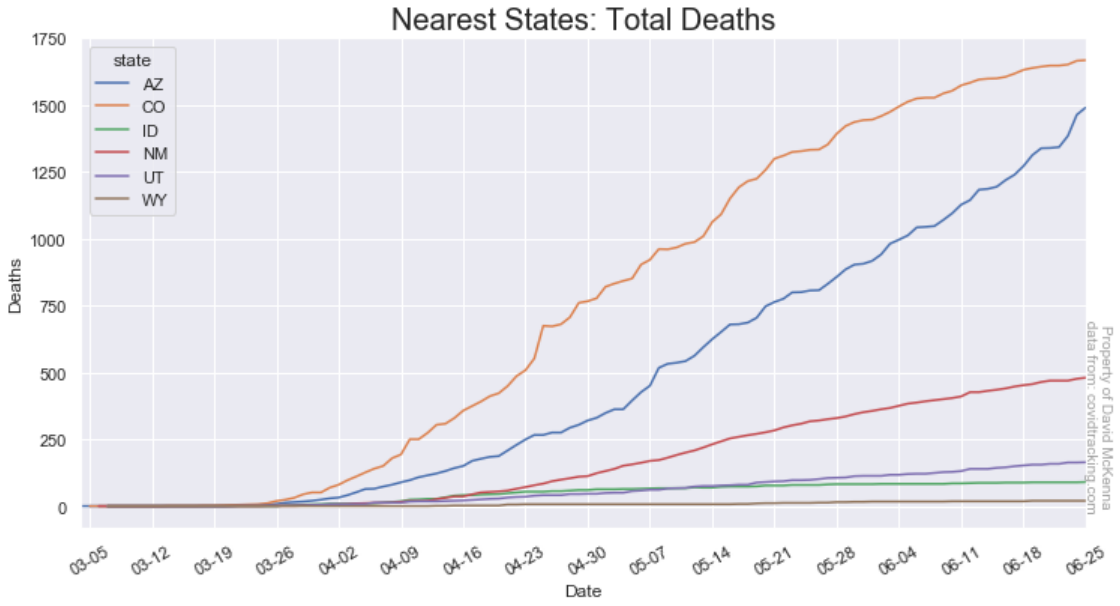
```
        label = 'Hospitalization Slope: {:.2f}'.format(linear_regressor_i.
 ↪coef_[0]))
ax1.plot(UT_1['date'],Y_t_pred, color='firebrick',
        linestyle = '--', linewidth = 3,
        label = 'ICU Increase Slope: {:.2f}'.format(linear_regressor_t.
 ↪coef_[0]))
ax1.tick_params(axis='y', labelcolor=color)
ax1.legend(loc='upper left')
fig = plt.ylabel('Case Increase', fontdict={'fontsize':12},color=color)

ax2 = ax1.twinx()

color = 'indigo'
fig = plt.scatter(UT_1['date'], UT_1['PosPerTest'],
                  marker='^',s=100,color='indigo' ,label='Percent Positive␣
 ↪Tests')
ax2 = plt.plot(UT_1['date'], UT_1.roll_PosPerTest_7, label='7 Day Average',␣
 ↪color='indigo')
ax2 = plt.ylabel('Positive/Test', fontdict={'fontsize':12},color=color)
ax2 = plt.tick_params(axis='y', labelcolor=color)
ax2 = plt.ylim(0,25)
ax2 = plt.legend(loc='upper right')

fig = plt.tight_layout()

#plt.savefig('UT_Test_+_Positive_Per_Test.png')
plt.show()
```

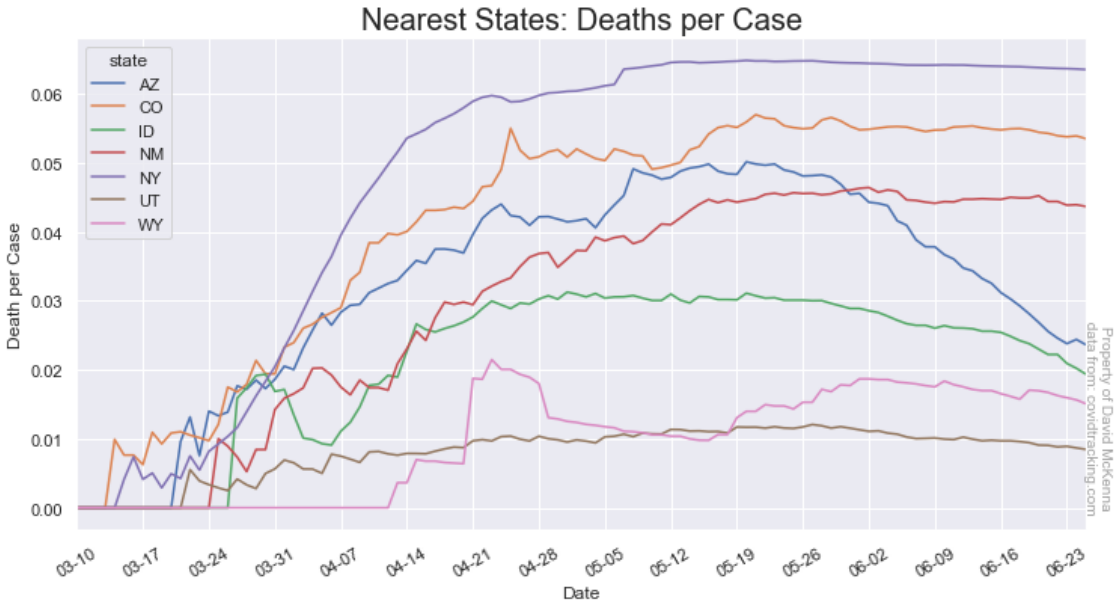


1.15 Now I compare the Deaths and Deaths per case in UT with nearby states

I added in New York to the Deaths Per Case to compare. When compared the nearest states UT has one of the lowest total deaths and the lowest deaths per case. All investigated states have a level or negative slope in the deaths per case, this is good news. That means that although cases are increasing the chance of survival is also increasing. This is most likely due to better treatment plans, although it could suggest the most at risk people already got infected and were lost. I like to think it is because of better treatment. A sudden increase in deaths per case canm indicate a saturation of ICU capacity.

[15]:
```
fc = df[df['state'].isin(['UT','AZ','CO','NM','ID','WY'])]
fc['Date'] = pd.to_datetime(df['date'].astype(str), infer_datetime_format=True)
fc['Date'] = fc[fc['date']>= '03-10-2020']
sns.set()
fig, ax = plt.subplots(figsize = (12,6))
watermark()
fig = fc.groupby(['date','state']).sum()['death'].unstack().plot(ax=ax)
date_form = DateFormatter("%m-%d")
ax.xaxis.set_major_formatter(date_form)
date_ticks()


plt.title('Nearest States: Total Deaths' , fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Deaths', fontdict={'fontsize':12})

plt.show()

# Plot Deaths per Case
# Add NY to the dataset
fc = df[df['state'].isin(['UT','AZ','CO','NM','ID','WY','NY'])]
fc['Date'] = pd.to_datetime(df['date'].astype(str), infer_datetime_format=True)
fc = fc[fc['Date']>= '03-10-2020']

fig, ax = plt.subplots(figsize = (12,6))
watermark()
fig = fc.groupby(['Date','state']).sum()['DperP'].unstack().plot(ax=ax)
date_ticks()
plt.title('Nearest States: Deaths per Case', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Death per Case', fontdict={'fontsize':12})

plt.show()
```



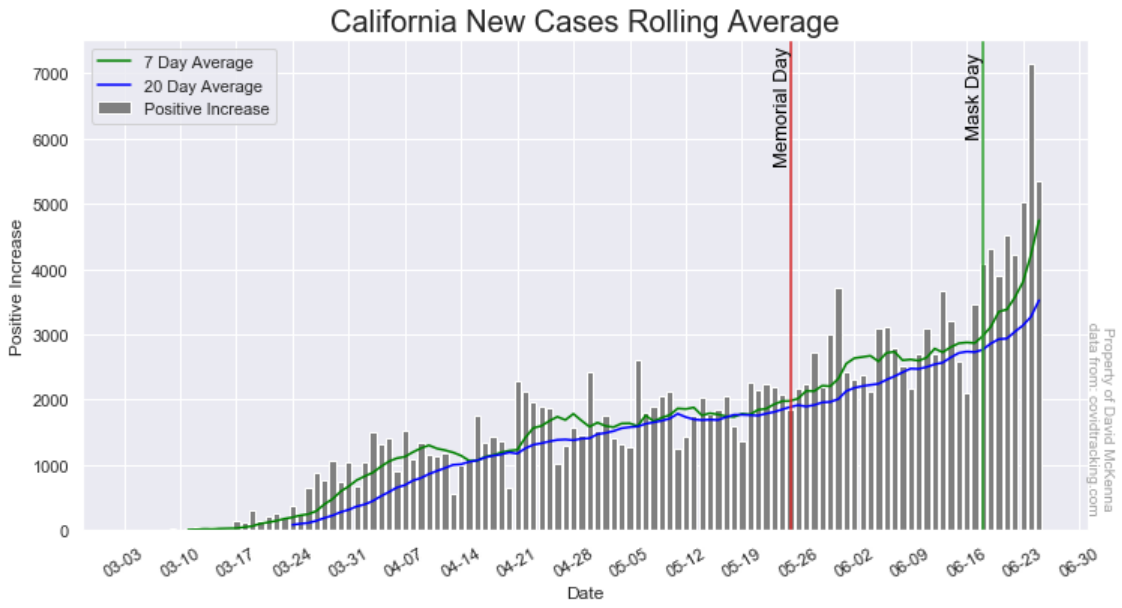


2 Now I look at CA doing similar analysis

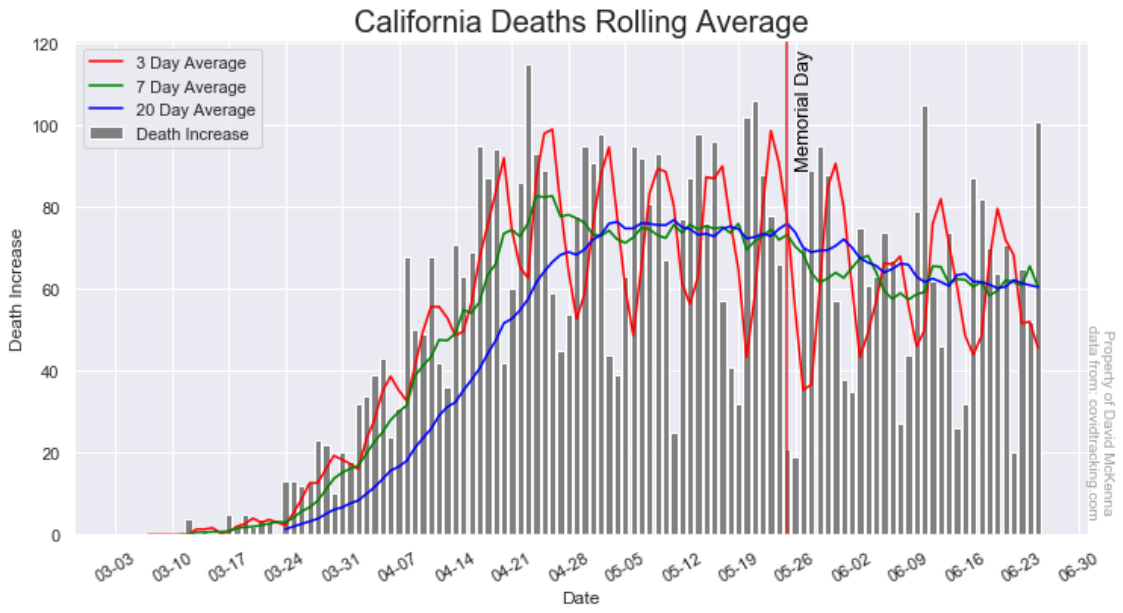
I will not explain as in depth since descriptions are available for the similar analysis for Utah.

[16]:
```
CA = df[df['state']=='CA']
maskDay = CA[CA['date']=='2020-06-18']['date_ordinal']
maskDay = int(maskDay)
```

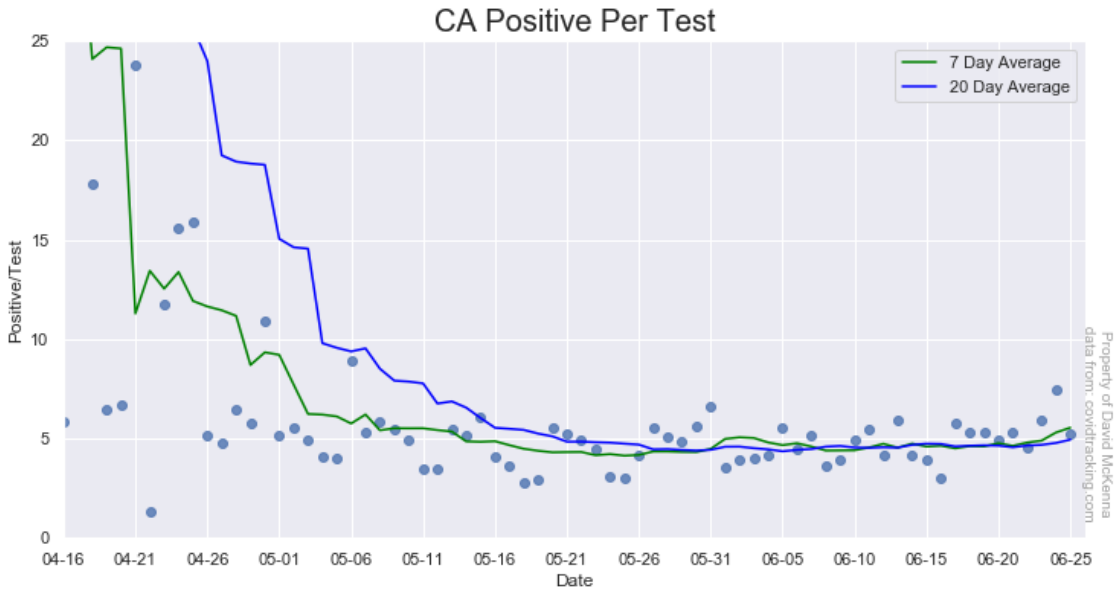
```
text_loc = max(CA['positiveIncrease'])-1500

Roll_Avg(CA, 'positiveIncrease', [7,20])

fig, ax = plt.subplots(figsize = (12,6))
watermark()


plt.bar(CA['date'], CA['positiveIncrease'],
        label='Positive Increase',color='grey')
plt.plot(CA['date'], CA.roll_positiveIncrease_7,
         label='7 Day Average', color='green')
plt.plot(CA['date'], CA.roll_positiveIncrease_20,
         label='20 Day Average', color='blue')
plt.legend(loc='upper left')
ax.axvline( x=MemorialDay, color='tab:red', linewidth=1.5)
ax.annotate('Memorial Day', (MemorialDay - 2 ,text_loc),
            color='black',rotation=90,fontsize=13)
ax.axvline( x=maskDay, color='tab:green', linewidth=1.5)
ax.annotate('     Mask Day', (maskDay - 2 ,text_loc),
            color='black',rotation=90,fontsize=13)
ax.set_xlim(CA['date'].min() , CA['date'].max() )

plt.title('California New Cases Rolling Average', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Positive Increase', fontdict={'fontsize':12})
plt.axis('tight')
date_ticks()
#plt.savefig('CA_Increase_Rolling_Avg.png')
plt.show()
```



[17]:
```
Roll_Avg(CA,'deathIncrease', [3,7,20])

fig, ax = plt.subplots(figsize = (12,6))
watermark()
date_ticks()
plt.bar(CA['date'], CA['deathIncrease'], label='Death Increase',color='grey')
plt.plot(CA['date'], CA.roll_deathIncrease_3, label='3 Day Average',␣
 ↪color='red')
plt.plot(CA['date'], CA.roll_deathIncrease_7, label='7 Day Average',␣
 ↪color='green')
plt.plot(CA['date'], CA.roll_deathIncrease_20, label='20 Day Average',␣
 ↪color='blue')
plt.legend(loc='upper left')
plt.xticks(rotation=30)
ax.axvline( x=MemorialDay, color='tab:red', linewidth=1.5)
ax.annotate('Memorial Day', (MemorialDay + 1␣
 ↪,90),color='black',rotation=90,fontsize=13)
ax.set_xlim(CA['date_ordinal'].min() , CA['date_ordinal'].max() )
plt.title('California Deaths Rolling Average', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Death Increase', fontdict={'fontsize':12})
plt.axis('tight')

#plt.savefig('CA_Death_Rolling_Avg.png')
plt.show()
```



[18]:
```
Roll_Avg(CA,'PosPerTest', [7,20])

start = CA[CA['date'] == '2020-04-16']['date']
end = CA.date.max() + pd.DateOffset(1)
fig, ax = plt.subplots(figsize = (12,6))
watermark()
date_ticks(5)

fig = sns.regplot(x = 'date_ordinal', y = 'PosPerTest', data = CA,␣
 ↪fit_reg=False)
plt.plot(CA['date'], CA.roll_PosPerTest_7, label='7 Day Average', color='green')
plt.plot(CA['date'], CA.roll_PosPerTest_20, label='20 Day Average',␣
 ↪color='blue')

plt.legend(loc='upper right')
ax.set_xlim(start, end)
ax.set_ylim(0,25)
plt.title('CA Positive Per Test', fontdict={'fontsize':20})
plt.xlabel('Date', fontdict={'fontsize':12})
plt.ylabel('Positive/Test', fontdict={'fontsize':12})

#plt.savefig('CA_Positive_Per_Test.png')
plt.show()
```



3 Now I look at the 10 states with the largest case increases.

I take the 7 day average of the case increase of all states then I choose the 10 largest. I plot these on a horizontal bar plot in decsending order.

[19]:
```
today = str(df.date.max())

df_tot =␣
 ↪df[['state','date','positiveIncrease','deathIncrease','totalTestResultsIncrease']].
 ↪reset_index()
df_tot['Date'] = pd.to_datetime(df['date'], format= '%Y%m%d')
df_tot.set_index(['state','Date'],inplace=True)
df_tot.sort_index(inplace=True)
df_tot['date_ordinal'] = pd.to_datetime(df_tot['date']).apply(lambda date: date.
 ↪toordinal())
df_tot['posDiff'] = df_tot.groupby(level='state')['positiveIncrease'].
 ↪apply(lambda x: (x.rolling(10).mean()))

largest = df_tot[df_tot['date']==today]['posDiff'].nlargest(10).astype(int)

fig, ax = plt.subplots(figsize = (12,6))
watermark()

df_2 = pd.DataFrame(largest).reset_index().sort_index(ascending=False)
plt.barh(df_2.state,df_2.posDiff,color = 'lightcoral')
plt.title('States with Largest 10 day Avg New Cases', fontsize=16)
plt.ylabel('State')
```

```
plt.xlabel('Avg New Cases')
#plt.savefig('Top_Ten_Increase.png')
plt.show()
```

3.1 Since total increase only tells part of the story:

3.1.1 I plot the 10 states with the largest 7-day average increase scaled by population. It was suggested to me that they might have more cases because they are testing a lot more (Trump loves this argument) so I added the 7-day average of tests per day per 1M population.

I import a dataset that contains the 2019 state populations. I create a state name and abbreviation dictionary so I can map the abbreviations in the covidtracking dataset to the state names in the population dataset. I then merge the datasets and plot.

[20]:
```
names = ['State','Census','Estimates␣
 ↪Base','2010','2011','2012','2013','2014','2015','2016','2017','2018','2019']
drop = ['Census','Estimates␣
 ↪Base','2010','2011','2012','2013','2014','2015','2016','2017','2018']
pop = pd.read_excel('pop_data.xlsx',header=4,names=names)
pop.drop(columns=drop,inplace=True) #drop the unused columns
```

[21]:
```
# Create the states names and abbreviations dictionary:
states = {
        'AK': '.Alaska',
        'AL': '.Alabama',
        'AR': '.Arkansas',
        'AS': '.American Samoa',
```

```
        'AZ': '.Arizona',
        'CA': '.California',
        'CO': '.Colorado',
        'CT': '.Connecticut',
        'DC': '.District of Columbia',
        'DE': '.Delaware',
        'FL': '.Florida',
        'GA': '.Georgia',
        'GU': '.Guam',
        'HI': '.Hawaii',
        'IA': '.Iowa',
        'ID': '.Idaho',
        'IL': '.Illinois',
        'IN': '.Indiana',
        'KS': '.Kansas',
        'KY': '.Kentucky',
        'LA': '.Louisiana',
        'MA': '.Massachusetts',
        'MD': '.Maryland',
        'ME': '.Maine',
        'MI': '.Michigan',
        'MN': '.Minnesota',
        'MO': '.Missouri',
        'MP': '.Northern Mariana Islands',
        'MS': '.Mississippi',
        'MT': '.Montana',
        'NA': '.National',
        'NC': '.North Carolina',
        'ND': '.North Dakota',
        'NE': '.Nebraska',
        'NH': '.New Hampshire',
        'NJ': '.New Jersey',
        'NM': '.New Mexico',
        'NV': '.Nevada',
        'NY': '.New York',
        'OH': '.Ohio',
        'OK': '.Oklahoma',
        'OR': '.Oregon',
        'PA': '.Pennsylvania',
        'PR': '.Puerto Rico',
        'RI': '.Rhode Island',
        'SC': '.South Carolina',
        'SD': '.South Dakota',
        'TN': '.Tennessee',
        'TX': '.Texas',
        'UT': '.Utah',
        'VA': '.Virginia',
```

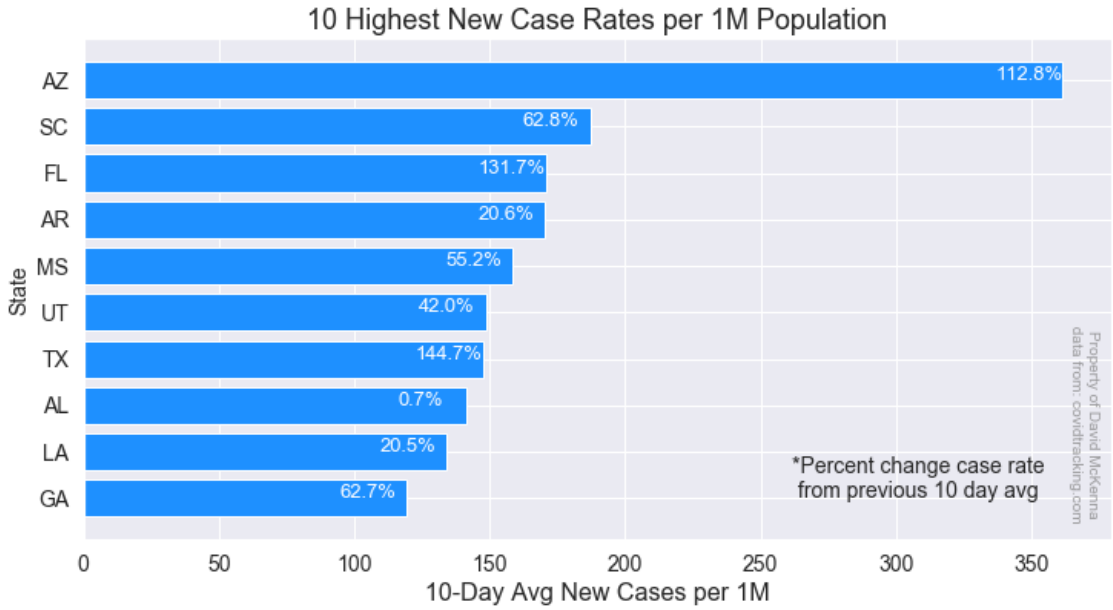
```
        'VI': '.Virgin Islands',
        'VT': '.Vermont',
        'WA': '.Washington',
        'WI': '.Wisconsin',
        'WV': '.West Virginia',
        'WY': '.Wyoming'
}
```

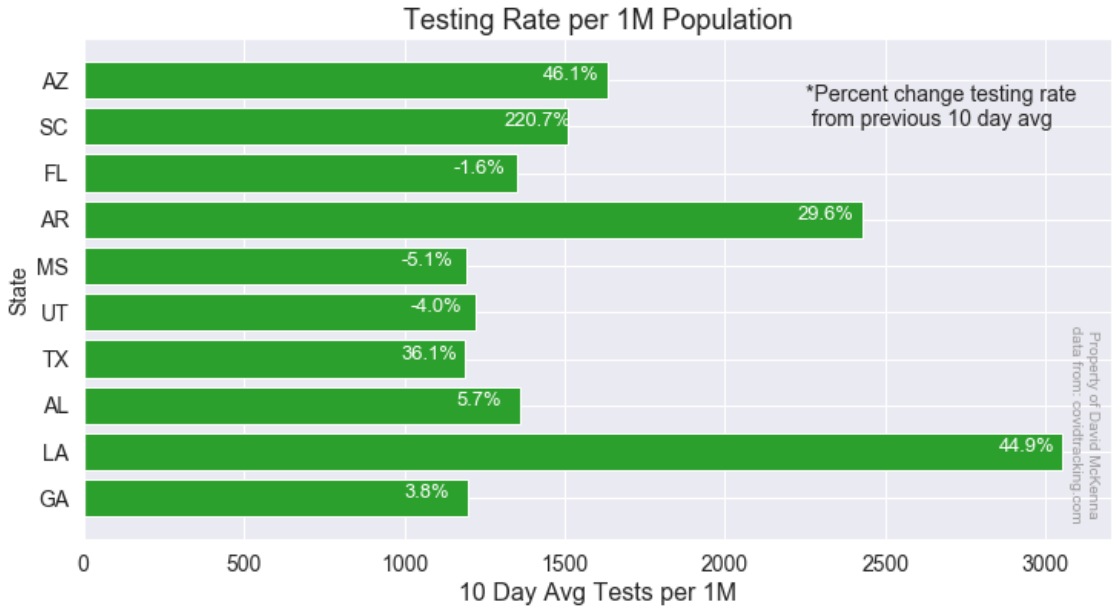
[22]:
```
# Subset the dataset with only the columns of interest, this speeds up the␣
 ↪merge:
df_1 =␣
 ↪df[['state','date','positiveIncrease','deathIncrease','totalTestResultsIncrease']].
 ↪reset_index()
df_1['full'] = df_1['state'].map(states)

# Merge datasets
df_pop = pd.merge(df_1,pop, left_on='full',right_on='State')
df_pop.drop(columns=['full','State'],inplace=True)

today = str(df_pop.date.max())
# sort index by formated date
df_pop['Date'] = pd.to_datetime(df_pop['date'], format= '%Y%m%d')
df_pop.set_index(['state','Date'],inplace=True)
df_pop.sort_index(inplace=True)

#calculate the scaled values
df_pop['PosIncScaled'] = df_pop['positiveIncrease'] / (df_pop['2019']/1000000)
df_pop['TestIncScaled'] = df_pop['totalTestResultsIncrease'] / (df_pop['2019']/
 ↪1000000)
```

[23]:
```
df_pop['PosDiffScaled'] = df_pop.groupby(level='state')['PosIncScaled'].
 ↪apply(lambda x: (x.rolling(10).mean()))
df_pop['TestIncScaled_m'] = df_pop.groupby(level='state')['TestIncScaled'].
 ↪apply(lambda x: (x.rolling(10).mean()))
df_pop['rolling_case'] = df_pop.groupby(level='state')['positiveIncrease'].
 ↪apply(lambda x: (x.rolling(10).mean()))
df_pop['rolling_case_shift'] = df_pop['rolling_case'].shift(10)
df_pop['rolling_test_shift'] = df_pop['TestIncScaled_m'].shift(10)
df_pop['pct_chg']= (df_pop['rolling_case'] - df_pop['rolling_case_shift'])/␣
 ↪df_pop['rolling_case_shift']*100
df_pop['test_pct_chg']= (df_pop['TestIncScaled_m'] -␣
 ↪df_pop['rolling_test_shift'])/ df_pop['rolling_test_shift']*100
df_pop.drop(columns = ['rolling_case_shift','rolling_case'])
```

```
largest = df_pop[df_pop['date']==today]['PosDiffScaled'].nlargest(10).
 ↪astype(int)


df_large = pd.DataFrame(largest).reset_index().sort_index(ascending=False)
states = list(df_large.state)
df_2 = df_pop[df_pop['date']==today]
df_2.reset_index(inplace=True)
df_2 = df_2[df_2['state'].isin(states)].set_index('state')
df_2 = df_2.reindex(states)


# Plot the scaled by 1M population top 10 states
fig, ax = plt.subplots(figsize = (12,6))
watermark(loc_x=0.89)

plt.barh(df_2.index,df_2.PosDiffScaled,color = 'dodgerblue')
plt.title('10 Highest New Case Rates per 1M Population', fontsize=18)
plt.ylabel('State',fontsize=14)
plt.xlabel('10-Day Avg New Cases per 1M',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
# put the percent change over last 10 days at end of bars
for i in range(len(df_2)):
    plt.annotate('{:.1f}%'.format(df_2['pct_chg'][i]),
                 xy=(df_2['PosDiffScaled'][i]-25,df_2.index[i]),
                 color='white',fontsize=13)
plt.annotate('*Percent change case rate \n from previous 10 day avg',
             xy = (max(df_2['PosDiffScaled'])-100,0),fontsize=14 )
plt.savefig('Top_Ten_Increase.png')
plt.show()

# define the
l = df_pop[df_pop['date']==today]
l.reset_index(inplace=True)
l = l[l['state'].isin(states)]

df_2 = pd.DataFrame(l).reset_index().set_index('state')
df_2 = df_2.reindex(states)
fig, ax = plt.subplots(figsize = (12,6))
plt.barh(df_2.index,df_2.TestIncScaled_m, color = 'tab:green')
watermark(loc_x=0.89)
for i in range(len(df_2)):
    plt.annotate('{:.1f}%'.format(df_2['test_pct_chg'][i]),
                 xy=(df_2['TestIncScaled_m'][i]-200,df_2.index[i]),
                 color='white',fontsize=13)
plt.annotate('*Percent change testing rate \n from previous 10 day avg',
```

```
            xy = (max(df_2['TestIncScaled_m'])-800,8),fontsize=14 )
plt.title('Testing Rate per 1M Population', fontsize=18)
plt.ylabel('State',fontsize=14)
plt.xlabel('10 Day Avg Tests per 1M',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('Top_Ten_Increase_tests.png')

plt.show()
```





4 To streamline and expand the basic analysis to any state I have defined the Basic_Analysis Function

This function takes in a string of the abbreviation of the state to be analyzed and the start date the analysis will begin (default is 2020-03-16), and test_date where the default is 202-04-15. Since testing didn't ramp up before that date the trend line (regression) is biased to a much higher slope than what is representative of the current situation. This date can be set further ahead to get an even better idea of the current situation, however I wouldn't set it any less than 4-weeks before the current date.

[24]:
```
def Basic_Analysis(state_abbrev, start_date = '2020-03-15',␣
↪test_date='2020-04-15',ICU_date='2020-05-01'):
    """Plot Case Increase and Death Increase with 7 & 20 day rolling averages
        for the state_abbrev"""
    def State_Subset(state_abbrev , start_date = '2020-03-15') :
        new_df = state_abbrev
        new_df = df[df['state']== state_abbrev]
        new_df = new_df[new_df['date']>= start_date]

        return new_df

    #subset the state of interest
    new_df = State_Subset(state_abbrev, start_date)
    interval = [7,20]#sets the intervals for the rolling averages

    #Calculate the rolling averages of positive Increase and deathIncrease
    Roll_Avg(new_df,'positiveIncrease',interval)
    Roll_Avg(new_df,'deathIncrease',interval)
    Roll_Avg(new_df,'PosPerTest',interval)
    sns.set()#set plot style to have grid

    # plot the positive increase figure
    fig, ax = plt.subplots(figsize = (12,6))
    watermark()
    date_ticks(5)
    plt.bar(new_df['date'], new_df['positiveIncrease'], label='Positive␣
↪Increase',color='grey')
    plt.plot(new_df['date'], new_df.roll_positiveIncrease_7,
         label='7 Day Average', color='green')
    plt.plot(new_df['date'], new_df.roll_positiveIncrease_20,
         label='20 Day Average', color='blue')

    plt.legend(loc='upper left')
```

```
    plt.title( str(state_abbrev)+' Case Increase Rolling Average',␣
↪fontdict={'fontsize':20})
    plt.xlabel('Date', fontdict={'fontsize':12})
    plt.ylabel('Case Increase', fontdict={'fontsize':12})

    plt.show()

    #Plot the death increase figure
    fig, ax = plt.subplots(figsize = (12,6))
    watermark()
    date_ticks(5)
    plt.bar(new_df['date'], new_df['deathIncrease'], label='Positive␣
↪Increase',color='indianred')
    plt.plot(new_df['date'], new_df.roll_deathIncrease_7,
         label='7 Day Average', color='green')
    plt.plot(new_df['date'], new_df.roll_deathIncrease_20,
         label='20 Day Average', color='blue')
    plt.legend(loc='upper left')
    plt.title( str(state_abbrev)+' Deaths Increase Rolling Average',␣
↪fontdict={'fontsize':20})
    plt.xlabel('Date', fontdict={'fontsize':12})
    plt.ylabel('Case Increase', fontdict={'fontsize':12})

    plt.show()

    #create the test df so we can limit the dates to be considered for␣
↪regression
    test_df = new_df[(new_df['PosPerTest']<50) & (new_df['PosPerTest']>0)]
    test_df = test_df[test_df['date']>= test_date]
    start = test_df[test_df['date'] == test_date]['date']
    end = test_df.date.max() + pd.DateOffset(1)


    # use the scikit LinearRegression to perfrom the least squares linear fit␣
↪of the data

    #setup the x values to be fit as ordinal because LinerRegressor doesnt like␣
↪datetime objs
    X_i = test_df['date_ordinal']
    X_i = np.array(X_i).reshape(-1,1)
    X_t = X_i

    # Create the Y values for the two fits we will do
    Y_i = test_df['totalTestResultsIncrease']
    Y_t = test_df['positiveIncrease']
```

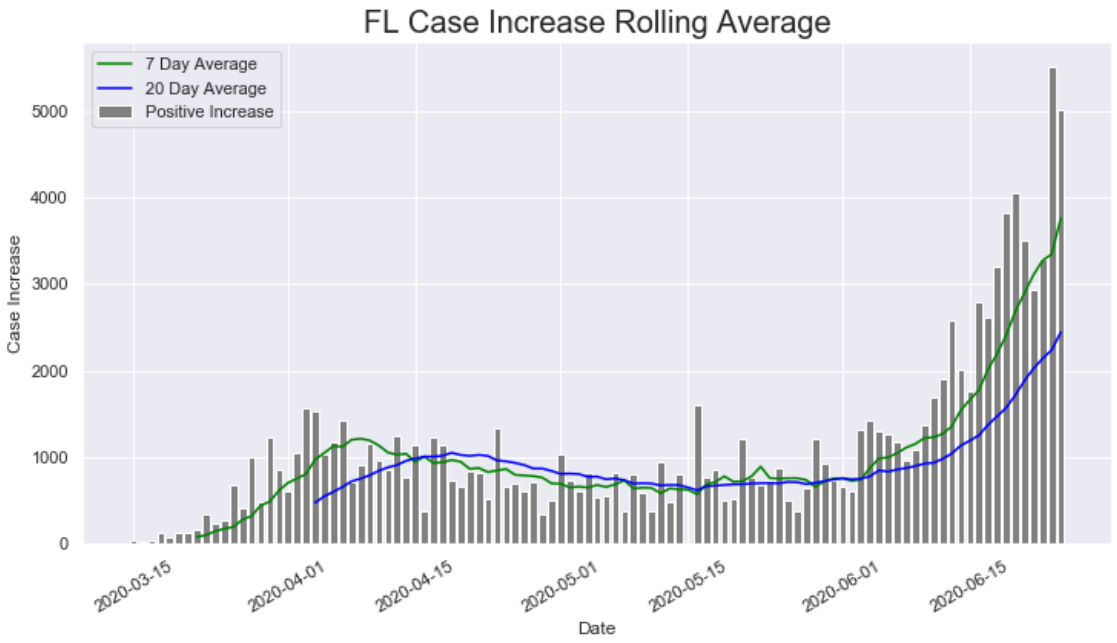
```
    linear_regressor_i = LinearRegression()  # create object for the class
    linear_regressor_i.fit(X_t, Y_i)  # perform linear regression
    Y_i_pred = linear_regressor_i.predict(X_i)  # make predictions

    linear_regressor_t = LinearRegression()
    linear_regressor_t.fit(X_t, Y_t)
    Y_t_pred = linear_regressor_t.predict(X_t)

    #plot the overlaying plots:
    sns.set_style("dark")  # to remove grid lines because of the 2 y axis they␣
↪overlap and are confusing
    fig, ax1 = plt.subplots(figsize = (12,6))

    plt.title(str(state_abbrev)+' Tests and Positive per Test',␣
↪fontdict={'fontsize':20})
    watermark(loc_x=0.94)
    date_ticks(interval=5)
    color = 'gray'

    ax1.bar(test_df['date'], test_df['totalTestResultsIncrease'],
        label='Total Tests',color='gray',alpha=0.5)
    ax1.bar(test_df['date'], test_df['positiveIncrease'],
        label='Positive Tests',color='tab:red',alpha=0.6)
    ax1.plot(test_df['date'],Y_i_pred, color='black',
        linestyle = '--', linewidth = 3,
        label = 'Test Increase Slope: {:.2f}'.format(linear_regressor_i.
↪coef_[0]))
    ax1.plot(test_df['date'],Y_t_pred, color='firebrick',
        linestyle = '--', linewidth = 3,
        label = 'Case Increase Slope: {:.2f}'.format(linear_regressor_t.
↪coef_[0]))
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.legend(loc='upper left')
    fig = plt.ylabel('Tests', fontdict={'fontsize':12},color=color)
    ax2 = ax1.twinx()
    color = 'green'
    fig = plt.scatter(test_df['date'], test_df['PosPerTest'],
                marker='+',s=100,color='tab:green' ,label='Percent Positive␣
↪Tests')
    ax2 = plt.plot(test_df['date'], test_df.roll_PosPerTest_7, label='7 Day␣
↪Average', color='tab:green')
    ax2 = plt.ylabel('Positive/Test', fontdict={'fontsize':12},color=color)
    ax2 = plt.tick_params(axis='y', labelcolor=color)
    ax2 = plt.ylim(0,25)
    ax2 = plt.legend(loc='upper right')
```

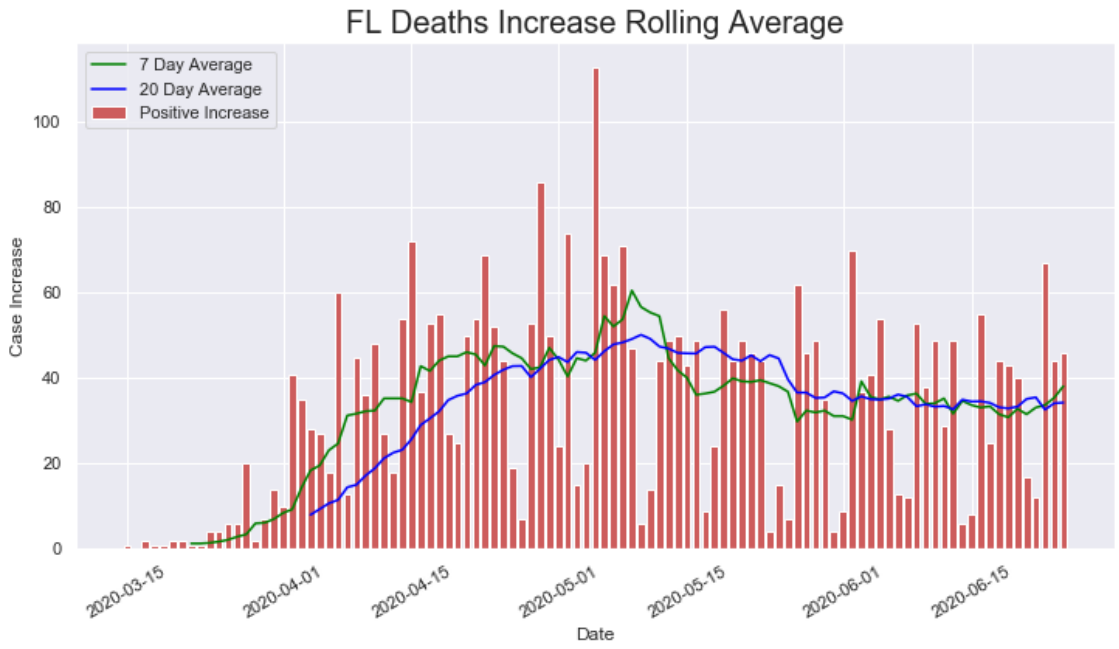
```
    fig = plt.tight_layout()
    plt.show()


#-------------------------------------------------------------------------------␣
↪
    #Perform regression on hospitalizations and ICU increase and overlay with␣
↪positive
    # per test and 7-day avg.

    # Because not all states have the ICU data (inIcuCumulative) I use to␣
↪calculate
    # the ICU increase the program does the analysis if there are at least 40␣
↪non-NULL
    # values and if not then it prints "No ICU Data Available"

    ICU_df = new_df[new_df['date']>= ICU_date]
    start = ICU_df[ICU_df['date'] == ICU_date]['date']
    end = ICU_df.date.max() + pd.DateOffset(1)
    if ICU_df.inIcuCumulative.notnull().sum() >40 :

        ICU_df = ICU_df.dropna(subset=['inIcuCumulative'])

        Roll_Avg(ICU_df,'PosPerTest',interval)
        #Hospital and ICU
        #create row for the ICU Increase:


        ICU_df.sort_index() #sort by date

        #create shifted column that I can then calculate the daily increase
        ICU_df['PreviousDayICU'] = ICU_df['inIcuCumulative'].shift(-1)
        for row in ICU_df.iterrows() :
            ICU_df['icuIncrease'] = ICU_df['inIcuCumulative'] -␣
↪ICU_df['PreviousDayICU']
        ICU_df = ICU_df[(ICU_df['icuIncrease']<1000) &␣
↪(ICU_df['icuIncrease']>0)]

        X_i = ICU_df['date_ordinal']
        X_i = np.array(X_i).reshape(-1,1)
        X_t = X_i

        # Create the Y values for the two fits we will do
        Y_i = ICU_df['hospitalizedIncrease']
        Y_t = ICU_df['icuIncrease']
```

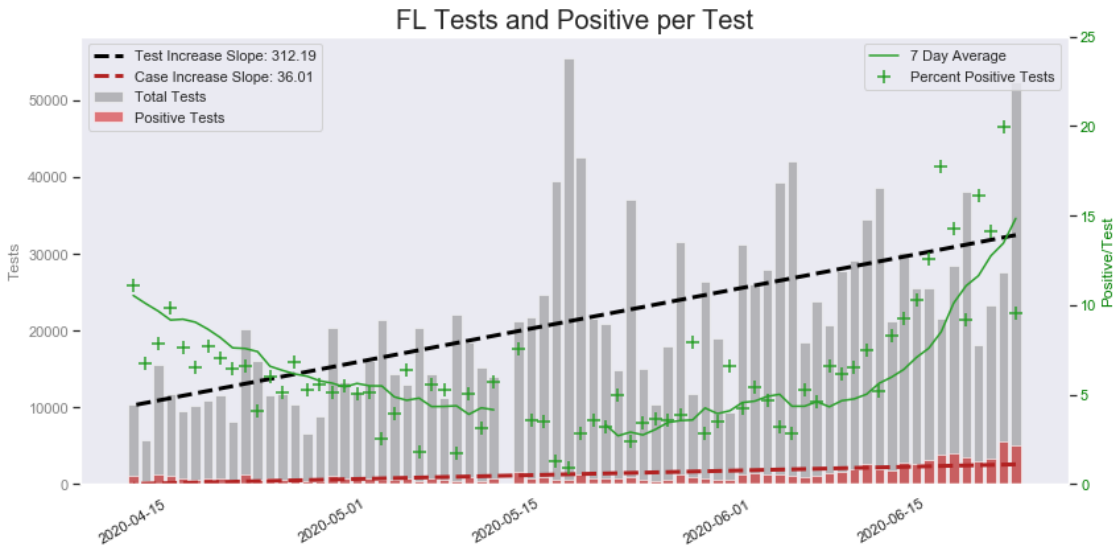
```
        linear_regressor_i = LinearRegression()  # create object for the class
        linear_regressor_i.fit(X_t, Y_i)  # perform linear regression
        Y_i_pred = linear_regressor_i.predict(X_i)  # make predictions

        linear_regressor_t = LinearRegression()
        linear_regressor_t.fit(X_t, Y_t)
        Y_t_pred = linear_regressor_t.predict(X_t)

        #plot the overlaying plots:
        sns.set_style("dark")  # to remove grid lines because of the 2 y axis␣
↪they overlap and are confusing
        fig, ax1 = plt.subplots(figsize = (12,6))

        plt.title(str(state_abbrev)+' Hospitalization/ICU & Positive per Test',␣
↪fontdict={'fontsize':20})
        watermark(loc_x=0.94)
        date_ticks(interval=5)
        color = 'tab:blue'

        ax1.bar(ICU_df['date'], ICU_df['hospitalizedIncrease'],
            label='Total Tests',color='tab:blue',alpha=0.5)
        ax1.bar(ICU_df['date'], ICU_df['icuIncrease'],
            label='Positive Tests',color='tab:red',alpha=0.6)
        ax1.plot(ICU_df['date'],Y_i_pred, color='mediumblue',
            linestyle = '--', linewidth = 3,
            label = 'Hospitalizations Slope: {:.2f}'.format(linear_regressor_i.
↪coef_[0]))
        ax1.plot(ICU_df['date'],Y_t_pred, color='firebrick',
            linestyle = '--', linewidth = 3,
            label = 'ICU Increase Slope: {:.2f}'.format(linear_regressor_t.
↪coef_[0]))
        ax1.tick_params(axis='y', labelcolor=color)
        ax1.legend(loc='upper left')
        fig = plt.ylabel('Cases', fontdict={'fontsize':12},color=color)
        ax2 = ax1.twinx()
        color = 'indigo'
        fig = plt.scatter(ICU_df['date'], ICU_df['PosPerTest'],
                    marker='+',s=100,color='indigo' ,label='Percent Positive␣
↪Tests')
        ax2 = plt.plot(ICU_df['date'], ICU_df.roll_PosPerTest_7, label='7 Day␣
↪Average', color='indigo')
        ax2 = plt.ylabel('Positive/Test', fontdict={'fontsize':12},color=color)
        ax2 = plt.tick_params(axis='y', labelcolor=color)
        ax2 = plt.ylim(0,25)
        ax2 = plt.legend(loc='upper right')
```

```
        fig = plt.tight_layout()

        plt.show()

    else :
        print("\n\n No ICU data available")
```

[25]:
```
Basic_Analysis('FL')
```

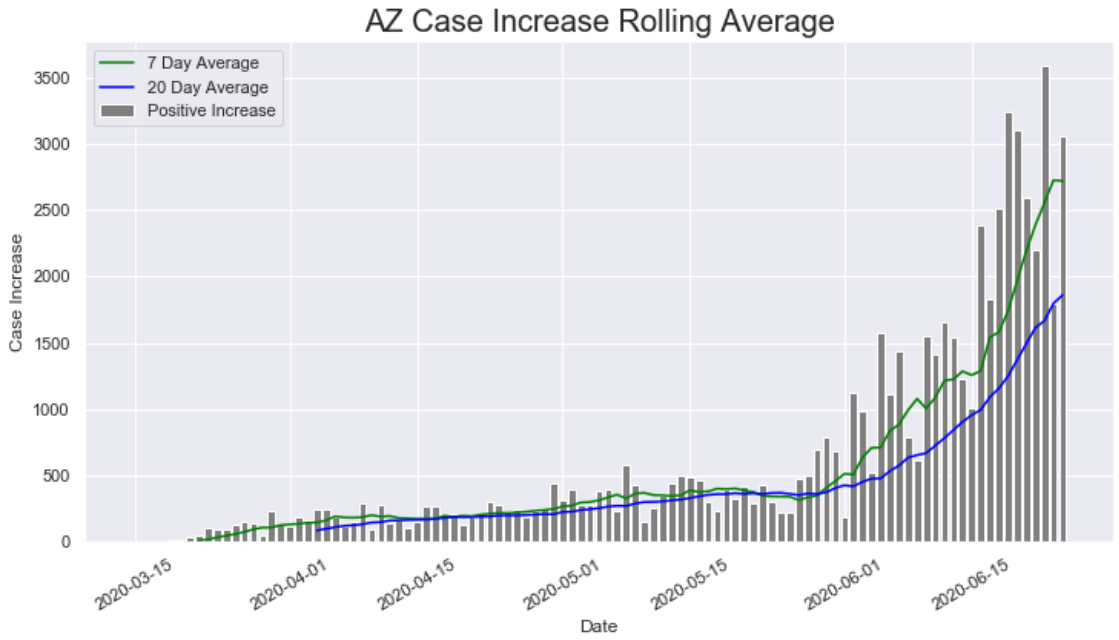


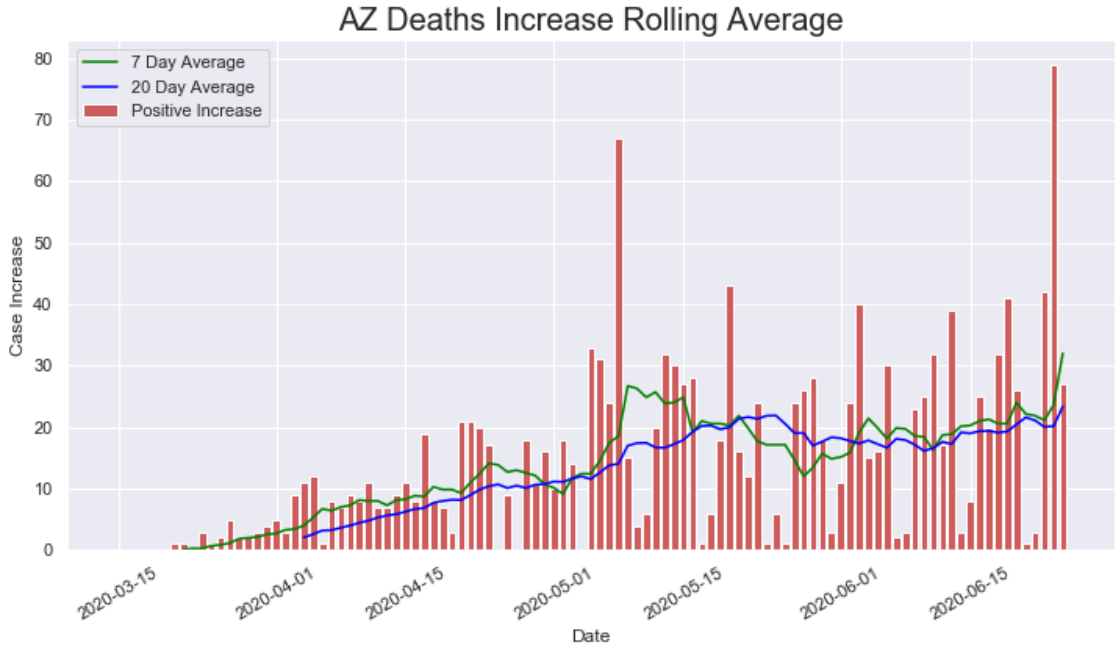


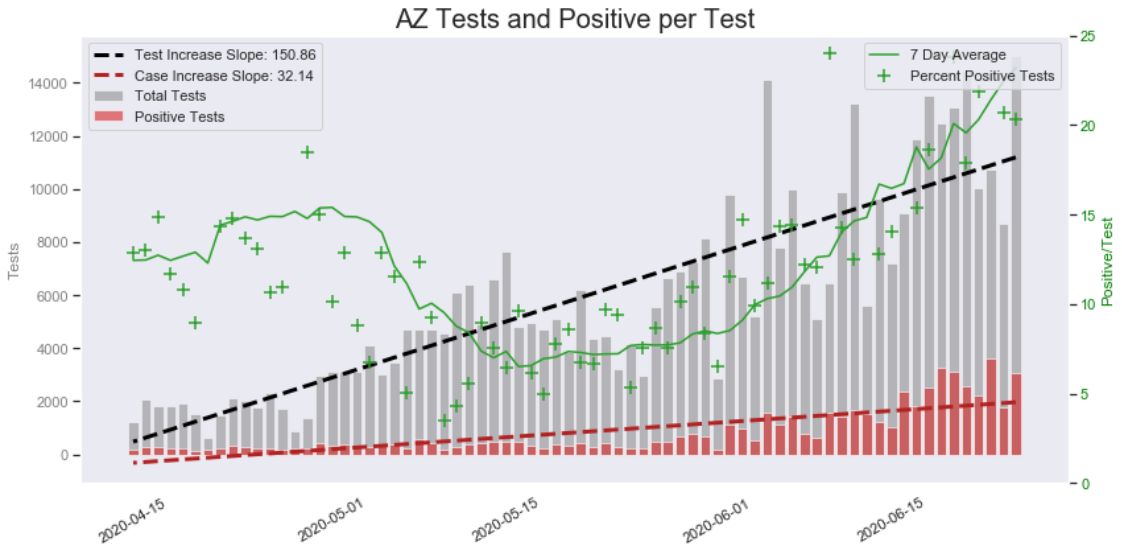



```
No ICU data available
```

[26]:
```
Basic_Analysis('AZ')
```

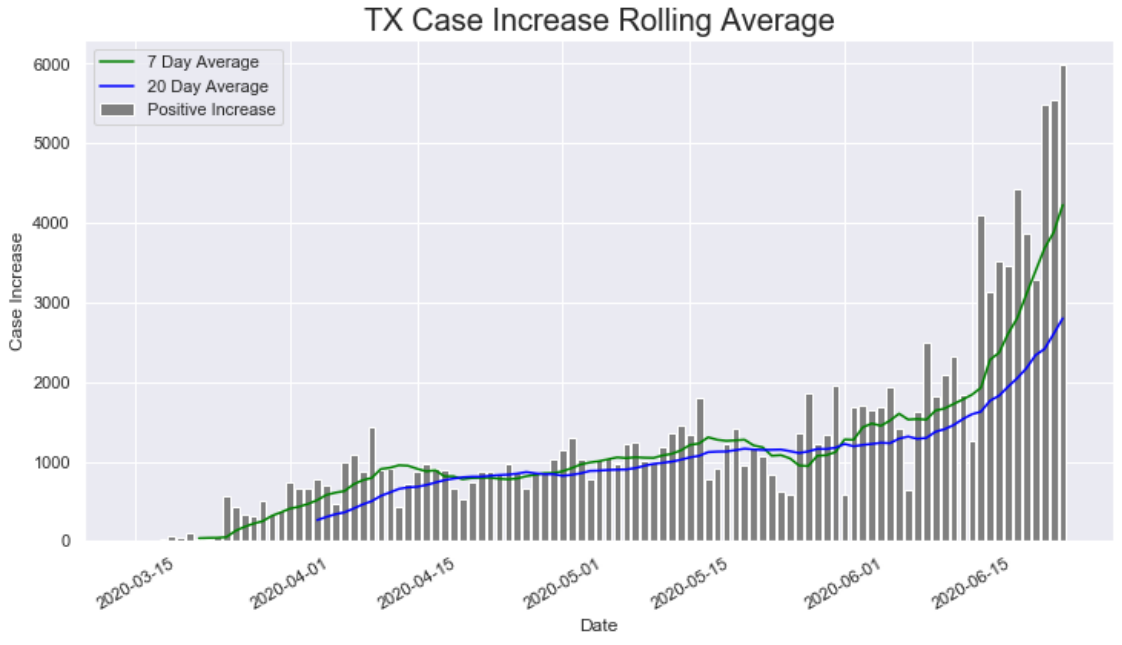

AZ Case Increase Rolling Average
7 Day Average
20 Day Average
Positive Increase
Case Increase
Date
0
500
1000
1500
2000
2500
3000
3500
2020-03-15
2020-04-01
2020-04-15
2020-05-01
2020-05-15
2020-06-01
2020-06-15

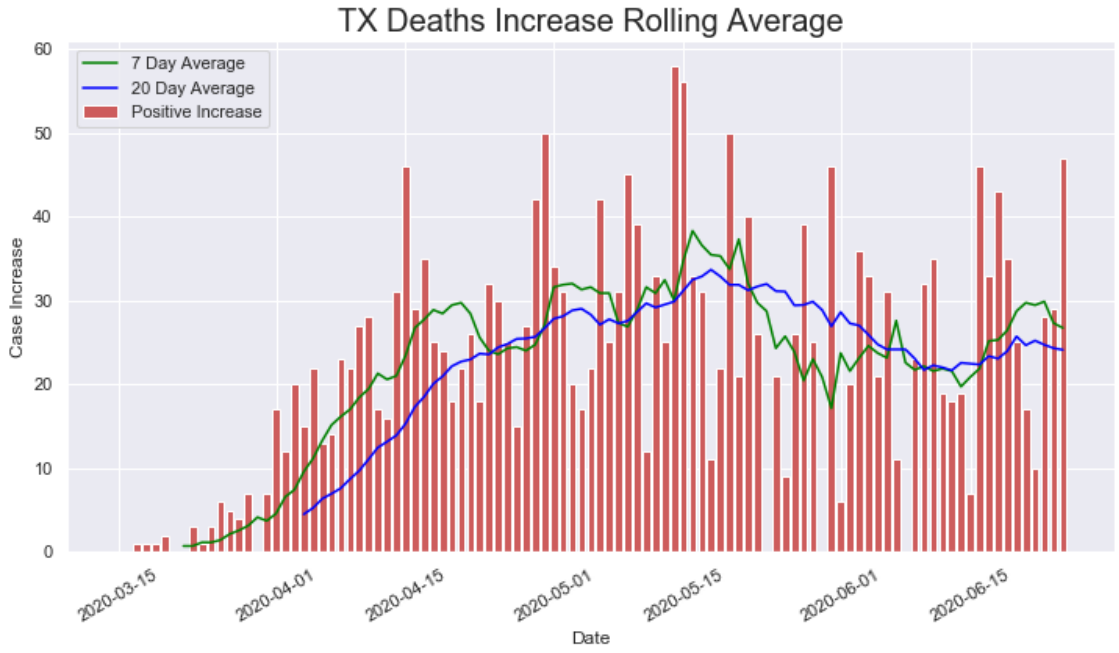

AZ Deaths Increase Rolling Average
7 Day Average
20 Day Average
Positive Increase
Case Increase
Date
0
10
20
30
40
50
60
70
80
2020-03-15
2020-04-01
2020-04-15
2020-05-01
2020-05-15
2020-06-01
2020-06-15

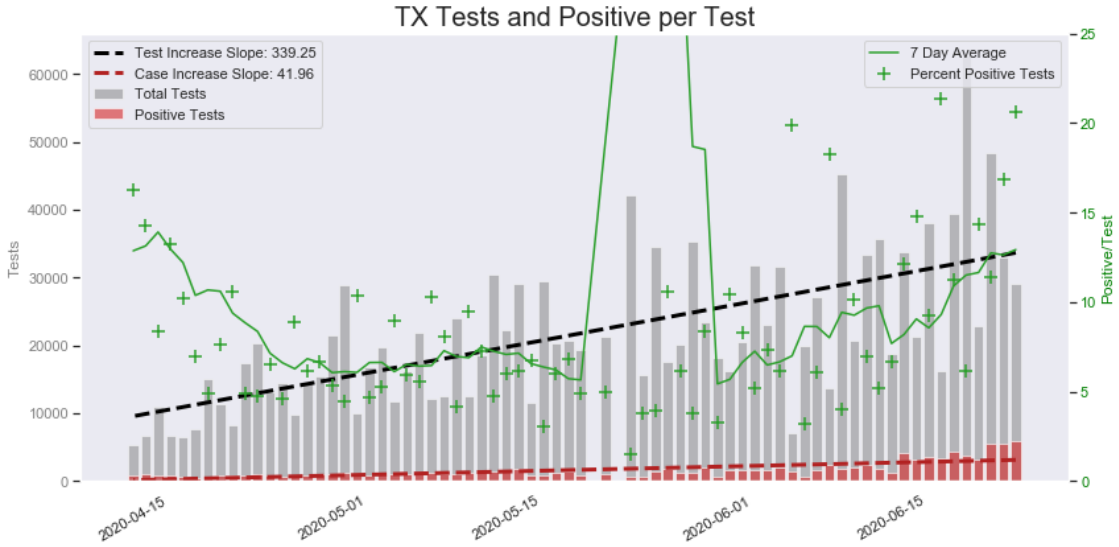


No ICU data available

[27]:
```
Basic_Analysis('TX')
```








```
No ICU data available
```

[ ]: