Lab 4: Pandas COVID

Introduction:

This notebook's purpose is to leverage pandas in order to explore data from the Covid-19 pandemic. Throughout this notebook publicly available data sets of Covid-19 data is cleaned, filtered, and visualized in order to lend a better understanding of the data than the original .csv can convey. In addition, this notebook explores the use of the *.corr()* pandas function, which uses the Pearson correlation coefficient to get the correlation between columns, both using iterated means and vectorized means of this function to discover runtime differences that exist.

Author: Nigel Nelson

Libraries:

Pandas Library

One of the primary goals of this notebook is to leverage pandas, which is a powerful open source data analysis and manipulation tool. As such, the pandas libray is imported in the below cell so it can be used for the remainder of the notebook.

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

Matplotlib Library

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python, it is going to be used throughout the notebook so it is imported in the below cell as **plt**.

In [2]:
```
import matplotlib.pyplot as plt
```

Functions:

Iterated Column Shifting

The below cell is responsible for using the data frame *.iterrows()* method in order to loop through each row of a supplied data frame, **df**, in order to return a list that represents a specified column, **column_name**, shifted by a given number of days, **shift**.

In [3]:
```
def iterated_shift(df, column_name, shift):
    none_list = [None] * len(df)
    for index, row in df.iterrows():
            if 0 <= (shift + index) < len(df):
                none_list[shift + index] = row[column_name]
    return none_list
```

Iterated Correlation

The below cell is responsible for looping from negative(**num_values**) to positive(**num_values**) on the supplied data frame, **df**, then leveraging the method **iterated_shift()** in order to get a specified column, **df_x**, shifted by the current index amount, so that the correlation between the shifted list and another specified column, **df_y**, can be concatenated to a list of correlations that is then returned.

In [4]:
```
def iterated_crosscorr(df, df_x, df_y, num_values):
    iterated_crosscorr = []
    for shift in range(-num_values, num_values):
        shifted_list = iterated_shift(df, df_x, shift)
        iterated_crosscorr.append(pd.Series(shifted_list).corr(df[df_y]))
    return iterated_crosscorr
```

Vectorized Shifted Correlation

The below cell allows for a vectorized approach to finding the Pearson correlation coefficient (https://en.wikipedia.org/wiki/Pearson_correlation_coefficient)between two data frame columns, **df_x** and **df_y**, when the column **df_y** is shifted a specified amount, **lag**.

In [5]:
```
def crosscorr(df_x, df_y, lag=0):
    return df_x.corr(df_y.shift(lag))
```

Print Max Correlation

The below cell is responsible for using a list of correlation coefficients, **corr_list**, which is correlations that are extrapolated from the data frame **df**, and finds the largest correlation coefficient within a given number values, **num_included_values**, in both the negative and positive direction of **corr_list**. The reason for only testing a portion of the correlation values is that **corr_list** is a list of correlations between to data frame columns that are shifted according to the day, and as such, at the extreme ends of the shifts, there can be only a few pieces of data to compare between the two, resulting in an untrustworthy correlation coefficient value.

In [6]:
```
def print_max_corr(corr_list, num_included_values)   :
    max_value = max(corr_list[len(corr_list)-num_included_values:len(corr_list
) + num_included_values])
    max_index = corr_list.index(max_value) - len(corr_list)/2
    print('The highest correlation is:')
    print(max_value)
    print('which occurs when days are offset by:')
    print(max_index)
```

Data Description:

The data set that this notebook is being ran on is Wisconsin's Covid-19 data which has been collected from covidtracking.com (https://covidtracking.com/data/state/wisconsin), which is a volunteer organization that compiles data directly from the public health authority in each US state and territory. The data that is being ran has data for the Covid-19 pandemic that begins on 3/3/20 and includes all of the days leading up to and including 1/6/21. The data was acquired in a .csv format and includes 310 dates that covers 42 columns of Covid-19 statistics such as deaths, positive tests, hospitalizations, and tests administered. The file is 48 Kb in size and all bench marking was conducted on the cluster using CPU, 1 node and 2 cores.

In [7]:
```
df = pd.read_csv(r'wisconsin-history.csv')
df.head()
```

Out[7]:

	date	state	dataQualityGrade	death	deathConfirmed	deathIncrease	deathProbable	hospita
0	2021-01-06	WI	A+	5435.0	5039.0	69	396.0	21
1	2021-01-05	WI	A+	5366.0	4979.0	97	387.0	21
2	2021-01-04	WI	A+	5269.0	4884.0	8	385.0	21
3	2021-01-03	WI	A+	5261.0	4875.0	5	386.0	21
4	2021-01-02	WI	A+	5256.0	4870.0	2	386.0	21

5 rows × 42 columns

Decriptive Statistics of Data

The below cell uses the data frame function **.describe()** which returns descriptive statistics for the data frame that it is being called on.

In [8]:
```
df.describe()
```

Out[8]:

	death	deathConfirmed	deathIncrease	deathProbable	hospitalized	hospitalizedCu
count	293.000000	248.000000	310.000000	192.000000	282.000000	282
mean	1507.843003	1689.314516	17.532258	85.520833	7521.102837	752
std	1417.536058	1295.378902	24.873738	120.464153	6197.693114	619
min	3.000000	340.000000	-1.000000	7.000000	337.000000	337
25%	595.000000	796.000000	2.250000	7.000000	2913.750000	291
50%	1025.000000	1160.500000	9.000000	10.000000	5405.000000	540
75%	1813.000000	2209.500000	16.000000	134.750000	10766.500000	1076
max	5435.000000	5039.000000	128.000000	396.000000	21971.000000	2197

8 rows × 39 columns

Filter By Death Amount

The below cell is responsible for filtering the data according to the condition that on that given day there was over 20 deaths. This filtered data set is then counted using the **len()** function and the number of days that this condition is met is displayed.

```
In [9]: len(df[df['deathIncrease'] > 20])
```

```
Out[9]: 63
```

Converting to Data Time

The below cell is responsible for converting the dates that are supplied in the .csv file into *Numpy datetime64*, which is a syntax for dates that allows for vectorized functions to be applied to the dates as well as abbreviations being used for things such as weeks and months.

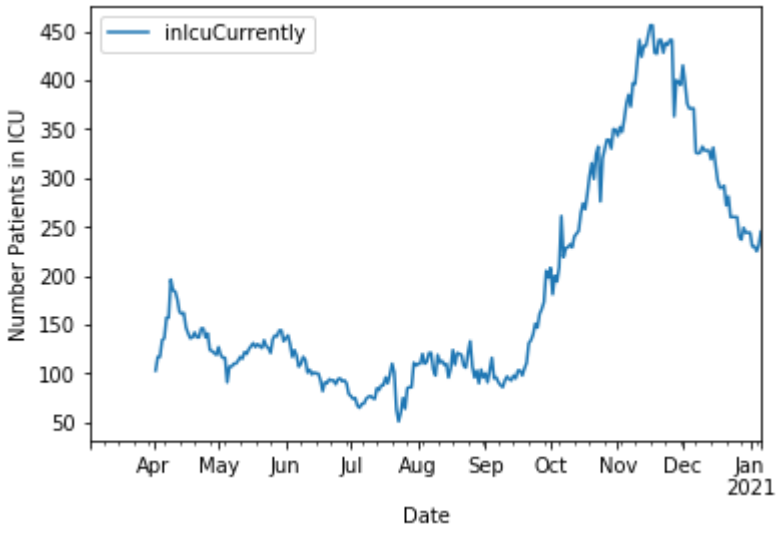
```
In [10]: df['date'] = pd.to_datetime(df['date'])
```

Number of Patients in ICU during the Pandemic

The below cell is responsible for plotting the **df** column **inIcuCurrently**, which represents the number of Covid-19 related patients there are in the ICU on a given day, and displaying that data for everyday that is avaible in the data set.

In [11]:
```
ax = df.plot(x='date', y='inIcuCurrently')
ax.set_xlabel('Date')
ax.set_ylabel('Number Patients in ICU')
```

Out[11]:
```
Text(0, 0.5, 'Number Patients in ICU')
```



Re-sampling Data by Week

The below cell is responnsible for taking the date frame **df**, and finding the mean of the data contained in **df** for each week of the data frame, and returning a new data frame instance.

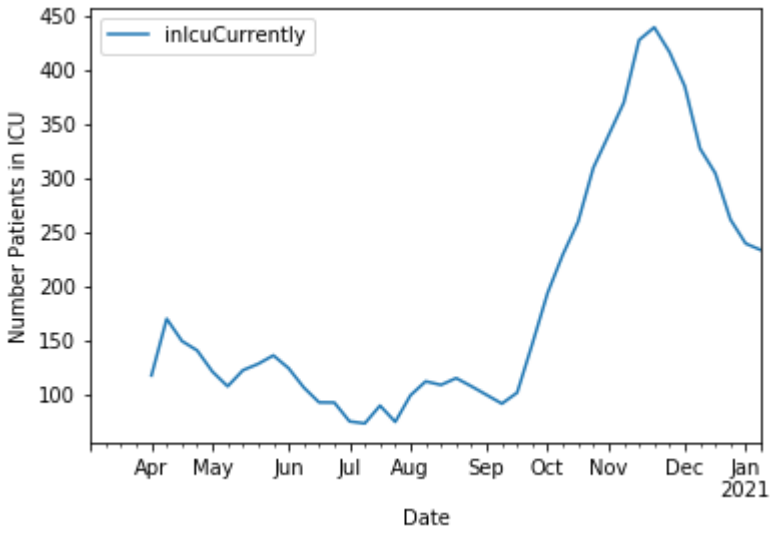
In [12]:
```
df_weekly_resample = df.resample('W', on='date').mean()
```

Plotting Weekly Re-sample

The below cell is responsible for plotting the data from **inIcuCurrently** that has been re-sampled according to a weekly basis, and displays this data for every week that is avaible in the data frame **df_weekly_resample**.

In [13]:
```
ax = df_weekly_resample.plot(y='inIcuCurrently')
ax.set_xlabel('Date')
ax.set_ylabel('Number Patients in ICU')
```

Out[13]: Text(0, 0.5, 'Number Patients in ICU')



Re-sampling Data by Month

The below cell is responnsible for taking the date frame **df**, and finding the mean of the data contained in **df** for each month of the data frame, and returning a new data frame instance.

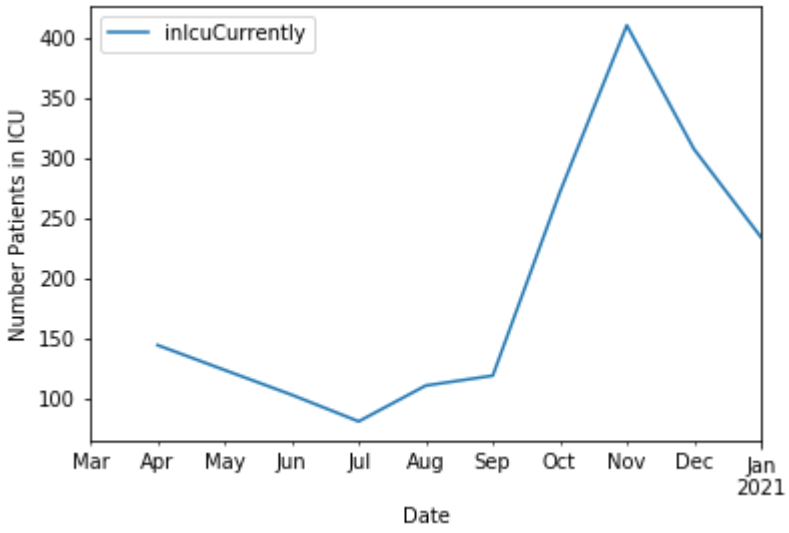
In [14]:
```
df_monthly_resample = df.resample('M', on='date').mean()
```

Plotting Monthly Re-sample

The below cell is responsible for plotting the data from **inIcuCurrently** that has been re-sampled according to a monthly basis, and displays this data for every month that is avaible in the data frame **df_monthly_resample**.

In [15]:
```
ax = df_monthly_resample.plot(y='inIcuCurrently')
ax.set_xlabel('Date')
ax.set_ylabel('Number Patients in ICU')
```

Out[15]: Text(0, 0.5, 'Number Patients in ICU')



Re-sample Observations

After resampling the data according to weeks and months some clear observations can be seen. The first trend seen is that as time moved from the beginning of the pandemic towards the summertime months, the number of patients in the ICU generally decreased. However, as 2020 moved out of summer and into the fall months, the number of patients in the ICU increased and increased until it reached a maximum value that took place in the month of November. Lastly, it can be seen that after that peak, that the number of patients in the ICU is trending downwards for the remainder of the data set.

Pearson Correlation Coefficient

The below cell is responsible for using the pandas function **.corr()** which uses the Pearson correlation coefficient to return correlation coefficient values for each combination of columns in **df**. A value of 1 means that the two columns represents a perfect linear relationship where as one increases so does the other. A value of 0 means that no linear relationship can be described between the two columns. Lastly, a value of -1 means that two columns represents a perfectly inverse linear relationship, whereas if one increases the other decreases.

In [16]:
```
df.corr().unstack().sort_values(ascending=False).head()
```

Out[16]:
```
totalTestsViralIncrease          totalTestsViralIncrease            1.0
totalTestsPeopleViral            totalTestsPeopleViral              1.0
totalTestResults                 totalTestResults                   1.0
                                 totalTestEncountersViral           1.0
totalTestEncountersViralIncrease totalTestEncountersViralIncrease   1.0
dtype: float64
```

Correlation Observations

When looking at the complete set of correlation values the first obvious observation is the fact that every column is compared to itself and as such there many instances of a value of 1 for correlation coefficients. The only correlation coefficient of 1.0 that was not a column being compared to itself, was the comparison of **totalTestResultsIncrease** and **totalTestEncountersViralIncrease**. My interpretation of this is that the only tests that were being conducted for Covid-19 were viral, and as such each positive test could also be described as a positive viral increase as well. The other near correlation values of 1.0 were columns that dealt with similar measures, such as **death** and **deathConfirmed**, which makes sense because using intuition it is obvious that as one increases the other will increase as well. No perfect inverse correlations were noted, the highest negative correlation coefficient found was -0.202, which is a very loose correlation and as such does not demonstrate a correlation of interest.

Vectorized Correlation for Shifted Days: 1st Variable Set

The below cell is responsible for using a vectorized approach to finding the correlation between the columns **hospitalizedCurrently**, which is the number of people hospitilized due to Covid-19 on a given day, and **InIcuCurrently**, which is the number of people in the ICU due to Covid-19 on a given day, when **InIcuCurrently** is offset from -20 days to 20 days.

In [17]:
```
%%time
vectorized_lag_corr = [crosscorr(df.hospitalizedCurrently, df.inIcuCurrently,
lag) for lag in range(-20,20)]
```

```
CPU times: user 15.9 ms, sys: 3.99 ms, total: 19.9 ms
Wall time: 18.9 ms
```

Highest Correlation Value for Vecotrized Shift: 1st Variable Set

The below cell is responsible for calling the function **print_max_corr** on the list **vectorized_lag_corr**, so that the highest correlation can be displayed from the list as well as at what offset of days did the highest correlation take place.

In [18]:
```
print_max_corr(vectorized_lag_corr, 40)
```

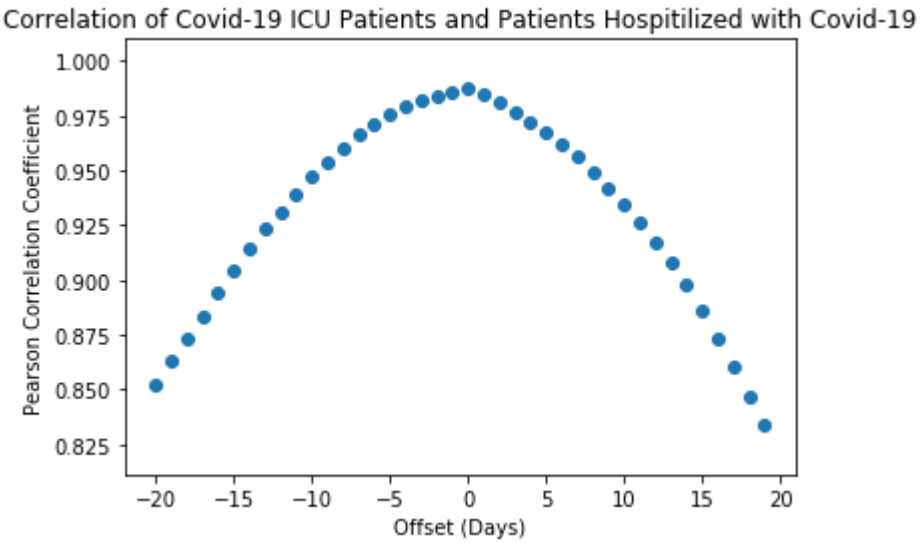
```
The highest correlation is:
0.9873264026693515
which occurs when days are offset by:
0.0
```

Plotting Correlation vs. Offset: Vectorized 1st Variable Set

The below cell is responsible for plotting the Pearson correlation coefficients from the list **vectorized_lag_corr** against the number of days that **InIcuCurrently** was offset by when compared to **hospitalizedCurrently**. Specifically, the plot below is displaying the correlations for the offsets from -20 days to +20 days.

In [19]:
```
plt.scatter(range(-20,20), vectorized_lag_corr)
plt.title('Correlation of Covid-19 ICU Patients and Patients Hospitilized with Covid-19')
plt.xlabel('Offset (Days)')
plt.ylabel('Pearson Correlation Coefficient')
```

Out[19]:
```
Text(0, 0.5, 'Pearson Correlation Coefficient')
```



Iterated Correlation for Shifted Days: 1st Variable Set

The below cell is responsible for using a iterated approach to finding the correlation between the columns **hospitalizedCurrently** and **InIcuCurrently**, when **InIcuCurrently** is offset from -20 days to 20 days. To accomplish this the function **iterated_crosscorr()** is used to return a list of the correlations between the aforementioned columns.

In [20]:
```
%time iterated_lag_corr = iterated_crosscorr(df, 'inIcuCurrently', 'hospitaliz
edCurrently', 20)
```

```
CPU times: user 762 ms, sys: 74 µs, total: 762 ms
Wall time: 762 ms
```

Highest Correlation Value for Iterated Shift: 1st Variable Set

The below cell is responsible for calling the function **print_max_corr** on the list **iterated_lag_corr**, so that the highest correlation can be displayed from the list as well as at what offset of days did the highest correlation take place.

In [21]:
```
print_max_corr(iterated_lag_corr, 40)
```

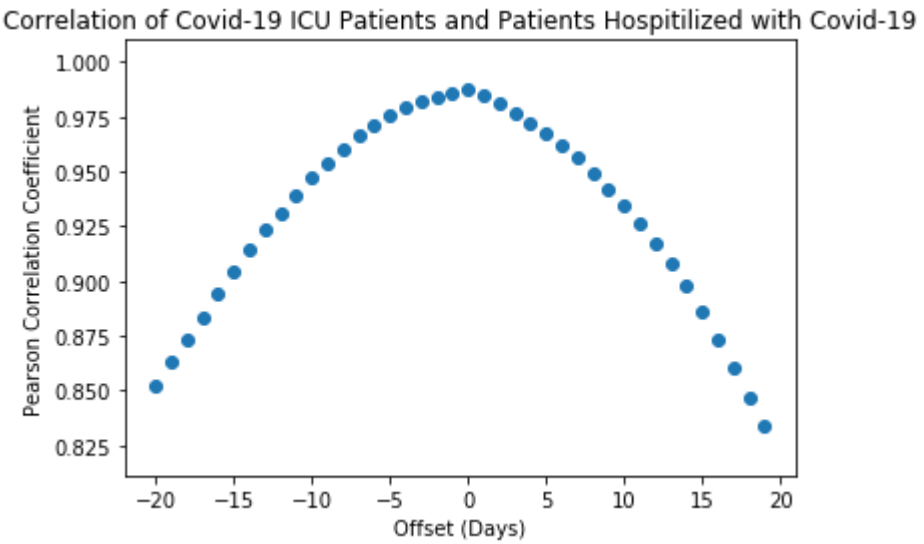
```
The highest correlation is:
0.9873264026693515
which occurs when days are offset by:
0.0
```

Plotting Correlation vs. Offset: Iterated 1st Variable Set

The below cell is responsible for plotting the Pearson correlation coefficients from the list **iterated_lag_corr** against the number of days that **InIcuCurrently** was offset by when compared to **hospitalizedCurrently**. Specifically, the plot below is displaying the correlations for the offsets from -20 days to +20 days.

In [22]:
```
plt.scatter(range(-20,20), iterated_lag_corr)
plt.title('Correlation of Covid-19 ICU Patients and Patients Hospitilized with
Covid-19')
plt.xlabel('Offset (Days)')
plt.ylabel('Pearson Correlation Coefficient')
```

Out[22]:
```
Text(0, 0.5, 'Pearson Correlation Coefficient')
```



Vectorized Correlation for Shifted Days: 2nd Variable Set

The below cell is responsible for using a vectorized approach to finding the correlation between the columns **positiveIncrease**, which is the daily increase in positive Covid-19 tests, and **deathIncrease**, which is the increase in Covid-19 related deaths per day, when **deathIncrease** is offset from -100 days to +100 days.

In [23]:
```
%%time
vectorized_lag_corr2 = [crosscorr(df.positiveIncrease, df.deathIncrease, lag)
for lag in range(-100,100)]
```

```
CPU times: user 70 ms, sys: 0 ns, total: 70 ms
Wall time: 68.9 ms
```

Highest Correlation Value for Vecotrized Shift: 2nd Variable Set

The below cell is responsible for calling the function **print_max_corr** on the list **vectorized_lag_corr2**, so that the highest correlation can be displayed from the list as well as at what offset of days did the highest correlation take place.

In [24]:
```
print_max_corr(vectorized_lag_corr2, 200)
```

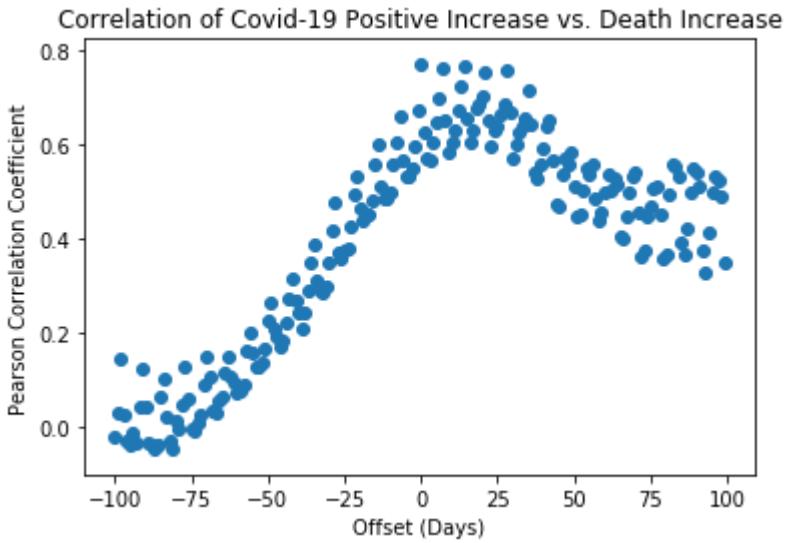
```
The highest correlation is:
0.7687842364762173
which occurs when days are offset by:
0.0
```

Plotting Correlation vs. Offset: Vectorized 2nd Variable Set

The below cell is responsible for plotting the Pearson correlation coefficients from the list **vectorized_lag_corr2** against the number of days that **positiveIncrease** was offset by when compared to **hospitalizedCurrently**. Specifically, the plot below is displaying the correlations for the offsets from -100 days to +100 days.

In [25]:
```
plt.scatter(range(-100, 100), vectorized_lag_corr2)
plt.title('Correlation of Covid-19 Positive Increase vs. Death Increase')
plt.xlabel('Offset (Days)')
plt.ylabel('Pearson Correlation Coefficient')
```

Out[25]:
```
Text(0, 0.5, 'Pearson Correlation Coefficient')
```



Iterated Correlation for Shifted Days: 2nd Variable Set

The below cell is responsible for using a iterated approach to finding the correlation between the columns **positiveIncrease**, which is the daily increase in positive Covid-19 tests, and **deathIncrease**, which is the increase in Covid-19 related deaths per day, when **deathIncrease** is offset from -100 days to 100 days. To accomplish this the function **iterated_crosscorr()** is used to return a list of the correlations between the aforementioned columns.

In [26]:
```
%time iterated_lag_corr2 = iterated_crosscorr(df, 'deathIncrease', 'positiveIn
crease', 100)
```

```
CPU times: user 3.72 s, sys: 3.56 ms, total: 3.73 s
Wall time: 3.73 s
```

Highest Correlation Value for Vecotrized Shift: 2nd Variable Set

The below cell is responsible for calling the function **print_max_corr** on the list **iterated_lag_corr2**, so that the highest correlation can be displayed from the list as well as at what offset of days did the highest correlation take place.

In [27]:
```
print_max_corr(iterated_lag_corr2, 200)
```

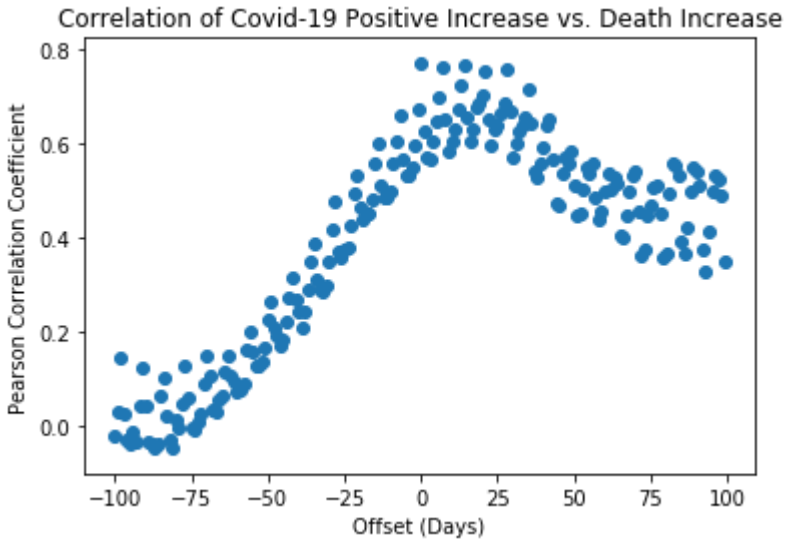
```
The highest correlation is:
0.7687842364762174
which occurs when days are offset by:
0.0
```

Plotting Correlation vs. Offset: Vectorized 2nd Variable Set

The below cell is responsible for plotting the Pearson correlation coefficients from the list **iterated_lag_corr2** against the number of days that **positiveIncrease** was offset by when compared to **hospitalizedCurrently**. Specifically, the plot below is displaying the correlations for the offsets from -100 days to +100 days.

In [28]:
```
plt.scatter(range(-100,100), iterated_lag_corr2)
plt.title('Correlation of Covid-19 Positive Increase vs. Death Increase')
plt.xlabel('Offset (Days)')
plt.ylabel('Pearson Correlation Coefficient')
```

Out[28]:
```
Text(0, 0.5, 'Pearson Correlation Coefficient')
```



Analysis of Vectorized vs. Iterated Appproach

For the first variable set, the vectorized approach takes approximately *16 ms* to gather all of the Pearson correlation coefficients for the data, while the iterated approach took approximately *758 ms* to do the equivalent operation. For the second variable set the vectorized approach took approximately *70 ms* to gather all of the Pearson correlation coefficients for the data, while the iterated approach took approximately *3.7 s*. To sum up the benchmarking of these two variables sets, it was demonstrated that the vectorization approach to finding Pearson correlation coefficients for two data frame columns when shifting the number of days was approximately **50 times** faster than the same operation for the iterated approach.

Conclusion

This notebook's purpose was to leverage pandas in order to explore data from the Covid-19 pandemic in the state of Wisconsin. Throughout this notebook publicly available data sets of Covid-19 data is cleaned, filtered, and visualized in order to lend a better understanding of the data than the original .csv can convey. In this notebook it was discovered that when re-sampling data according to weeks and months it was seem that the number of patients in the ICU on a given day, **inIcuCurrently**, was at its lowest towards the end of June and the beginning of July, and peaked in the beginning of November. In addition, the Pearson Correlation Coefficient was used with a vectorized approach and an iterated approach in order to explore correlation in the data as well as the runtime differences between the two approaches. Both approaches found that when the dates were offset to find how many days yielded the highest correlation between **inIcuCurrently** and **hospitalizedCurrently** that the highest correlation found was approximately **0.987326402669** with an offset of **0** days. In addition, for the same correlation experiment ran between **positiveIncrease** and **deathIncrease** both approaches found that the highest correlation was approximately **0.768784236476** with an offset of **0** days. Through benchmarking these two variables sets with both the vectorized and iterated approach, it was discovered that the vectorized approach offers a significant runtime advantage where in testing it was approximately **50 times** faster the iterated approach.