Table of Contents

1 Predicting Covid Cases From Google Search Data: A Time Series Analysis

2 Business and Data Understanding

2.1 Business Problem

The city of Philadelphia has been tracking its COVID-19 cases since the beginning of the pandemic. They want to know if they can use Google's COVID-19 related search data to better predict COVID-19 rates. If a model to give advanced warning could be made it would give the city of Philadelphia time to be able to prepare for spikes in COVID-19 cases and increase preventative measures (mask mandates, encouraging social distancing, warn hospitals, etc.) when needed.

2.2 Data

In order to do this analysis, there were two data sets that we put together.

The independent variables were taken from [Google's Explore COVID-19 Symptoms Search Trends (https://pair-code.github.io/covid19_symptom_dataset/?country=IE)](https://pair-code.github.io/covid19_symptom_dataset/?country=IE). The data was downloaded from the

USA region (sub region of Pennsylvania) at the daily resolution. All of the data from January 1st, 2020 through November 11th, 2022 was then collated into one data frame, containing 68,805 rows and 430 columns. This data had all been scaled and normalized (https://storage.googleapis.com/gcp-public-data-symptom-search/COVID-19%20Search%20Trends%20symptoms%20dataset%20documentation%20.pdf) prior to being downloaded.

The target variable was taken from COVID-19 Data for Pennsylvania (https://www.health.pa.gov/topics/disease/coronavirus/pages/Cases.aspx). This data spanned from March 1st, 2020 until March 14, 2023 and included 75,412 rows and 12 columns.

2.2.1 Limitations of the dataset

While these two datasets do include a fairly comprehensive list of the search terms and COVID-19 case counts, they do not include all of the possible elements relevant to the rise and fall of COVID-19 cases - for example they don't take into account the proliferation of novel versions of the virus (e.g. Delta, Omicron, etc.). The dataset is also limited by time - COVID-19 was only proclaimed a pandemic by the World Health Organization on March 11th, 2020 (https://www.yalemedicine.org/news/covid-timeline). As such, it could be that better predictions will be available as more time passes, allowing for more data to be collected.

2.2.2 Why We Used This Dataset

As the initial inquiry was about if search trends could be used to predict COVID-19, we felt that these data sets were a perfect place to start! Google is a leading search engine, so it seemed an intuitive place to collect search data from. All of the acquired data was free and publicly available.

2.2.3 Dataset Size

Initially, we had two datasets, one with **68,805** rows and **430** columns, the other with **75,412** rows and **12** columns. After subsetting these datasets to include **only the Philadelphia region**, cleaning the data, and matching the dates of the datasets, we had **991** rows (representing from March 8th, 2020 to November 13th, 2022) and **423** columns.

3 Initial Look at the Data

3.1 Imports

In [1]:

```python
# imports
from statsmodels.tsa.statespace.varmax import VARMAX
from statsmodels.tsa.api import VAR
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import pearsonr
import statsmodels.api as sm
import pmdarima as pm
import seaborn as sns
import pandas as pd
import numpy as np
from numpy import log
from datetime import datetime, timedelta
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')
```

In [2]:

```python
# bringing in our 4 dataframes

public_data = pd.read_csv(
    "data/COVID-19_Aggregate_Cases_Current_Weekly_County_Health.csv")
google_data_2020 = pd.read_csv(
    "data/2020_sub_region_1_daily_2020_US_Pennsylvania_daily_symptoms_dataset
google_data_2021 = pd.read_csv(
    "data/2021_sub_region_1_daily_2021_US_Pennsylvania_daily_symptoms_dataset
google_data_2022 = pd.read_csv(
    "data/2022_sub_region_1_daily_2022_US_Pennsylvania_daily_symptoms_dataset
```

3.2 Data Preprocessing

3.2.1 Google Search Data

Let's start by consolidating all of our separate Google files into one data frame.

In [3]:

```python
# consolidating the google data into one dataframe
google_dataframes = [google_data_2020, google_data_2021, google_data_2022]
google_data = pd.concat(google_dataframes)
google_data.head()
```

Out[3]:

	country_region_code	country_region	sub_region_1	sub_region_1_code	sub_region_2	sub_region_2_code	
0	US	United States	Pennsylvania	US-PA	NaN	NaN	C
1	US	United States	Pennsylvania	US-PA	NaN	NaN	C
2	US	United States	Pennsylvania	US-PA	NaN	NaN	C
3	US	United States	Pennsylvania	US-PA	NaN	NaN	C
4	US	United States	Pennsylvania	US-PA	NaN	NaN	C

5 rows × 430 columns

In [4]:
```
# looking at the size and makeup of our dataframe
google_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 68805 entries, 0 to 20826
Columns: 430 entries, country_region_code to symptom:pancreatitis
dtypes: object(430)
memory usage: 226.2+ MB
```

Now, let's extract *only* the data relevant to Philadelphia and clean up our dataset a bit.

In [5]:
```
google_philly = google_data[google_data.sub_region_2 == 'Philadelphia County'
google_philly.head()
```

Out[5]:

	country_region_code	country_region	sub_region_1	sub_region_1_code	sub_region_2	sub_region_2_code
17934	US	United States	Pennsylvania	US-PA	Philadelphia County	4210
17935	US	United States	Pennsylvania	US-PA	Philadelphia County	4210
17936	US	United States	Pennsylvania	US-PA	Philadelphia County	4210
17937	US	United States	Pennsylvania	US-PA	Philadelphia County	4210
17938	US	United States	Pennsylvania	US-PA	Philadelphia County	4210

5 rows × 430 columns

In [6]:

```python
# removing extraneous columns from our `google_data`
google_philly.drop(columns=["sub_region_2", "sub_region_2_code", "country_reg
                            "sub_region_1_code", "place_id"], inplace=True)

google_philly.head()
```

```
/var/folders/nz/h8wmnpz55qb3srn4_mj451lc0000gn/T/ipykernel_6058/3665995247.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  google_philly.drop(columns=["sub_region_2", "sub_region_2_code", "country_regi
on_code", "country_region", "sub_region_1",
```

Out[6]:

	date	symptom:Abdominal obesity	symptom:Abdominal pain	symptom:Acne	symptom:Actinic keratosis	symptom:Acute bronchitis
17934	2020-01-01	2.35	4.46	7.76	0.19	0.62
17935	2020-01-02	2.37	4.17	8.13	0.24	0.66
17936	2020-01-03	2.13	4.18	7.57	0.24	0.78
17937	2020-01-04	2.37	4.5	8.85	0.16	0.69
17938	2020-01-05	2.36	4.12	8.58	0.07	0.62

5 rows × 423 columns

In [7]:

```python
# make the date our index, then sort by the date index
google_philly['date'] = pd.to_datetime(google_philly['date'])
google_philly.set_index('date', inplace=True)
google_philly = google_philly.sort_index()
```

```
/var/folders/nz/h8wmnpz55qb3srn4_mj451lc0000gn/T/ipykernel_6058/3499777103.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  google_philly['date'] = pd.to_datetime(google_philly['date'])
```

In [8]:
```
google_philly.head()
```
Out[8]:

date	symptom:Abdominal obesity	symptom:Abdominal pain	symptom:Acne	symptom:Actinic keratosis	symptom:Acute bronchitis	sympt
2020-01-01	2.35	4.46	7.76	0.19	0.62	
2020-01-02	2.37	4.17	8.13	0.24	0.66	
2020-01-03	2.13	4.18	7.57	0.24	0.78	
2020-01-04	2.37	4.5	8.85	0.16	0.69	
2020-01-05	2.36	4.12	8.58	0.07	0.62	

5 rows × 422 columns

In [9]:
```
# changing all our variable types from objects to float
google_philly = google_philly.astype('float')
```
In [10]:
```
# https://stackoverflow.com/questions/52044348/check-for-any-missing-dates-in

# checking for any missing dates!
pd.date_range(start='2020-01-01',
              end='2022-11-13').difference(google_philly.index)
```
Out[10]:
```
DatetimeIndex([], dtype='datetime64[ns]', freq='D')
```
In [11]:
```
# check for missing values in `google_philly`
google_philly.isnull().sum().sum()
```
Out[11]:
```
3176
```
In [12]:
```
# checking for 0 in the whole data frame - we don't find any, so lets use 0 a
(google_philly == 0).any().sum()
```
Out[12]:
```
0
```
In [13]:
```
# turning our NaN's to 0's
google_philly.fillna(0, inplace=True)
```
In [14]:
```
# checking for NaN's after our transformation
google_philly.isnull().sum().sum()
```
Out[14]:
```
0
```
In [15]:
```
# code from https://stackoverflow.com/questions/55679401/remove-prefix-or-suf

# removing 'symptom:' prefix from all our symptoms
google_philly.columns = google_philly.columns.map(
    lambda x: x.removeprefix('symptom:'))
```

In [16]:

```
google_philly.head()
```

Out[16]:

date	Abdominal obesity	Abdominal pain	Acne	Actinic keratosis	Acute bronchitis	Adrenal crisis	Ageusia	Alcoholism	Allergic conjunctivitis
2020-01-01	2.35	4.46	7.76	0.19	0.62	0.09	0.00	5.11	0.00
2020-01-02	2.37	4.17	8.13	0.24	0.66	0.05	0.00	4.35	0.00
2020-01-03	2.13	4.18	7.57	0.24	0.78	0.07	0.06	4.01	0.09
2020-01-04	2.37	4.50	8.85	0.16	0.69	0.09	0.00	4.38	0.09
2020-01-05	2.36	4.12	8.58	0.07	0.62	0.07	0.00	3.82	0.08

5 rows × 422 columns

In [17]:

```
google_philly.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1048 entries, 2020-01-01 to 2022-11-13
Columns: 422 entries, Abdominal obesity to pancreatitis
dtypes: float64(422)
memory usage: 3.4 MB
```

3.2.2 Public COVID-19 PA Data

In [18]:

```
# start processing the public_data - make the `Date` the index
public_data['Date'] = pd.to_datetime(public_data['Date'])
public_data.set_index('Date', inplace=True)
public_data = public_data.sort_index()
```

In [19]:
```
# looking at the data
public_data
```

Out[19]:

Date	Jurisdiction	New Cases	7-day Average New Cases	Cumulative cases	Population (2019)	New Case Rate	7-Day Average New Case Rate	Cumulative Case Rate	County FIPS Code	Long
2020-03-01	Lancaster	0	NaN	0	545724	0.0	NaN	0.0	42071	-76.25
2020-03-01	Blair	0	NaN	0	121829	0.0	NaN	0.0	42013	-78.34
2020-03-01	Susquehanna	0	NaN	0	40328	0.0	NaN	0.0	42115	-75.80
2020-03-01	Berks	0	NaN	0	421164	0.0	NaN	0.0	42011	-75.93
2020-03-01	Clinton	0	NaN	0	38632	0.0	NaN	0.0	42035	-77.64
...	...	...	...	...	...	...	...	...	...	
2023-03-14	Dauphin	15	16.6	72765	278299	5.4	6.0	26146.3	42043	-76.77
2023-03-14	Crawford	14	12.3	25018	84629	16.5	14.5	29562.0	42039	-80.11
2023-03-14	Venango	4	4.4	13866	50668	7.9	8.7	27366.4	42121	-79.76
2023-03-14	Luzerne	27	23.6	95034	317417	8.5	7.4	29939.8	42079	-75.99
2023-03-14	Lancaster	48	37.1	151152	545724	8.8	6.8	27697.5	42071	-76.25

75412 rows × 12 columns

In [20]:

```python
# subsetting the data to only include Philadelphia
public_philly = public_data[public_data.Jurisdiction == 'Philadelphia']
# making it so the dates will match when we join the two dataframes
public_philly = public_philly.loc[: "2022-11-16"]
# looking at the data again
public_philly
```

Out[20]:

Date	Jurisdiction	New Cases	7-day Average New Cases	Cumulative cases	Population (2019)	New Case Rate	7-Day Average New Case Rate	Cumulative Case Rate	County FIPS Code	Longit
2020-03-01	Philadelphia	0	NaN	0	1584064	0.0	NaN	0.0	42101	-75.140
2020-03-02	Philadelphia	0	NaN	0	1584064	0.0	NaN	0.0	42101	-75.140
2020-03-03	Philadelphia	0	NaN	0	1584064	0.0	NaN	0.0	42101	-75.140
2020-03-04	Philadelphia	0	NaN	0	1584064	0.0	NaN	0.0	42101	-75.140
2020-03-05	Philadelphia	0	NaN	0	1584064	0.0	NaN	0.0	42101	-75.140
...	...	...	...	...	...	...	...	...	...	
2022-11-12	Philadelphia	116	159.0	372093	1584064	7.3	10.0	23489.8	42101	-75.140
2022-11-13	Philadelphia	78	157.1	372171	1584064	4.9	9.9	23494.7	42101	-75.140
2022-11-14	Philadelphia	165	158.1	372336	1584064	10.4	10.0	23505.1	42101	-75.140
2022-11-15	Philadelphia	171	152.7	372507	1584064	10.8	9.6	23515.9	42101	-75.140
2022-11-16	Philadelphia	203	153.7	372710	1584064	12.8	9.7	23528.7	42101	-75.140

991 rows × 12 columns

In [21]:
```
# no NaN's that we have to deal with
public_philly.isnull().sum()
```

Out[21]:
```
Jurisdiction                        0
New Cases                           0
7-day Average New Cases             7
Cumulative cases                    0
Population (2019)                   0
New Case Rate                       0
7-Day Average New Case Rate         7
Cumulative Case Rate                0
County FIPS Code                    0
Longitude                           0
Latitude                            0
Georeferenced Lat & Long            0
dtype: int64
```

3.2.3 Joining the Google and Public COVID-19 Data

In [22]:
```
google_philly = google_philly.loc["2020-03-01": "2022-11-16"]
```

After trying both the `New Cases` and the `7-day Average New Cases` as the target variable, the `7-day Average New Cases` was chosen - and after running both models, it had a lower error rate according to all metrics (RMSE, MAE, MAPE).

In [23]:
```
# join data sets and decide on target variable -drop the first 7 days with mi
philly_data = pd.concat(
    [google_philly, public_philly['7-day Average New Cases']], axis=1).dropna
# '7-day Average New Cases' is too long - I renamed it as 'Target' instead
philly_data.rename(columns={'7-day Average New Cases': 'Target'}, inplace=Tru
target = philly_data['Target']
symptoms = philly_data.drop(['Target'], axis=1)
```

In [24]:
```
symptoms
```

Out[24]:

	Abdominal obesity	Abdominal pain	Acne	Actinic keratosis	Acute bronchitis	Adrenal crisis	Ageusia	Alcoholism	Allergic conjunctivitis	A
2020-03-08	2.55	3.95	8.81	0.22	0.45	0.09	0.00	3.60	0.00	
2020-03-09	2.09	4.02	7.45	0.23	0.62	0.12	0.05	3.64	0.09	
2020-03-10	2.10	3.88	7.47	0.20	0.65	0.09	0.06	3.51	0.09	
2020-03-11	1.97	3.76	7.01	0.20	0.76	0.09	0.04	3.59	0.07	
2020-03-12	1.42	3.44	6.45	0.19	0.72	0.08	0.04	3.40	0.11	
...	...	...	...	...	...	...	...	...	...	
2022-11-09	1.34	3.44	6.12	0.29	0.62	0.07	0.09	2.89	0.08	
2022-11-10	1.41	3.45	6.24	0.21	0.59	0.14	0.07	3.18	0.07	
2022-11-11	1.33	3.56	6.30	0.24	0.47	0.10	0.10	3.46	0.04	
2022-11-12	1.42	3.68	6.82	0.22	0.45	0.10	0.06	3.54	0.07	
2022-11-13	1.51	3.81	6.93	0.22	0.48	0.07	0.06	3.45	0.07	

981 rows × 422 columns

In [25]:
```
philly_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 981 entries, 2020-03-08 to 2022-11-13
Freq: D
Columns: 423 entries, Abdominal obesity to Target
dtypes: float64(423)
memory usage: 3.2 MB
```

In [26]:
```
philly_data.describe()
```

Out[26]:

	Abdominal obesity	Abdominal pain	Acne	Actinic keratosis	Acute bronchitis	Adrenal crisis	Ageusia	Alcoholism	conju
count	981.000000	981.000000	981.000000	981.000000	981.000000	981.000000	981.000000	981.000000	981
mean	2.126891	3.670204	7.592018	0.251407	0.276901	0.087238	0.135984	3.500499	0
std	0.512854	0.313704	0.989178	0.053219	0.122890	0.033182	0.093180	0.409302	0
min	0.960000	2.590000	5.340000	0.110000	0.000000	0.000000	0.000000	2.600000	0
25%	1.750000	3.500000	6.850000	0.210000	0.190000	0.070000	0.070000	3.320000	0
50%	2.100000	3.640000	7.530000	0.250000	0.250000	0.090000	0.110000	3.460000	0
75%	2.500000	3.820000	8.160000	0.290000	0.350000	0.100000	0.170000	3.610000	0
max	3.730000	8.400000	12.340000	0.480000	0.760000	0.420000	0.720000	8.790000	0

8 rows × 423 columns

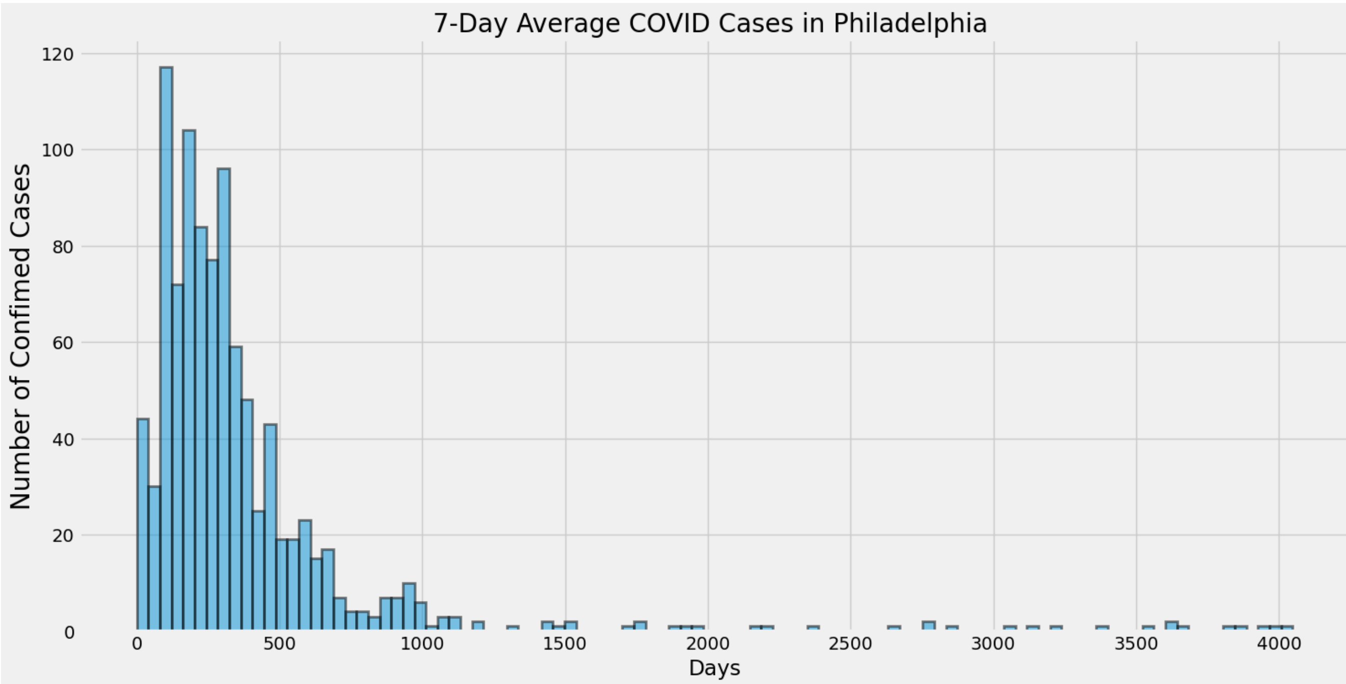
3.3 Initial Visualizations

In [27]:

```python
# looking at the distribution of our target variable
target.hist(bins=100, figsize=(16, 8), alpha=0.5,
            edgecolor="black", linewidth=2)
plt.title(
    '7-Day Average COVID Cases in Philadelphia ', fontsize=20)
plt.xlabel('Days')
plt.ylabel('Number of Confimed Cases', fontsize=20)
```

Out[27]:

```
Text(0, 0.5, 'Number of Confimed Cases')
```

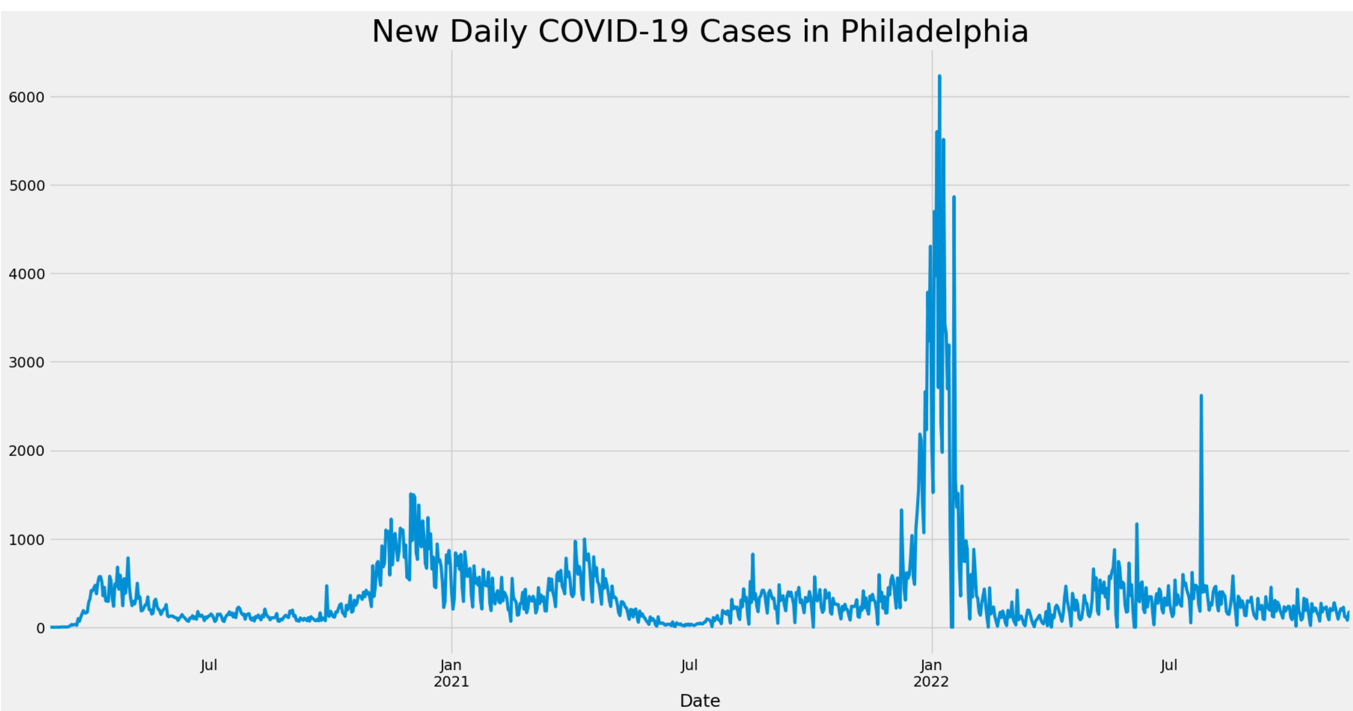


In [28]:

```python
# look at the raw data
ax = public_philly['New Cases'].plot(figsize=(20, 10), linewidth=3)
plt.title('New Daily COVID-19 Cases in Philadelphia', fontsize=30)
```

Out[28]:

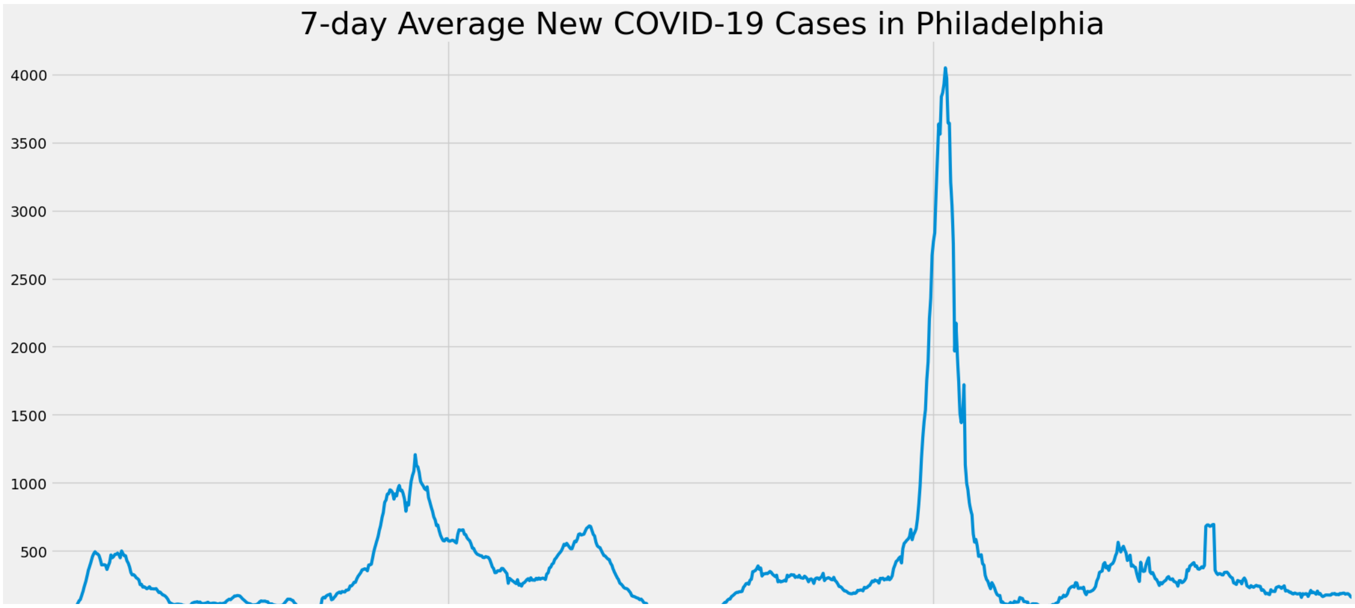
```
Text(0.5, 1.0, 'New Daily COVID-19 Cases in Philadelphia')
```



In [29]:

```
# Look at our target variable
ax = target.plot(figsize=(20, 10), linewidth=3)
plt.title('7-day Average New COVID-19 Cases in Philadelphia', fontsize=30)
```

Out[29]: Text(0.5, 1.0, '7-day Average New COVID-19 Cases in Philadelphia')



3.3.1 Test Train Split: Target Variable

In [30]:

```
# this will leave us with 20 days in our test set and 961 in our training set
cutoff_test = 20

X_train = symptoms[:-cutoff_test]
X_test = symptoms[-cutoff_test:]

y_train = target[:-cutoff_test]
y_test = target[-cutoff_test:]
```

4 Base Model of the Target Variable

Now that we've seen the target variable, lets check for stationarity.

In [31]:

```
# Dicky Fuller Test for Stationarity
def dicky_fuller(TS):
    """
    This function takes in a time series and evaluates it according to the Di
    """
    result = adfuller(TS)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
```

In [32]:

```python
# Dicky Fuller Test for Stationarity
dicky_fuller(y_train)
```

```
ADF Statistic: -4.014737
p-value: 0.001335
Critical Values:
	1%: -3.437
	5%: -2.865
	10%: -2.568
```

Our target data is stationary! Lets do a grid search for the optimal orders for our data.

In [33]:

```python
# origionally all of the max elements were set to 5, but that took 2+ hours t
def sarima_elements(TS):
    '''
This function will calculate the optimal elements to use in our model. The mo
this function requires both the time series and the order of elements in orde

An ARIMA model requires 3 elements - p, d, and q. This function will cycle th
to find the optimal values for these elements.

p - this is the order of the auto-regressive (AR) part of the ARIMA model. Th
incorporate into our model, thereby enabling our model to take the past into
d - integrated (I) part of the model, which states the order of
how many times the model has been differenced to find optimal  stationarity.
q - this is the moving average (MA) order of the ARIMA model, which represent

The seasonal ARIMA contains these three elements along with a seasonal elemen
the approximate number of weeks in a year.
    '''
    auto = pm.auto_arima(TS, start_p=4, start_q=0, max_p=7,
                         max_q=3, max_d=3, start_P=0, start_Q=0, max_P=3,
                         max_Q=3, m=52, max_order=None, stepwise=True,
                         trace=True, random_state=42)
    return auto
```

In [34]:

```python
# note: running this cell takes +- 20 min
target_elements = sarima_elements(y_train)
```

```
Performing stepwise search to minimize aic
 ARIMA(4,0,0)(0,0,0)[52] intercept   : AIC=10219.485, Time=0.51 sec
 ARIMA(0,0,0)(0,0,0)[52] intercept   : AIC=14671.621, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[52] intercept   : AIC=inf, Time=7.31 sec
 ARIMA(0,0,1)(0,0,1)[52] intercept   : AIC=13466.858, Time=8.89 sec
 ARIMA(0,0,0)(0,0,0)[52]             : AIC=15115.024, Time=0.01 sec
 ARIMA(4,0,0)(1,0,0)[52] intercept   : AIC=10221.237, Time=11.06 sec
 ARIMA(4,0,0)(0,0,1)[52] intercept   : AIC=10221.233, Time=7.69 sec
 ARIMA(4,0,0)(1,0,1)[52] intercept   : AIC=10222.968, Time=39.50 sec
 ARIMA(3,0,0)(0,0,0)[52] intercept   : AIC=10302.137, Time=0.25 sec
 ARIMA(5,0,0)(0,0,0)[52] intercept   : AIC=10200.762, Time=1.30 sec
 ARIMA(5,0,0)(1,0,0)[52] intercept   : AIC=10202.610, Time=38.50 sec
 ARIMA(5,0,0)(0,0,1)[52] intercept   : AIC=10202.609, Time=13.56 sec
 ARIMA(5,0,0)(1,0,1)[52] intercept   : AIC=10204.612, Time=18.28 sec
 ARIMA(6,0,0)(0,0,0)[52] intercept   : AIC=10172.628, Time=0.78 sec
 ARIMA(6,0,0)(1,0,0)[52] intercept   : AIC=10174.597, Time=31.50 sec
 ARIMA(6,0,0)(0,0,1)[52] intercept   : AIC=10174.598, Time=22.97 sec
 ARIMA(6,0,0)(1,0,1)[52] intercept   : AIC=10176.595, Time=27.15 sec
 ARIMA(7,0,0)(0,0,0)[52] intercept   : AIC=10167.093, Time=1.58 sec
 ARIMA(7,0,0)(1,0,0)[52] intercept   : AIC=10169.068, Time=43.86 sec
```

In [35]:

```python
# get the plots and summary about our model
target_elements.plot_diagnostics(figsize=(12, 8))
target_elements.summary()
```

Out[35]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	961
Model:	SARIMAX(7, 0, 3)	**Log Likelihood**	-5004.812
Date:	Mon, 01 May 2023	**AIC**	10033.623
Time:	08:23:37	**BIC**	10092.039
Sample:	0	**HQIC**	10055.868
	- 961		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	5.8144	9.004	0.646	0.518	-11.833	23.462
ar.L1	0.1649	0.032	5.160	0.000	0.102	0.228

In [36]:

```python
# getting the first and last date of the test data
last_date_test = y_test.last_valid_index()
first_date_test = y_test.first_valid_index()
print(f" First Date: {first_date_test}, Last Date: {last_date_test}")
```

```
First Date: 2022-10-25 00:00:00, Last Date: 2022-11-13 00:00:00
```

In [37]:

```python
# creating our baseline ARIMA model for our target data
ARIMA_MODEL = sm.tsa.statespace.SARIMAX((y_train), order=target_elements.orde
                                        seasonal_order=target_elements.season
                                        enforce_invertibility=False)

# Fit the model and return the results
output = ARIMA_MODEL.fit()

# forcast
output_forcast = output.forecast(steps=last_date_test)
```

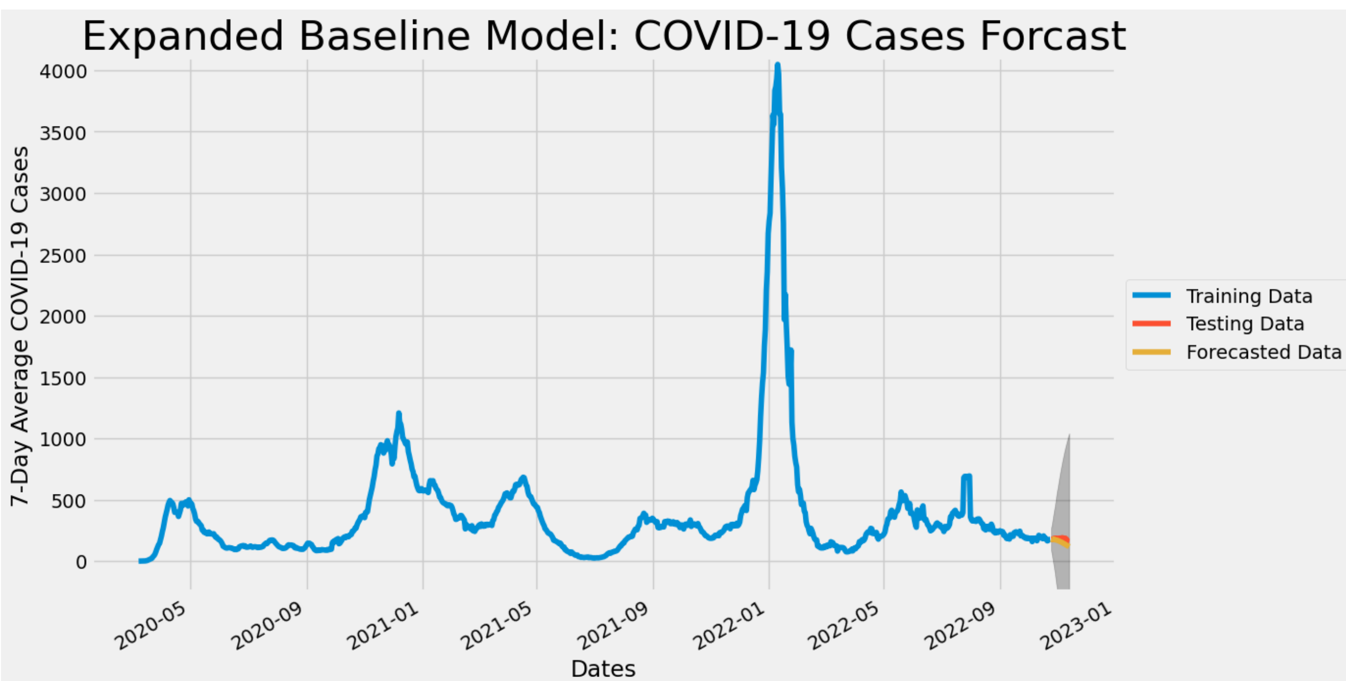
```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =           11     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.14749D+00    |proj g|=  2.52192D-01

At iterate    5    f=  5.67577D+00    |proj g|=  4.35944D-01

 This problem is unconstrained.


At iterate   10    f=  5.24885D+00    |proj g|=  1.98109D-01

At iterate   15    f=  5.19825D+00    |proj g|=  7.42093D-02

At iterate   20    f=  5.19459D+00    |proj g|=  4.95326D-02

At iterate   25    f=  5.19436D+00    |proj g|=  1.25846D-02

At iterate   30    f=  5.19434D+00    |proj g|=  1.33236D-03

At iterate   35    f=  5.19430D+00    |proj g|=  1.49973D-02

At iterate   40    f=  5.19267D+00    |proj g|=  9.81870D-02

At iterate   45    f=  5.18275D+00    |proj g|=  3.62561D-02

At iterate   50    f=  5.18034D+00    |proj g|=  1.46386D-02

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
   11     50     62      1     0     0   1.464D-02   5.180D+00
  F =   5.1803440634537745

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
```

```
/Users/rachelsanderlin/opt/anaconda3/envs/covid-env/lib/python3.9/site-packages/
statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimizati
on failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

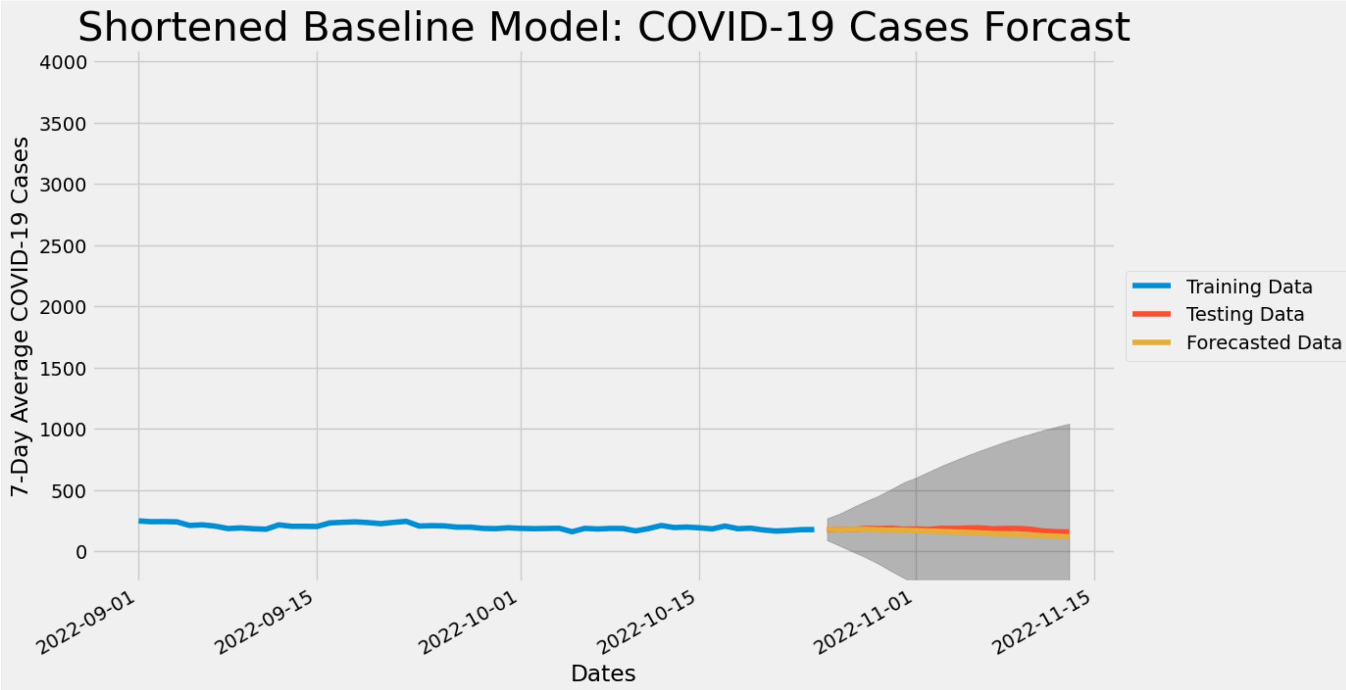
In [38]:
```python
# get the confidence interval for our model
output_conf = output.get_forecast(steps=last_date_test).conf_int()
```

In [39]:
```python
def plot_model(y_train, y_test, predicted_forcast, title, y_lable, conf_inter
    """
    The purpose of this function is to plot both the real and forcasted data
    It takes in the training data, the testing data, and the forcasted data.
    must be specified, and a confidence interval and cutoff date are optional

    """
    # plotting the prediction
    fig, ax = plt.subplots(figsize=(12, 8))
    # taking only a potion of the data to better show how the predicted data
    ax.plot(y_train[cutoff_date:], label='Training Data')
    ax.plot(y_test, label='Testing Data')
    predicted_forcast.plot(ax=ax, label='Forecasted Data')
    # plotting the confidence interval
    ax.fill_between(conf_interval.index,
                    conf_interval.iloc[:, 0],
                    conf_interval.iloc[:, 1], color='k', alpha=0.25)
    #set max, min amounts of y-axis
    ax.set_ylim([-250, 4100])
    ax.set_title(title, fontsize=30)
    ax.set_xlabel('Dates')
    ax.set_ylabel(y_lable)
    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

In [40]:
```python
plot_model(y_train, y_test, output_forcast, "Expanded Baseline Model: COVID-1
           '7-Day Average COVID-19 Cases', conf_interval=output_conf)
```



In [41]:
```
plot_model(y_train, y_test, output_forcast, "Shortened Baseline Model: COVID-
             '7-Day Average COVID-19 Cases', conf_interval=output_conf, cutoff_
```



Initially when this data was modeled, the model predicted a straight line. This led to the assumption that something was wrong with the model. We tried rerunning the parameters, subtracting the moving average from the target variable, and taking the log of the target data, but none of these actions significantly improved upon the model. The data was also run through Facebook Prophet in a different notebook (see [here (https://github.com/sanderlin2013/Predicting-COVID-19-in-Philly/blob/main/sandbox/alt_notebook_prophet.ipynb)]) and we tried modeling the data pre and post the Omicron spike seen around January 2022 in another notebook as well (see [here (https://github.com/sanderlin2013/Predicting-COVID-19-in-Philly/blob/main/sandbox/alt_notebook_remove_omicron.ipynb)]).

At the end of the day, none of these methods made a difference - the best model we could find used the 7 day average case counts, and the predictions more or less trended towards a straight line during their prediction. This apparently is not a unique occurrence in univariate SARIMA models - in the official [documentation on forecasting in statsmodels (https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_forecasting.html#Prediction-vs-Forecasting)] it states, specifically about a straight line SARIMAX forcast: "The forecast above may not look very impressive, as it is almost a **straight line**. This is because this is a very simple, univariate forecasting model. Nonetheless, *keep in mind that these simple forecasting models can be extremely competitive*."

For more information on this phenomenon, feel free to read more [here (https://github.com/statsmodels/statsmodels/issues/3852)].

Finally, it became clear that what was most affecting my predictions was that the model was trying to predict too far into the future. By reducing the test data from 20% of the dataset (almost 100 days) to 20 days, the model performance vastly improved.

Lets look at some loss functions to assess how well the model preformed.

In [42]:
```
# https://stackoverflow.com/questions/3308102/how-to-extract-the-n-th-element
forcast_date_list = list(output_forcast.items())
n = 1
forcast_list = [x[n] for x in forcast_date_list]
```

In [43]:
```
print('MAE:', np.mean(abs(forcast_list - y_test.values)))
print('RMSE:', np.sqrt(np.mean((forcast_list - y_test.values)**2)))
print('MAPE:', np.mean(abs((forcast_list - y_test.values)/y_test.values)))
```

```
MAE: 24.85583845359837
RMSE: 29.799693896902426
MAPE: 0.13903013173746206
```

Based on these error terms and looking at the plot of our model, we can see that our model isn't perfect, and like most time series performs worse (the confidence interval widens) the farther out we try to predict the data.

Lets see if we can improve on these predictions by using a **multivariate time series model**.

5 Multivariate Time Series Model

5.1 PCA

The first step to trying out multivariate models is figuring out what variables we would like to include in our model! In our dataset we have over 400 symptoms to consider from the Google dataset - using all of them would be too computationally expensive. Luckily, we can use Principal Component Analysis (PCA) for dimensionality reduction.

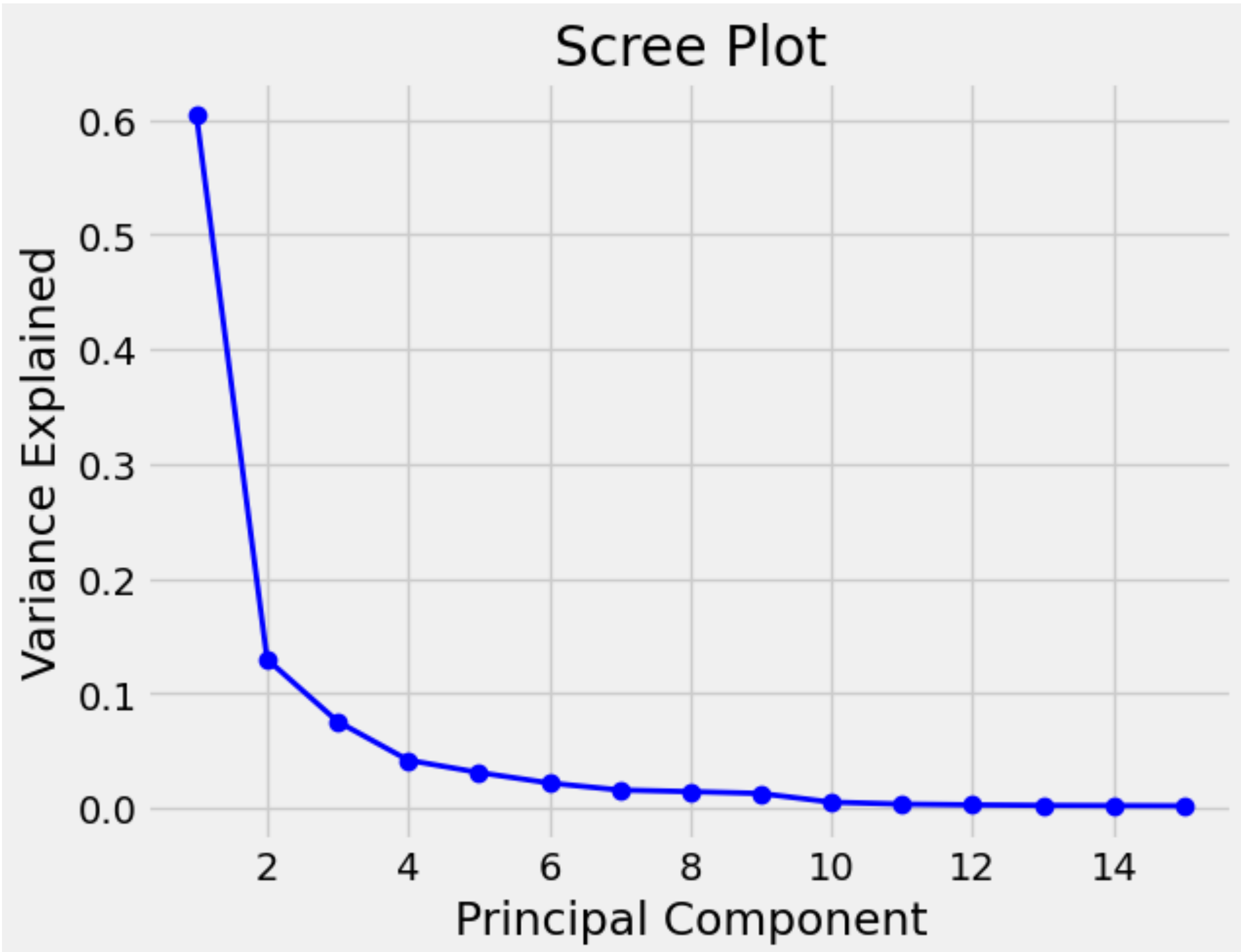
PCA involves transforming a large set of variables into a smaller set of uncorrelated variables while keeping most of the information in the original data. PCA works by identifying the principal components (PCs) of the data, which are linear combinations of the original variables that explain the most variance in the data. The first PC explains the most variance, followed by the second PC, and so on. By selecting only the top few PCs, we can reduce the dimensionality of the data while retaining most of the important information.

The goal was to reduce the dimensions of our data into a manageable number of components and then hopefully use those components to build a better but not overly complex Vector Autoregressive (VAR) model.

The first step in using PCA is figuring out how many components we would want the PCA to sort our data into.

In [44]:

```
# perform PCA on training data
pca = PCA(n_components=15).fit(X_train)
# plot scree plot
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_,
         'o-', linewidth=2, color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```



The number of components we're going to choose will be 2, based on where the scree plot drops off.

In [45]:

```
# restructuring our `y_train` data so we can add it to our VAR model later
df_y_train = pd.DataFrame(y_train)
df_y_train = df_y_train.reset_index(names=['date'])
df_y_train
```

Out[45]:

	date	Target
0	2020-03-08	0.1
1	2020-03-09	0.1
2	2020-03-10	0.4
3	2020-03-11	0.9
4	2020-03-12	1.1
...	...	...
956	2022-10-20	173.9
957	2022-10-21	165.0
958	2022-10-22	169.3
959	2022-10-23	176.4
960	2022-10-24	176.9

961 rows × 2 columns

In [46]:

```
# instantiating the PCA
pca = PCA(n_components=2)
# fitting and transforming the PCA on our training data
principalComponents = pca.fit_transform(X_train)
# dataframe with PC's
df_pca = pd.DataFrame(principalComponents, columns=['PC1', 'PC2'])

# creating on dataframe, with the dates as an index
principalComponents = pd.concat([df_y_train, df_pca], axis=1)
principalComponents.set_index('date', inplace=True)
```

In [47]:

```python
principalComponents
```

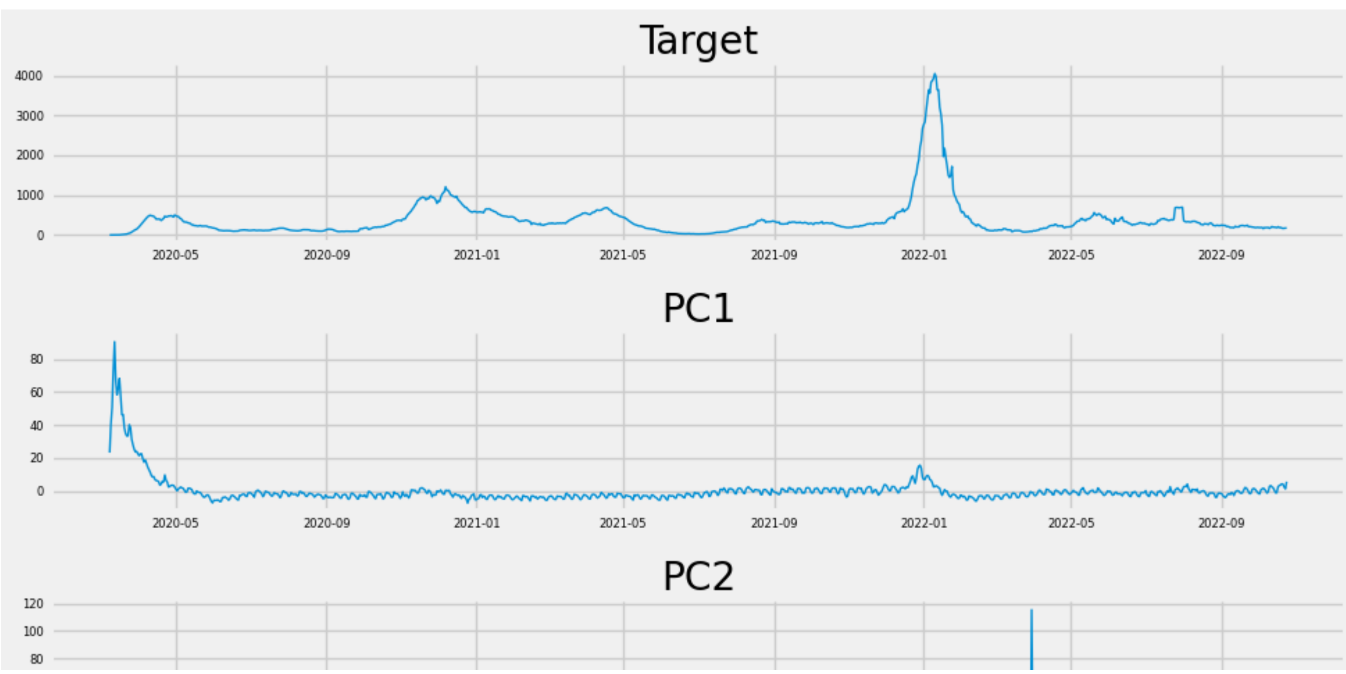
Out[47]:

date	Target	PC1	PC2
2020-03-08	0.1	23.163343	-0.492309
2020-03-09	0.1	40.021992	0.519253
2020-03-10	0.4	50.712437	0.548543
2020-03-11	0.9	72.832200	0.588393
2020-03-12	1.1	90.530683	0.211749
...	...	...	...
2022-10-20	173.9	4.224713	0.566310
2022-10-21	165.0	4.043852	-0.005439
2022-10-22	169.3	2.355044	-0.628784
2022-10-23	176.4	1.226569	-0.540444
2022-10-24	176.9	5.609688	0.662038

961 rows × 3 columns

In [48]:

```python
# modeling the PC's and target data
fig, axes = plt.subplots(nrows=3, figsize=(10, 6))
for i, ax in enumerate(axes.flatten()):
    data = principalComponents[principalComponents.columns[i]]
    ax.plot(data, linewidth=1)
    ax.set_title(principalComponents.columns[i])
    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.spines["top"].set_alpha(0)
    ax.tick_params(labelsize=6)

plt.tight_layout()
```



In [49]:
```
# explained variance for each PC
print(pca.explained_variance_ratio_)
```

```
[0.60385586 0.12999278]
```

These two components explain ~73% of the explained variance. Let's see if these variables are stationary by looking at their Dicky-Fuller scores.

In [50]:
```
for col in principalComponents:
    print(col)
    dicky_fuller(principalComponents[col])
```

```
Target
ADF Statistic: -4.014737
p-value: 0.001335
Critical Values:
	1%: -3.437
	5%: -2.865
	10%: -2.568
PC1
ADF Statistic: -5.959656
p-value: 0.000000
Critical Values:
	1%: -3.437
	5%: -2.865
	10%: -2.568
PC2
ADF Statistic: -18.063000
p-value: 0.000000
Critical Values:
	1%: -3.437
	5%: -2.865
	10%: -2.568
```

Let's further explore these variables using a Granger causality test.

5.1.1 Granger Causality Test

What is a Granger causality test? Granger causality is a statistical test that helps determine whether one time series can be used to predict another time series. The test is based on the idea that if time series X "Granger-causes" another time series Y, then information about the past values of X should help predict the future values of Y better than just using information about the past values of Y alone. In other words, if changes in X can be used to predict changes in Y better than simply looking at past values of Y, then X is said to Granger-cause Y.

Here we are going to look at the Granger-cause between our target variable and `PC1`. When looking at these Granger-causes, it's most important to look at the p-values.

In [51]:

```python
print('7-day Average New Cases causes PC1\n')
print('------------------')
granger_1 = grangercausalitytests(principalComponents[['PC1', 'Target']], 10)

print('\nPC1 causes 7-day Average New Cases\n')
print('------------------')
granger_2 = grangercausalitytests(principalComponents[['Target', 'PC1']], 10)
```

```
7-day Average New Cases causes PC1

------------------

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.5953  , p=0.2069  , df_denom=957, df_num=1
ssr based chi2 test:   chi2=1.6003  , p=0.2059  , df=1
likelihood ratio test: chi2=1.5989  , p=0.2061  , df=1
parameter F test:         F=1.5953  , p=0.2069  , df_denom=957, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.2393  , p=0.7872  , df_denom=954, df_num=2
ssr based chi2 test:   chi2=0.4811  , p=0.7862  , df=2
likelihood ratio test: chi2=0.4810  , p=0.7862  , df=2
parameter F test:         F=0.2393  , p=0.7872  , df_denom=954, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.5418  , p=0.6538  , df_denom=951, df_num=3
ssr based chi2 test:   chi2=1.6374  , p=0.6509  , df=3
likelihood ratio test: chi2=1.6360  , p=0.6513  , df=3
parameter F test:         F=0.5418  , p=0.6538  , df_denom=951, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.6090  , p=0.6563  , df_denom=948, df_num=4
ssr based chi2 test:   chi2=2.4590  , p=0.6520  , df=4
likelihood ratio test: chi2=2.4559  , p=0.6526  , df=4
parameter F test:         F=0.6090  , p=0.6563  , df_denom=948, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.8708  , p=0.0969  , df_denom=945, df_num=5
ssr based chi2 test:   chi2=9.4629  , p=0.0920  , df=5
likelihood ratio test: chi2=9.4164  , p=0.0936  , df=5
parameter F test:         F=1.8708  , p=0.0969  , df_denom=945, df_num=5

Granger Causality
number of lags (no zero) 6
ssr based F test:         F=1.2394  , p=0.2835  , df_denom=942, df_num=6
ssr based chi2 test:   chi2=7.5388  , p=0.2739  , df=6
likelihood ratio test: chi2=7.5092  , p=0.2763  , df=6
parameter F test:         F=1.2394  , p=0.2835  , df_denom=942, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=1.5935  , p=0.1335  , df_denom=939, df_num=7
ssr based chi2 test:   chi2=11.3329 , p=0.1247  , df=7
likelihood ratio test: chi2=11.2661 , p=0.1274  , df=7
parameter F test:         F=1.5935  , p=0.1335  , df_denom=939, df_num=7
```

```
Granger Causality
number of lags (no zero) 8
ssr based F test:         F=2.2840  , p=0.0201  , df_denom=936, df_num=8
ssr based chi2 test:   chi2=18.6038 , p=0.0171  , df=8
likelihood ratio test: chi2=18.4245 , p=0.0183  , df=8
parameter F test:         F=2.2840  , p=0.0201  , df_denom=936, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=3.9801  , p=0.0001  , df_denom=933, df_num=9
ssr based chi2 test:   chi2=36.5499 , p=0.0000  , df=9
likelihood ratio test: chi2=35.8658 , p=0.0000  , df=9
parameter F test:         F=3.9801  , p=0.0001  , df_denom=933, df_num=9

Granger Causality
number of lags (no zero) 10
ssr based F test:         F=2.8594  , p=0.0016  , df_denom=930, df_num=10
ssr based chi2 test:   chi2=29.2399 , p=0.0011  , df=10
likelihood ratio test: chi2=28.7994 , p=0.0013  , df=10
parameter F test:         F=2.8594  , p=0.0016  , df_denom=930, df_num=10

PC1 causes 7-day Average New Cases

------------------

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=11.5187 , p=0.0007  , df_denom=957, df_num=1
ssr based chi2 test:   chi2=11.5548 , p=0.0007  , df=1
likelihood ratio test: chi2=11.4858 , p=0.0007  , df=1
parameter F test:         F=11.5187 , p=0.0007  , df_denom=957, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=3.4343  , p=0.0326  , df_denom=954, df_num=2
ssr based chi2 test:   chi2=6.9046  , p=0.0317  , df=2
likelihood ratio test: chi2=6.8799  , p=0.0321  , df=2
parameter F test:         F=3.4343  , p=0.0326  , df_denom=954, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.3553  , p=0.2551  , df_denom=951, df_num=3
ssr based chi2 test:   chi2=4.0958  , p=0.2513  , df=3
likelihood ratio test: chi2=4.0870  , p=0.2522  , df=3
parameter F test:         F=1.3553  , p=0.2551  , df_denom=951, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.9998  , p=0.4067  , df_denom=948, df_num=4
ssr based chi2 test:   chi2=4.0372  , p=0.4010  , df=4
likelihood ratio test: chi2=4.0287  , p=0.4021  , df=4
parameter F test:         F=0.9998  , p=0.4067  , df_denom=948, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.1108  , p=0.3529  , df_denom=945, df_num=5
ssr based chi2 test:   chi2=5.6184  , p=0.3451  , df=5
likelihood ratio test: chi2=5.6020  , p=0.3469  , df=5
parameter F test:         F=1.1108  , p=0.3529  , df_denom=945, df_num=5

Granger Causality
number of lags (no zero) 6
```

```
ssr based F test:         F=1.8044  , p=0.0952  , df_denom=942, df_num=6
ssr based chi2 test:   chi2=10.9759 , p=0.0891  , df=6
likelihood ratio test: chi2=10.9133 , p=0.0911  , df=6
parameter F test:         F=1.8044  , p=0.0952  , df_denom=942, df_num=6

Granger Causality
number of lags (no zero) 7
ssr based F test:         F=2.2231  , p=0.0304  , df_denom=939, df_num=7
ssr based chi2 test:   chi2=15.8106 , p=0.0269  , df=7
likelihood ratio test: chi2=15.6811 , p=0.0282  , df=7
parameter F test:         F=2.2231  , p=0.0304  , df_denom=939, df_num=7

Granger Causality
number of lags (no zero) 8
ssr based F test:         F=2.4105  , p=0.0141  , df_denom=936, df_num=8
ssr based chi2 test:   chi2=19.6340 , p=0.0118  , df=8
likelihood ratio test: chi2=19.4345 , p=0.0127  , df=8
parameter F test:         F=2.4105  , p=0.0141  , df_denom=936, df_num=8

Granger Causality
number of lags (no zero) 9
ssr based F test:         F=3.2553  , p=0.0007  , df_denom=933, df_num=9
ssr based chi2 test:   chi2=29.8943 , p=0.0005  , df=9
likelihood ratio test: chi2=29.4346 , p=0.0005  , df=9
parameter F test:         F=3.2553  , p=0.0007  , df_denom=933, df_num=9

Granger Causality
number of lags (no zero) 10
ssr based F test:         F=3.8007  , p=0.0000  , df_denom=930, df_num=10
ssr based chi2 test:   chi2=38.8650 , p=0.0000  , df=10
likelihood ratio test: chi2=38.0918 , p=0.0000  , df=10
parameter F test:         F=3.8007  , p=0.0000  , df_denom=930, df_num=10
```

Here we see that while there are only a few viable lags when examining if our target variable Granger-causes `PC1` (lags 8, 9, and 10), there are more lags that express that `PC1` Granger-causes our target variable (lags 8, 9, and 10 as well as 7, 2, and 1).

In short, we may be able to use the lags of the other variable to better model our predictions then if we modeled the variables separately. Because we have more lags that express that `PC1` Granger-causes our target variable, which seems to indicate a higher likelihood that the causality is more weighed in that direction.

I chose not to run a Granger-causality test on `PC2`, as most of the explained variance (due to the nature of PCA) is in `PC1`. As such, it is highly likely we'll see non-significant Granger-causality between `PC2` and our target variable.

That being said we will still include `PC2` in our final model as it's interaction terms could aid in making our model more predictive. The hope is that by including `PC1` and `PC2` in our new model, we will create a better but not overly complex model.

5.2 VAR model

Vector Autoregressive (VAR) models are one type statistical models used to model the behavior of multiple variables over time. In VAR models, each variable is modeled as a linear function of its own past values, as well as the past values of all the other variables in the system. The model assumes that the variables are jointly dependent and that each variable can be explained by the others in the system. VAR models can be used for forecasting and causal analysis. The order of a VAR model specifies the number of lags of each variable that are included in the model. We are going to use a VAR model to see if it can better predict future

COVID-19 case rates than our baseline model.

In [52]:
```
# creating a dataframe with the variables we want in our VAR model
var_df = principalComponents[['Target', 'PC1', 'PC2']]
var_df
```

Out[52]:

date	Target	PC1	PC2
2020-03-08	0.1	23.163343	-0.492309
2020-03-09	0.1	40.021992	0.519253
2020-03-10	0.4	50.712437	0.548543
2020-03-11	0.9	72.832200	0.588393
2020-03-12	1.1	90.530683	0.211749
...	...	...	...
2022-10-20	173.9	4.224713	0.566310
2022-10-21	165.0	4.043852	-0.005439
2022-10-22	169.3	2.355044	-0.628784
2022-10-23	176.4	1.226569	-0.540444
2022-10-24	176.9	5.609688	0.662038

961 rows × 3 columns

In [53]:
```
# instantiating the model
var_model = VAR(var_df)
```

```
/Users/rachelsanderlin/opt/anaconda3/envs/covid-env/lib/python3.9/site-packages/
statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information wa
s provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

In [54]:

```python
# code inspiration taken from https://github.com/nachi-hebbar/Multivariate-Ti

# finding the best order for our VAR model
sorted_order = var_model.select_order(maxlags=20)
print(sorted_order.summary())
```

```
 VAR Order Selection (* highlights the minimums)
==================================================
       AIC         BIC         FPE         HQIC
--------------------------------------------------
0        17.76       17.77   5.149e+07       17.76
1        11.05       11.11   6.281e+04       11.07
2        10.96       11.06   5.734e+04       11.00
3        10.81       10.96   4.934e+04       10.87
4        10.69       10.89   4.393e+04       10.77
5        10.60       10.85   4.017e+04       10.70
6        10.42       10.72   3.361e+04       10.53
7        10.41       10.75   3.317e+04       10.54
8        10.14      10.52*   2.523e+04      10.28*
9        10.14       10.57   2.526e+04       10.30
10       10.13       10.61   2.505e+04       10.31
11       10.14       10.67   2.542e+04       10.34
12       10.14       10.71   2.523e+04       10.35
13       10.14       10.76   2.532e+04       10.37
14       10.16       10.82   2.573e+04       10.41
15      10.03*       10.74  2.262e+04*       10.30
16       10.03       10.79   2.267e+04       10.32
17       10.03       10.84   2.278e+04       10.34
18       10.04       10.89   2.285e+04       10.36
19       10.05       10.94   2.310e+04       10.39
20       10.04       10.98   2.289e+04       10.40
--------------------------------------------------
```

In [55]:

```
# fitting our VAR model
var_model = VARMAX(var_df, order=(8, 0), enforce_stationarity=True)
fitted_model = var_model.fit(disp=False)
print(fitted_model.summary())
```

```
/Users/rachelsanderlin/opt/anaconda3/envs/covid-env/lib/python3.9/site-packages/
statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information wa
s provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/Users/rachelsanderlin/opt/anaconda3/envs/covid-env/lib/python3.9/site-packages/
statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimizati
on failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "

                              Statespace Model Results
================================================================================
====
Dep. Variable:     ['Target', 'PC1', 'PC2']   No. Observations:
961
Model:                               VAR(8)   Log Likelihood                 -951
1.623
                                + intercept   AIC                            1918
5.247
Date:                      Mon, 01 May 2023   BIC                            1957
9.553
Time:                              08:24:32   HQIC                           1933
5.399
Sample:                          03-08-2020
                               - 10-24-2022
Covariance Type:                        opg
================================================================================
================
Ljung-Box (L1) (Q):        0.04, 0.54, 0.12   Jarque-Bera (JB):   58707.06, 9060
4.06, 32631491.09
Prob(Q):                   0.85, 0.46, 0.73   Prob(JB):
0.00, 0.00, 0.00
Heteroskedasticity (H): 10.08, 0.20, 41.29   Skew:                            -
0.47, 0.95, 29.59
Prob(H) (two-sided):       0.00, 0.00, 0.00   Kurtosis:                        41.2
8, 50.53, 903.80
                           Results for equation Target
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      4.8784      4.264      1.144      0.253      -3.480      13.236
L1.Target      1.0248      0.016     64.982      0.000       0.994       1.056
L1.PC1         0.2995      3.435      0.087      0.931      -6.432       7.031
L1.PC2         0.1035     10.371      0.010      0.992     -20.223      20.430
L2.Target      0.2089      0.026      7.914      0.000       0.157       0.261
L2.PC1         1.2810      4.076      0.314      0.753      -6.709       9.271
L2.PC2         0.0048     10.126      0.000      1.000     -19.841      19.851
L3.Target      0.0129      0.038      0.343      0.732      -0.061       0.087
L3.PC1        -1.8626      3.901     -0.478      0.633      -9.508       5.782
L3.PC2        -0.1428      9.461     -0.015      0.988     -18.685      18.400
L4.Target     -0.0655      0.031     -2.135      0.033      -0.126      -0.005
L4.PC1         2.1082      2.694      0.783      0.434      -3.172       7.388
L4.PC2         0.0466      9.217      0.005      0.996     -18.019      18.112
L5.Target      0.0382      0.035      1.093      0.275      -0.030       0.107
L5.PC1        -2.4766      3.606     -0.687      0.492      -9.544       4.591
L5.PC2        -0.1646     12.962     -0.013      0.990     -25.569      25.240
L6.Target     -0.1575      0.029     -5.469      0.000      -0.214      -0.101
```

L6.PC1	1.8176	3.374	0.539	0.590	−4.796	8.431
L6.PC2	−0.0372	12.343	−0.003	0.998	−24.228	24.154
L7.Target	−0.4397	0.029	−15.307	0.000	−0.496	−0.383
L7.PC1	−1.2617	3.416	−0.369	0.712	−7.956	5.433
L7.PC2	−0.0419	9.270	−0.005	0.996	−18.211	18.128
L8.Target	0.3651	0.017	21.408	0.000	0.332	0.399
L8.PC1	0.3728	2.142	0.174	0.862	−3.824	4.570
L8.PC2	−0.2474	6.457	−0.038	0.969	−12.902	12.407

Results for equation PC1

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0430	0.156	0.276	0.783	−0.263	0.349
L1.Target	−0.0014	0.003	−0.480	0.631	−0.007	0.004
L1.PC1	1.2950	0.072	18.100	0.000	1.155	1.435
L1.PC2	0.0011	0.315	0.003	0.997	−0.617	0.619
L2.Target	0.0015	0.004	0.390	0.697	−0.006	0.009
L2.PC1	−0.5568	0.072	−7.736	0.000	−0.698	−0.416
L2.PC2	−0.0195	0.298	−0.066	0.948	−0.603	0.564
L3.Target	−0.0003	0.003	−0.086	0.931	−0.007	0.006
L3.PC1	0.2526	0.081	3.124	0.002	0.094	0.411
L3.PC2	0.0033	0.301	0.011	0.991	−0.587	0.594
L4.Target	0.0027	0.003	0.878	0.380	−0.003	0.009
L4.PC1	−0.0122	0.097	−0.126	0.900	−0.202	0.178
L4.PC2	0.0082	0.219	0.037	0.970	−0.421	0.437
L5.Target	−0.0016	0.004	−0.420	0.674	−0.009	0.006
L5.PC1	−0.0585	0.053	−1.109	0.267	−0.162	0.045
L5.PC2	0.0291	0.135	0.216	0.829	−0.235	0.293
L6.Target	−0.0004	0.004	−0.100	0.920	−0.009	0.008
L6.PC1	0.0620	0.056	1.103	0.270	−0.048	0.172
L6.PC2	0.0544	0.154	0.354	0.723	−0.247	0.355
L7.Target	0.0002	0.003	0.049	0.961	−0.006	0.006
L7.PC1	0.4031	0.076	5.324	0.000	0.255	0.552
L7.PC2	0.0223	0.153	0.146	0.884	−0.277	0.322
L8.Target	−0.0007	0.002	−0.302	0.763	−0.005	0.004
L8.PC1	−0.4395	0.041	−10.743	0.000	−0.520	−0.359
L8.PC2	−0.0131	0.145	−0.090	0.928	−0.298	0.272

Results for equation PC2

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.1492	1.731	0.086	0.931	−3.244	3.542
L1.Target	0.0007	0.030	0.022	0.982	−0.058	0.059
L1.PC1	0.2284	0.785	0.291	0.771	−1.311	1.767
L1.PC2	0.2306	0.353	0.652	0.514	−0.462	0.923
L2.Target	−0.0006	0.048	−0.012	0.991	−0.094	0.093
L2.PC1	−0.1944	0.921	−0.211	0.833	−2.000	1.611
L2.PC2	0.0590	0.488	0.121	0.904	−0.897	1.015
L3.Target	0.0002	0.038	0.005	0.996	−0.074	0.075
L3.PC1	−0.0517	0.807	−0.064	0.949	−1.633	1.529
L3.PC2	0.0259	0.275	0.094	0.925	−0.514	0.566
L4.Target	0.0002	0.045	0.003	0.997	−0.087	0.088
L4.PC1	−0.0062	1.026	−0.006	0.995	−2.018	2.005
L4.PC2	−0.0015	1.236	−0.001	0.999	−2.425	2.422
L5.Target	−0.0005	0.044	−0.011	0.991	−0.086	0.085
L5.PC1	−0.0920	1.159	−0.079	0.937	−2.365	2.181
L5.PC2	0.0097	2.974	0.003	0.997	−5.819	5.838
L6.Target	−0.0006	0.056	−0.010	0.992	−0.111	0.110
L6.PC1	0.1051	1.224	0.086	0.932	−2.293	2.503
L6.PC2	0.0183	4.428	0.004	0.997	−8.660	8.696
L7.Target	0.0001	0.057	0.002	0.998	−0.111	0.111
L7.PC1	0.0788	0.861	0.092	0.927	−1.608	1.766

```
L7.PC2          0.0161       3.387       0.005       0.996      -6.623       6.655
L8.Target       0.0003       0.039       0.006       0.995      -0.076       0.077
L8.PC1         -0.0883       0.510      -0.173       0.862      -1.087       0.911
L8.PC2          0.0156       1.257       0.012       0.990      -2.448       2.479
                                Error covariance matrix
================================================================================
=======
                          coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-------
sqrt.var.Target        44.3787      1.020     43.498      0.000      42.379
46.378
sqrt.cov.Target.PC1     0.0457      0.134      0.340      0.734      -0.218
0.309
sqrt.var.PC1            1.6783      0.048     34.884      0.000       1.584
1.773
sqrt.cov.Target.PC2    -0.0613      3.022     -0.020      0.984      -5.984
5.861
sqrt.cov.PC1.PC2        0.2179      2.232      0.098      0.922      -4.156
4.592
sqrt.var.PC2            3.7849      0.084     45.268      0.000       3.621
3.949
================================================================================
=======

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

In [56]:

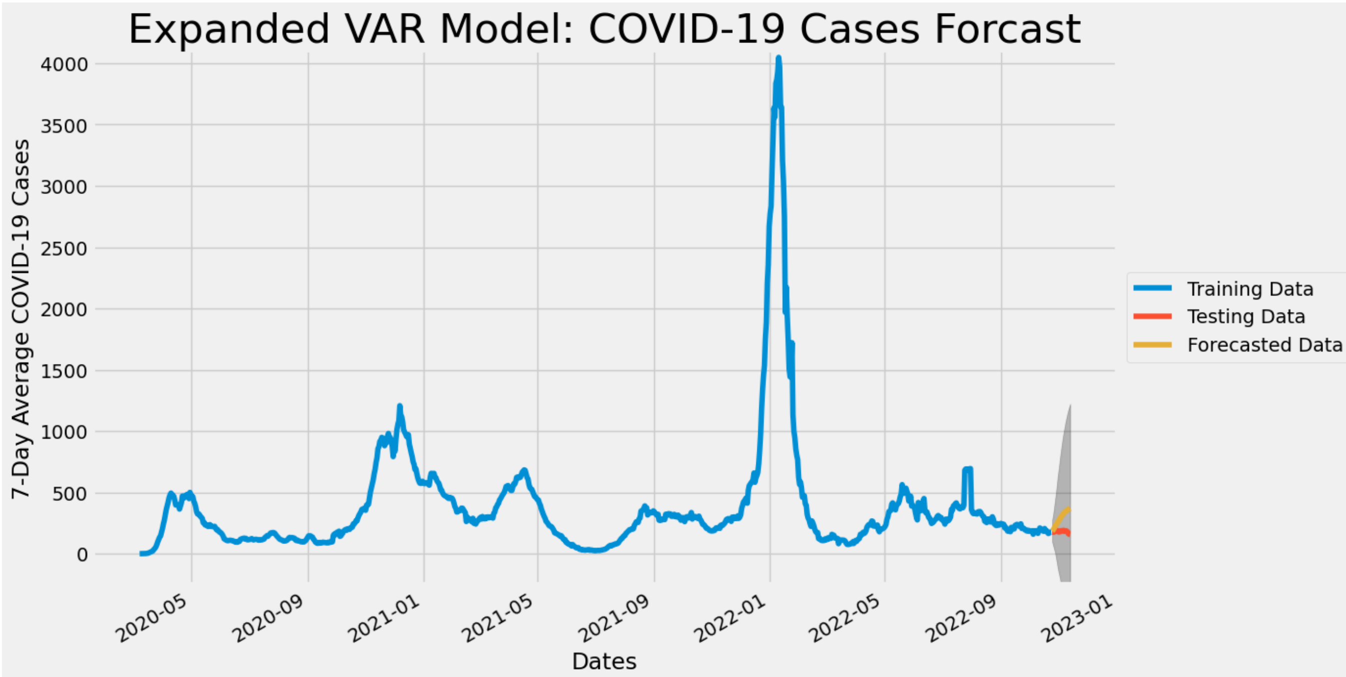
```python
# getting predictions based on the VAR model
predict = fitted_model.get_prediction(
    start=first_date_test, end=last_date_test)
predictions = predict.predicted_mean

# get confidence interval
var_conf_all = fitted_model.get_forecast(steps=last_date_test).conf_int()
var_conf = var_conf_all[["lower Target", "upper Target"]]
```

In [57]:

```python
# make DataFrame for predictions
predictions.columns = ['Target predicted', 'PC1 predicted', 'PC2 predicted',
```

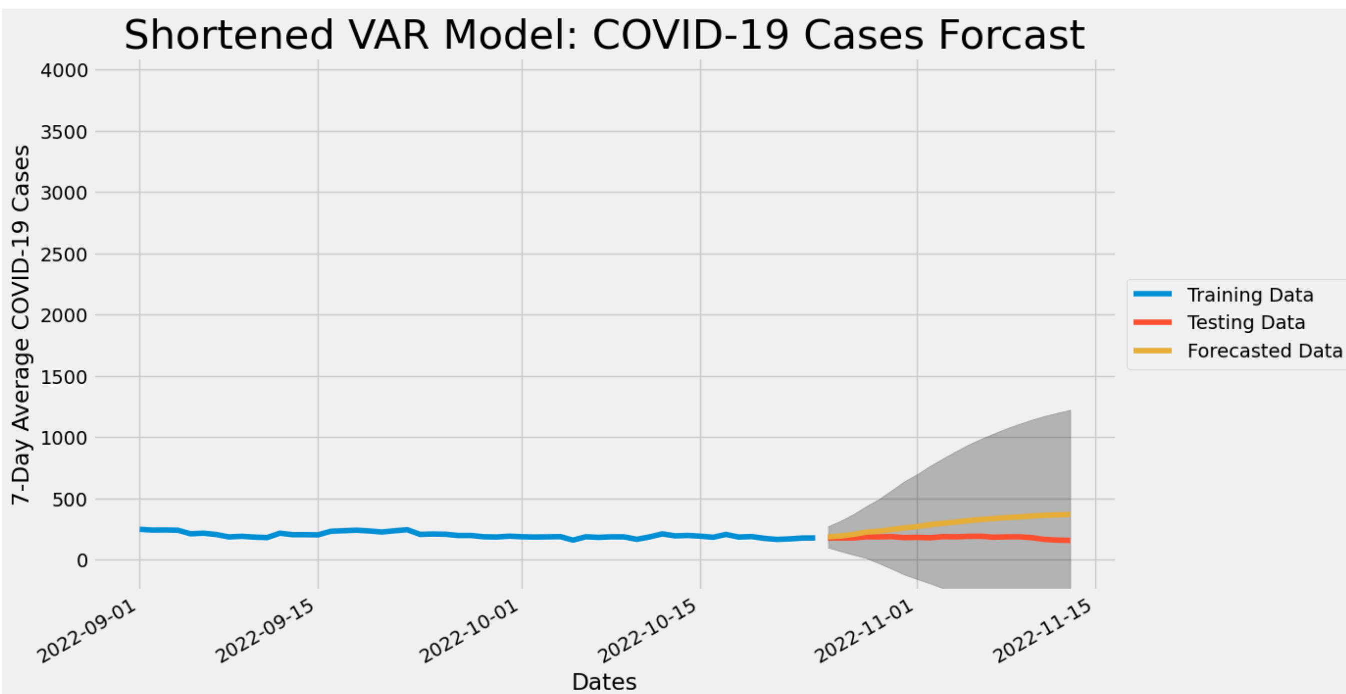
In [58]:

```
plot_model(var_df["Target"], y_test, predictions["Target predicted"], "Expand
            '7-Day Average COVID-19 Cases', conf_interval=var_conf)
```



In [59]:

```
plot_model(var_df["Target"], y_test, predictions["Target predicted"], "Shorte
            '7-Day Average COVID-19 Cases', conf_interval=var_conf, cutoff_dat
```



6 Final Model Evaluation

In [60]:
```
# Looking at the loss functions to evaluate how our data preformed
print('MAE:', np.mean(abs(predictions["Target predicted"] - y_test.values)))
print('RMSE:', np.sqrt(
    np.mean((predictions["Target predicted"] - y_test.values)**2)))
print('MAPE:', np.mean(
    abs((predictions["Target predicted"] - y_test.values)/y_test.values)))
```

```
MAE: 111.87319697010344
RMSE: 128.19036422590986
MAPE: 0.6321903172448982
```

All of the above loss functions for our VAR model performed worse compared to the original (base) model. Just a reminder, the original models loss functions were:

In [61]:
```
print('MAE:', np.mean(abs(forcast_list - y_test.values)))
print('RMSE:', np.sqrt(np.mean((forcast_list - y_test.values)**2)))
print('MAPE:', np.mean(abs((forcast_list - y_test.values)/y_test.values)))
```

```
MAE: 24.85583845359837
RMSE: 29.799693896902426
MAPE: 0.13903013173746206
```

As such, I would say that our VAR model (using the VAR model and PCA the way we did) *did not* have better prediction capabilities than simply modeling the daily COVID-19 cases.

7 Conclusion

7.1 Summary of Analysis

The analysis began by cleaning and processing the Google COVID-19 search data and the public Pennsylvania COVID-19 data. Both datasets were then subset so as to only include Philadelphia county. These datasets were then joined together. After joining the datasets and creating some initial visualizations of the case counts data, a train-test split was performed. We then used `pmdarima.arima.auto_arima` (https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html) to run a grid search. This grid search allowed us to find the optimal orders to model the chosen target variable (`7-Day Average COVID-19 Cases`) using `statsmodels.tsa.statespace.sarimax.SARIMAX` (https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html#statsmodels.tsa.statespace.sarimax.SARIMAX). This SARIMAX model was our baseline model. We then performed a scree plot to find the optimal number of components to run in our PCA (Principal Component Analysis). Based on the scree plot we chose to reduce our dimensions to two components. After assessing the principal components, we used them along with our target variable in our VAR (Vector Auto Regression) model. The hope was that by combining the target data along with these two principal components, we would create a better (but not overly complex) model. The VAR model we implemented used `statsmodels.tsa.statespace.varmax.VARMAX` (https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.varmax.VARMAX.html). We used loss functions to evaluate and compare our two models.

	Baseline Model	VAR Model
MAE	24.86	111.87

RMSE	29.80	128.19
MAPE	0.14	0.63

As the baseline model outperformed the VAR model, we *cannot* say that using the Google search trends is helpful in predicting COVID-19 cases, at least with the model created in this notebook.

7.2 Recommendations

For now, the best way to predict COVID-19 cases in Philadelphia is by looking at Philadelphia's previous COVID-19 cases.

7.3 Next Steps

There are other types of models that may better utilize the COVID-19 Google search data. Trying out these alternate methods were not possible in the time frame allowed for this project, but may give different results.

Some possible directions to explore:

- Modeling VARMA or VARMAX models.
- Using crossvalidation and or recursive modeling methods (documentation [here (https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_forecasting.html#Cross-validation)](https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_forecasting.html#Cross-validation)).
- Include other possibly relevant data (e.g. when novel COVID-19 outbreaks happened, public opinion about COVID-19, vaccination rates) which could improve the predictive ability of the model.