Table of Contents

1 Top 5 Zip Codes to Invest In

1.1 Summary

Real estate data from Zillow Research (https://www.zillow.com/research/data/), which offers comprehensive data on zip codes in the Philadelphia metro area (consisting of 281 Philadelphia zip codes) was used for this analysis. The dataset originally included 14,723 zip codes, so preprocessing started by removing the extraneous (non-Philadelphia) zip codes and irrelevant variables. The top ten Philadelphia zip codes were then chosen by sorting the zip codes by highest ROI (https://www.investopedia.com/terms/r/returnoninvestment.asp) after the 2008 crash. Afterwards, the data was melted to format it for time series modeling. The top ten zip codes were then split into training and validation sets, and modeled in order to predict how the real estate values would change over 3 years. Our final model used `statsmodels.tsa.statespace.sarimax.SARIMAX` (https://www.statsmodels.org/stable/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html) to create SARIMA models to forecast real estate values. Prior to this we used the same package to extract the optimal elements needed to make each dataset stationary. The capability to extract these

elements was why we chose this specific package. Using the predicted data and the validation data, we looked at the RMSE (https://www.statology.org/how-to-interpret-rmse/) for each model, and chose models with the lowest RMSE scores as our best modeled zip codes. Finally, we predicted the ROI for real estate purchased in our final five zip codes. Our final recommended zip codes to invest in, with their expected ROI's in January 2020 are: `19148` (22%), `19123` (19%), `19130` (17%), `19145` (14%), `19428` (13%).

1.2 Business and Data Understanding

1.2.1 Business Problem

A real estate investment firm centered in Philadelphia reached out to a consultant to help them figure out **"What are the top 5 best zip codes for us to invest in?"**. The goal of the analysis in this notebook is to answer the stakeholders question by recommending the top 5 zip codes to invest in the Philly Metro area *and* explain the logic behind those recommendations.

1.3 Dataset

The dataset was originally extracted from Zillow Research (https://www.zillow.com/research/data/). This specific dataset can be found here (https://github.com/learn-co-curriculum/dsc-phase-4-choosing-a-dataset/tree/main/time-series). The dataset includes **14,723** rows (representing zip codes) and **272** columns, and includes the median sales information for each zip code from April 1996 through April 2018.

1.3.1 Limitations of the Dataset

The goal of this analysis is to find the best zip codes to invest in, but this particular dataset doesn't include enough ancillary information about the datasets to fully grasp (with out further information) why some zip codes are more expensive or have changing housing prices. Additionally, this dataset provides the median monthly housing prices in a zip code. Using the median monthly price also removed some of the granularity of the dataset, which may have enabled us to create more accurate models.

1.3.2 Why We Used This Dataset

Zillow is a website that advertises properties for rent or sale all over the USA. Due to it's easy accessibility and expansive data collection, including the Philadelphia metro area, we chose to use this dataset.

1.3.3 Dataset Size

In the raw data, we had **14,723** rows and **272** columns, but after selecting the data relevant to the Philadelphia metro area we had **281** rows and **272** original columns. The dataset covers from April 1996 through April 2018, and includes the median sales information for every month between those years.

2 Initial Look at the Data

2.0.1 Imports

In [1]:
```
# imports
from math import sqrt
from sklearn.metrics import mean_squared_error
import warnings
from pylab import hist, show, xticks
import itertools
from statsmodels.tsa.stattools import adfuller
from matplotlib.pylab import rcParams
import statsmodels.api as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')
warnings.filterwarnings('ignore')
```

In [2]:
```
import session_info
session_info.show()
```

Out[2]: Click to view session information

In [3]:
```
# initial look at the raw data and importing the second column (with t
data = pd.read_csv("zillow_data.csv", dtype={'RegionName': object})
data.head()
```

Out[3]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	33540
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	23690
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	21220
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	50090
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	7730

5 rows × 272 columns

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Columns: 272 entries, RegionID to 2018-04
dtypes: float64(219), int64(48), object(5)
memory usage: 30.6+ MB
```

2.1 Data Preprocessing

We are going to get rid of the columns `RegionID` and `SizeRank` as they seem irrelevant to our current line of questioning. (We don't care about the size or region of the zip codes we recommend.) We are then going to reformat our data so we can look for and deal with any NaN's we may come across.

In [5]:
```
data.drop(['RegionID', 'SizeRank'], axis=1, inplace=True)
```

In [6]:
```
data.head()
```

Out[6]:

	RegionName	City	State	Metro	CountyName	1996-04	1996-05	1996-06	1996-0
0	60657	Chicago	IL	Chicago	Cook	334200.0	335400.0	336500.0	337600
1	75070	McKinney	TX	Dallas-Fort Worth	Collin	235700.0	236900.0	236700.0	235400
2	77494	Katy	TX	Houston	Harris	210400.0	212200.0	212200.0	210700
3	60614	Chicago	IL	Chicago	Cook	498100.0	500900.0	503100.0	504600
4	79936	El Paso	TX	El Paso	El Paso	77300.0	77300.0	77300.0	77300

5 rows × 270 columns

In [7]:
```
# pulling out Philly data specifically
phil_data = data.loc[data["Metro"] == "Philadelphia"]
phil_data
```

Out[7]:

	RegionName	City	State	Metro	CountyName	1996-04	1996-05	1996-0
194	19111	Philadelphia	PA	Philadelphia	Philadelphia	84600.0	84500.0	84400.
282	19124	Philadelphia	PA	Philadelphia	Philadelphia	43100.0	43000.0	42900.
329	19446	Lansdale	PA	Philadelphia	Montgomery	155300.0	154900.0	154600.
343	19020	Bensalem	PA	Philadelphia	Bucks	122000.0	121800.0	121600.
351	19720	New Castle	DE	Philadelphia	New Castle	91300.0	92000.0	92600.
...	...	...	...	...	...	...	...	...
13729	08026	Gibbsboro	NJ	Philadelphia	Camden	100500.0	100800.0	101100.
14189	08067	Oldmans	NJ	Philadelphia	Salem	81300.0	81600.0	81900.
14239	19453	Upper Providence	PA	Philadelphia	Montgomery	110100.0	109900.0	109700.
14496	18915	Hatfield	PA	Philadelphia	Montgomery	165300.0	165300.0	165400.

In [8]:
```
# checking for null values
phil_data[pd.isna(phil_data).any(axis=1)]
```

Out[8]:

RegionName	City	State	Metro	CountyName	1996-04	1996-05	1996-06	1996-07	1996-08

0 rows × 270 columns

2.2 Exploratory Data Analysis

In order to evaluate our data, we are going to use return on investment (ROI) (https://www.investopedia.com/terms/r/returnoninvestment.asp). Because our data includes the 2008 housing market crash, two columns showing the ROI both before (`growth1`) and after the crash (`growth2`) were created. The specific dates chosen to delineate these periods were taken from here (https://www.thebalancemoney.com/stock-market-crash-of-2008-3305535).

In [9]:
```
# create metrics to judge return on investment (ROI)

# total ROI
phil_data['t_growth'] = (phil_data['2018-04'] -
                         phil_data['1996-04']) / phil_data['1996-04']
# ROI pre-2008 crash
phil_data['growth1'] = (phil_data['2007-09'] -
                        phil_data['1996-04']) / phil_data['1996-04']
# ROI after 2008 crash
phil_data['growth2'] = (phil_data['2018-04'] -
                        phil_data['2011-03']) / phil_data['2011-03']

# re-check the info in a bit more manageable slice
phil_shortform = phil_data.loc[:, ['RegionName', 'City', 'State',
                                   'Metro', 'CountyName', 't_growth',
phil_shortform.head()
```

Out[9]:

	RegionName	City	State	Metro	CountyName	t_growth	growth1	growth2
194	19111	Philadelphia	PA	Philadelphia	Philadelphia	1.078014	1.141844	0.132003
282	19124	Philadelphia	PA	Philadelphia	Philadelphia	1.046404	1.125290	0.119289
329	19446	Lansdale	PA	Philadelphia	Montgomery	1.034127	1.020605	0.144151
343	19020	Bensalem	PA	Philadelphia	Bucks	1.077049	1.204918	0.109457
351	19720	New Castle	DE	Philadelphia	New Castle	0.960570	1.111720	0.198928

In [10]:
```
# checking for null values
phil_data[pd.isna(phil_data).any(axis=1)]
```

Out[10]:

RegionName	City	State	Metro	CountyName	1996-04	1996-05	1996-06	1996-07	1996-08

0 rows × 273 columns

In [11]:
```
# re-check the info in a bit more manageable slice
phil_shortform.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 281 entries, 194 to 14537
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   RegionName  281 non-null    object 
 1   City        281 non-null    object 
 2   State       281 non-null    object 
 3   Metro       281 non-null    object 
 4   CountyName  281 non-null    object 
 5   t_growth    281 non-null    float64
 6   growth1     281 non-null    float64
 7   growth2     281 non-null    float64
dtypes: float64(3), object(5)
memory usage: 19.8+ KB
```

In [12]:
```
phil_shortform.describe()
```

Out[12]:

	t_growth	growth1	growth2
count	281.000000	281.000000	281.000000
mean	1.086182	1.181123	0.114567
std	0.755744	0.430239	0.132367
min	0.237132	0.713235	-0.147526
25%	0.774878	1.004141	0.040498
50%	0.960227	1.108796	0.114919
75%	1.148588	1.209726	0.165386
max	7.046012	4.311475	1.059172

Here we are looking at the top 10 philly zipcodes with the highest ROI after the 2008 crash.

In [13]:
```
# find the top 10 zipcodes in Philadelphia who grew the most after the
phil_shortform.sort_values('growth2', ascending=False).head(10)
```

Out[13]:

	RegionName	City	State	Metro	CountyName	t_growth	growth1	growt
666	19145	Philadelphia	PA	Philadelphia	Philadelphia	4.155556	1.975309	1.0591
3615	19125	Philadelphia	PA	Philadelphia	Philadelphia	7.046012	3.288344	0.956(
518	19148	Philadelphia	PA	Philadelphia	Philadelphia	4.111922	2.481752	0.586{
940	19103	Philadelphia	PA	Philadelphia	Philadelphia	3.784584	2.756508	0.542:
784	19147	Philadelphia	PA	Philadelphia	Philadelphia	5.941176	4.105169	0.541[
5214	19123	Philadelphia	PA	Philadelphia	Philadelphia	5.709836	4.311475	0.508(
531	19131	Philadelphia	PA	Philadelphia	Philadelphia	1.686649	1.130790	0.478:
1807	19130	Philadelphia	PA	Philadelphia	Philadelphia	4.596606	3.567885	0.356(
5510	19106	Philadelphia	PA	Philadelphia	Philadelphia	2.665775	2.334560	0.321(
4747	19428	Conshohocken	PA	Philadelphia	Montgomery	1.773519	1.267422	0.310:

A note on the top 10 Philly zip codes:

It's interesting to note that all but one of the top 10 zip codes are in the city of Philadelphia proper (the exception being Conshohocken `19428`). Unsuprisingly, this list also includes `19106` - the historical district of Philadelphia, `19103` - Rittenhouse Square where wealthy New York businessmen have bought apartments, `19145`, `19147`, `19148` - are all zip codes in South Philadelphia, an area that is known to be gentrifying, and `19125` (Fishtown) which has turned into a hip spot for artists, bars, and restaurants.

In [14]:
```
# find the top 10 zipcodes who had the worst recovery after the crash
phil_shortform.sort_values('growth2', ascending=False).tail(10)
```

Out[14]:

	RegionName	City	State	Metro	CountyName	t_growth	growth1	growth
4316	19018	Upper Darby	PA	Philadelphia	Delaware	0.474866	0.819251	-0.09216
2766	19026	Drexel Hill	PA	Philadelphia	Delaware	0.390034	0.820447	-0.09355
12721	08045	Lawnside	NJ	Philadelphia	Camden	0.537572	1.086705	-0.09523
10482	08059	Mount Ephraim	NJ	Philadelphia	Camden	0.623843	1.108796	-0.09890
1725	19082	Upper Darby	PA	Philadelphia	Delaware	0.371429	0.947619	-0.10187
678	08021	Lindenwold	NJ	Philadelphia	Camden	0.471698	1.100236	-0.10280
12781	08063	National Park	NJ	Philadelphia	Gloucester	0.476942	1.134709	-0.10711
9024	19079	Sharon Hill	PA	Philadelphia	Delaware	0.425076	0.738532	-0.13623
8037	08079	Salem	NJ	Philadelphia	Salem	0.436765	1.039706	-0.13996
7613	08030	Gloucester City	NJ	Philadelphia	Camden	0.402458	1.227343	-0.14752

Perhaps unsuprisingly, all of the worst performing zipcodes are located outside of the city of Philadelphia.

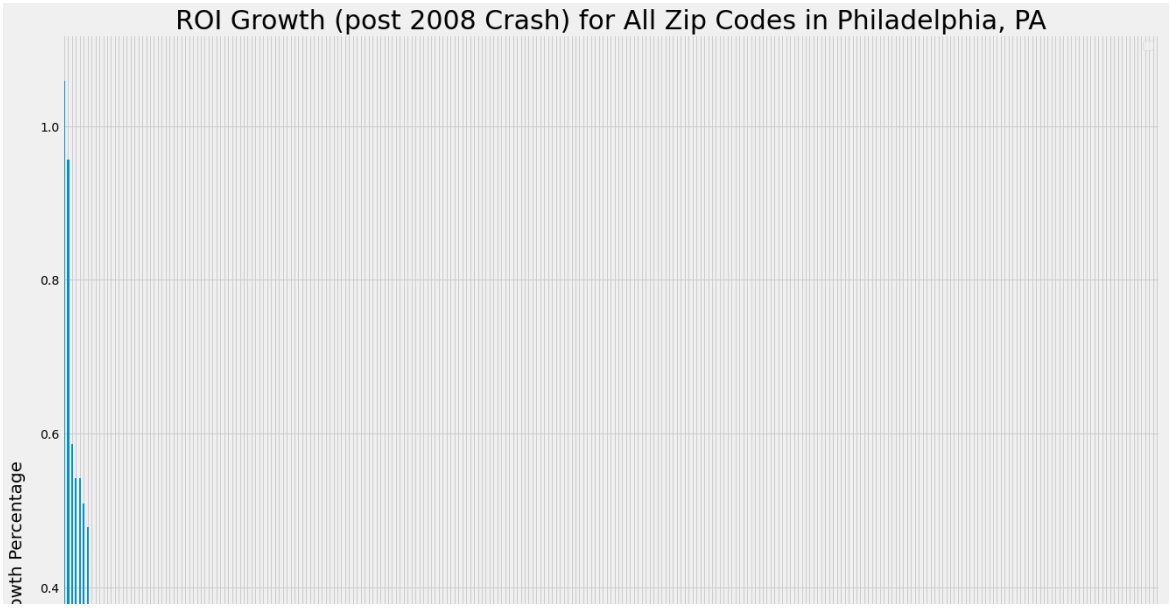
3 Visualization to Finalize the List of Zip Codes

To start off, let's take a look at the ROI (https://www.investopedia.com/terms/r/returnoninvestment.asp) growth of all of the zip codes in the Philadelphia region.

In [15]:
```
phil_by_growth2 = phil_shortform[['RegionName', 'growth2']].sort_value
    by='growth2', ascending=False)
```

In [16]:
```
phil_by_growth2.plot.bar(x='RegionName', y='growth2', figsize=(20, 20)
plt.title(
    'ROI Growth (post 2008 Crash) for All Zip Codes in Philadelphia, P
plt.legend('')
plt.xlabel('Zip codes', fontsize=20)
plt.ylabel('Growth Percentage', fontsize=20)
```

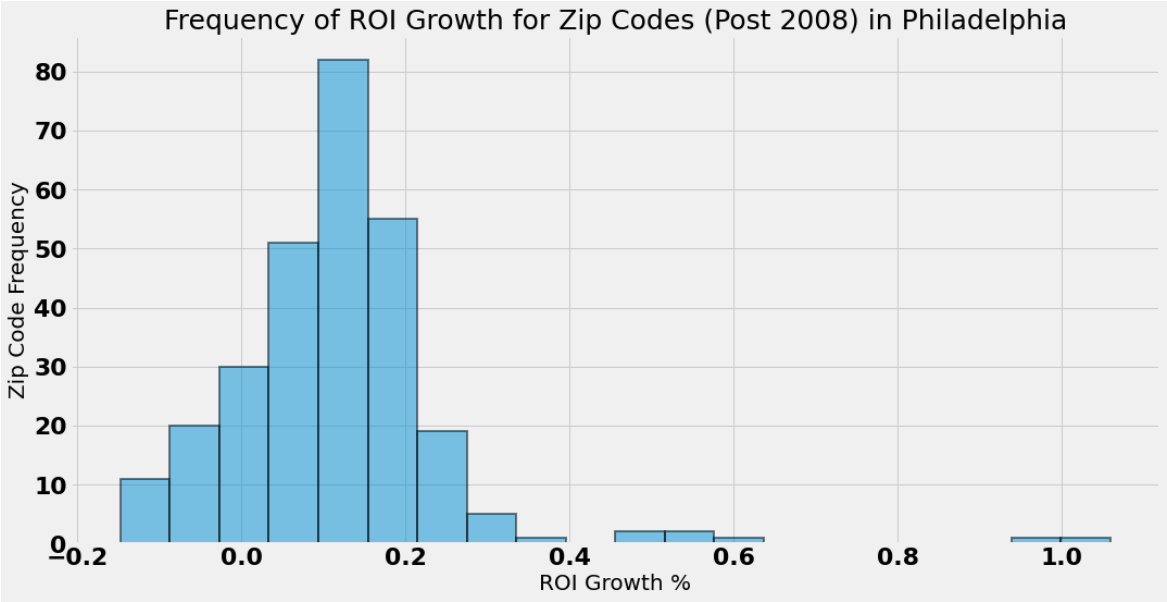
Out[16]:
```
Text(0, 0.5, 'Growth Percentage')
```



While the above graph gives us a good general sense of ROI in the Philadelphia metro area after the 2008 crash, it's near impossible to see any specific zip code - lets make a histogram to get a better sense of the frequency of the data, and a new bar graph with the top 10 zip codes.

In [17]:
```
# formatting for the rest of our visualizations
font = {'family': 'DejaVu Sans',
        'weight': 'bold',
        'size': 22}

plt.rc('font', **font)
```

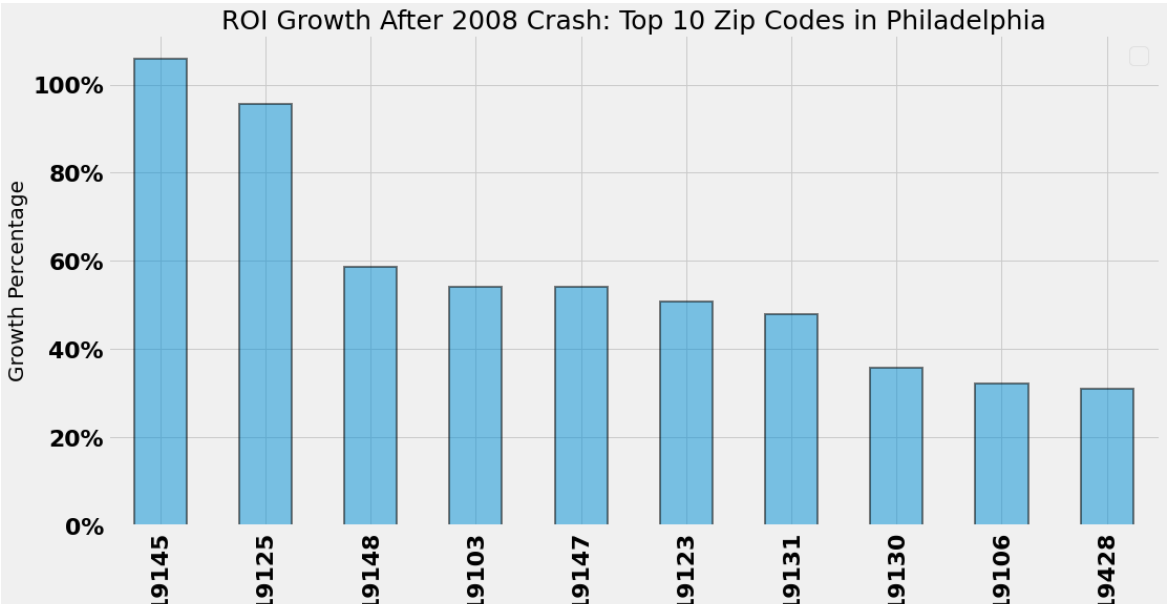
In [18]:
```
# making the histogram
phil_by_growth2.hist(bins=20, figsize=(16, 8), alpha=0.5,
                     edgecolor="black", linewidth=2)
plt.title(
    'Frequency of ROI Growth for Zip Codes (Post 2008) in Philadelphia
plt.xlabel('ROI Growth %', fontsize=20)
plt.ylabel('Zip Code Frequency', fontsize=20);
```



In the above histogram we see that most of the growth in the area was around 10-15%, with some zip codes having a median *decrease* in their ROI. One zip code seems to have more than doubled it's ROI. In the bar graph below, we'll be able to see which zip code that is!

In [19]:
```
ax = phil_by_growth2.head(10).plot.bar(x='RegionName', y='growth2', fi
    16, 8), alpha=0.5, edgecolor="black", linewidth=2)
plt.title(
    'ROI Growth After 2008 Crash: Top 10 Zip Codes in Philadelphia', f
plt.legend('')
ax.set_yticklabels(["0%", "20%", "40%", "60%", "80%", "100%"])
plt.xlabel('Zip Codes', fontsize=20)
plt.ylabel('Growth Percentage', fontsize=20)
```

Out[19]:
```
Text(0, 0.5, 'Growth Percentage')
```



Here we see that zip code 19145 more than doubled it's value, closely followed by 19125 . Following this, the rest of the top ten zip codes seem to have around a 60-30% value increase following the 2008 crash.

3.1 Melting the Data

Now we are going to reshape our data, so that we can model the how each zip code performs on it's own.

In [20]:
```
# looking at our origional unsorted data again
phil_data.head()
```

Out[20]:

	RegionName	City	State	Metro	CountyName	1996-04	1996-05	1996-06
194	19111	Philadelphia	PA	Philadelphia	Philadelphia	84600.0	84500.0	84400.0
282	19124	Philadelphia	PA	Philadelphia	Philadelphia	43100.0	43000.0	42900.0
329	19446	Lansdale	PA	Philadelphia	Montgomery	155300.0	154900.0	154600.0
343	19020	Bensalem	PA	Philadelphia	Bucks	122000.0	121800.0	121600.0
351	19720	New Castle	DE	Philadelphia	New Castle	91300.0	92000.0	92600.0

5 rows × 273 columns

In [21]:
```
# pulling out the date columns so we can more easily change the format
dates = phil_data.loc[:, ~phil_data.columns.isin(
    ['RegionName', 'City', 'State', 'Metro', 'CountyName', 't_growth',
```

In [22]:
```
# adjusting the column names so we can use pd.melt later
column_names_to_change = {}

for column in dates.columns:
    column_names_to_change[column] = f'{column}-01'

phil_data.rename(columns=column_names_to_change, inplace=True)

phil_data.head()
```

Out[22]:

	RegionName	City	State	Metro	CountyName	1996-04-01	1996-05-01	1996-
194	19111	Philadelphia	PA	Philadelphia	Philadelphia	84600.0	84500.0	84
282	19124	Philadelphia	PA	Philadelphia	Philadelphia	43100.0	43000.0	42
329	19446	Lansdale	PA	Philadelphia	Montgomery	155300.0	154900.0	154
343	19020	Bensalem	PA	Philadelphia	Bucks	122000.0	121800.0	121
351	19720	New Castle	DE	Philadelphia	New Castle	91300.0	92000.0	92

5 rows × 273 columns

In [23]:
```
# taking the top 10 zipcodes and their information
top_10_philly = phil_data.sort_values(by='growth2', ascending=False).h
top_10_philly
```

Out[23]:

	RegionName	City	State	Metro	CountyName	1996-04-01	1996-05-01	1
666	19145	Philadelphia	PA	Philadelphia	Philadelphia	40500.0	40500.0	
3615	19125	Philadelphia	PA	Philadelphia	Philadelphia	32600.0	32700.0	
518	19148	Philadelphia	PA	Philadelphia	Philadelphia	41100.0	41100.0	
940	19103	Philadelphia	PA	Philadelphia	Philadelphia	195900.0	196800.0	
784	19147	Philadelphia	PA	Philadelphia	Philadelphia	56100.0	55600.0	
5214	19123	Philadelphia	PA	Philadelphia	Philadelphia	61000.0	60600.0	
531	19131	Philadelphia	PA	Philadelphia	Philadelphia	36700.0	36700.0	
1807	19130	Philadelphia	PA	Philadelphia	Philadelphia	76600.0	76700.0	
5510	19106	Philadelphia	PA	Philadelphia	Philadelphia	298900.0	298900.0	
4747	19428	Conshohocken	PA	Philadelphia	Montgomery	114800.0	114800.0	

10 rows × 273 columns

In [24]:
```
def after_crash(df):
    '''reduce our data so we only will see the dates/values *after* th

    # droping all columns with infomation from before April 2011
    df_after_crash = df.drop(df.columns[5:185], axis=1, inplace=True)
    return df_after_crash


after_crash(top_10_philly)
top_10_philly
```

Out[24]:

	RegionName	City	State	Metro	CountyName	2011-04-01	2011-05-01	2(
666	19145	Philadelphia	PA	Philadelphia	Philadelphia	101400.0	100400.0	
3615	19125	Philadelphia	PA	Philadelphia	Philadelphia	132300.0	129400.0	
518	19148	Philadelphia	PA	Philadelphia	Philadelphia	130100.0	127500.0	
940	19103	Philadelphia	PA	Philadelphia	Philadelphia	605700.0	605100.0	
784	19147	Philadelphia	PA	Philadelphia	Philadelphia	251100.0	249600.0	
5214	19123	Philadelphia	PA	Philadelphia	Philadelphia	268600.0	262000.0	
531	19131	Philadelphia	PA	Philadelphia	Philadelphia	66200.0	65800.0	
1807	19130	Philadelphia	PA	Philadelphia	Philadelphia	316300.0	315000.0	
5510	19106	Philadelphia	PA	Philadelphia	Philadelphia	825800.0	826600.0	
4747	19428	Conshohocken	PA	Philadelphia	Montgomery	241700.0	240200.0	

10 rows × 93 columns

In [25]:
```
def melt_data(df, plot=False):
    '''
    Here we are going to melt the data, so we can have our time series
    This will allow us to use timeseries modeling techniques later on.
    '''

    # melting the data so we have it in the long version, instead of w
    melted = pd.melt(df, id_vars=['RegionName', 'City', 'State', 'Metr
                     'CountyName', 't_growth', 'growth1', 'growth2'],
    # makeing our new column `time` a datetime variable
    melted['time'] = pd.to_datetime(melted['time'], infer_datetime_for
    # getting rid of rows that are missing `value` values
    melted_nonan = melted.dropna(subset=['value'])
    # set `time` as index so we can model it as a time series dataset
    melted_nonan.set_index('time', inplace=True)
    # drop all the columns we no longer need
    if plot:
        melted_nonan.drop(labels=['City', 'State', 'Metro', 'CountyNam
                          't_growth', 'growth1', 'growth2'], axis=1, i
    else:
        melted_nonan.drop(labels=['RegionName', 'City', 'State', 'Metr
                          'CountyName', 't_growth', 'growth1', 'growth
    return melted_nonan
```

In [26]:
```
# handy list of just the zipcodes names
zip_list = [zip_code for zip_code in top_10_philly['RegionName']]
# zip_list
```

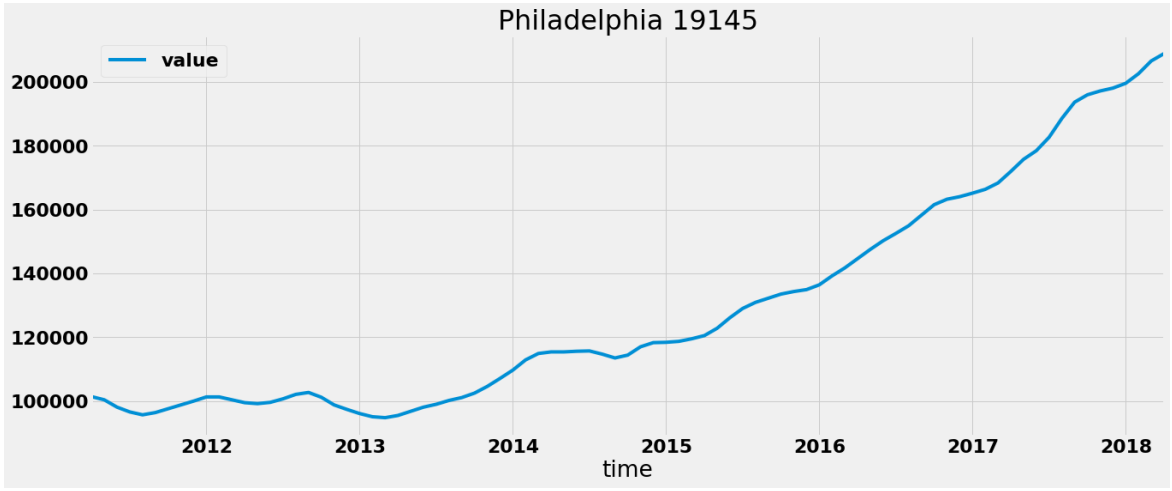
In [27]:
```
top_10_melt = []

# using a for loop to melt the data for each of our zipcodes individua
for zip_code in top_10_philly['RegionName']:
    top_10_melt.append({
        'zip_code': zip_code,
        'data': melt_data(top_10_philly.loc[(top_10_philly['RegionName
    })

# pulling out one timeseries set to make sure it worked
top_10_melt[0]['data']
```

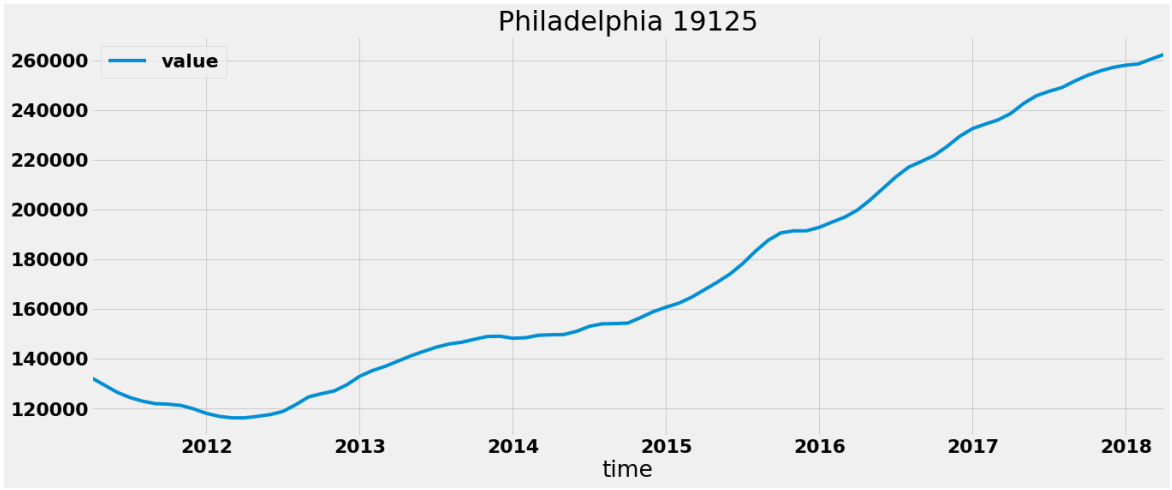
Out[27]:

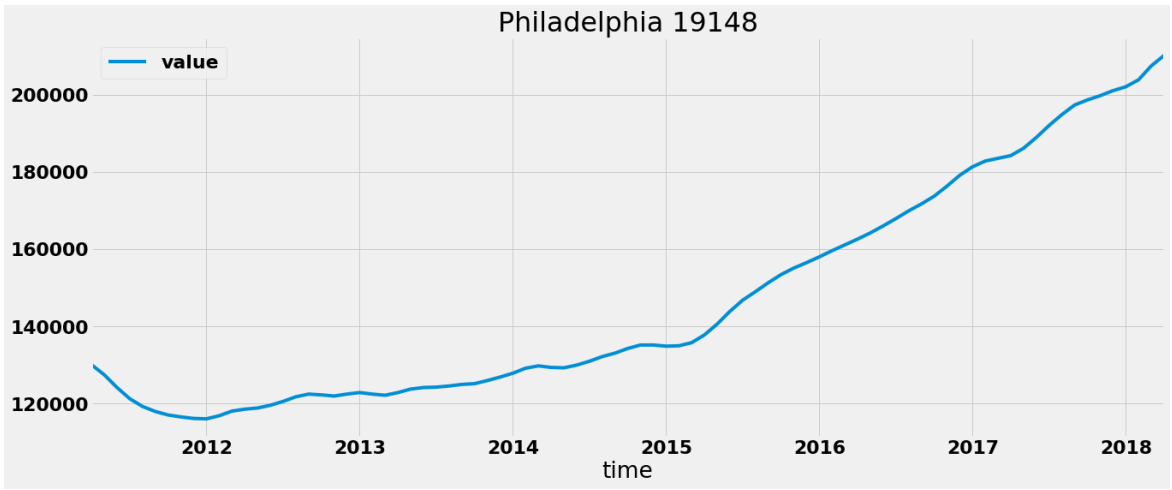
time	value
2011-04-01	101400.0
2011-05-01	100400.0
2011-06-01	98100.0
2011-07-01	96600.0
2011-08-01	95700.0
...	...
2017-12-01	198000.0
2018-01-01	199500.0
2018-02-01	202500.0
2018-03-01	206500.0

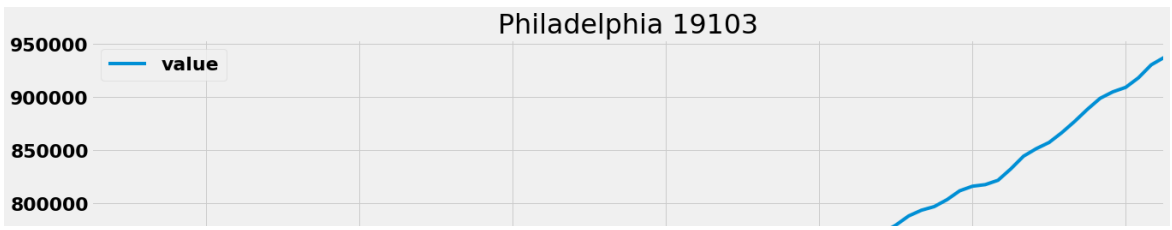
4 Initial Visualizations

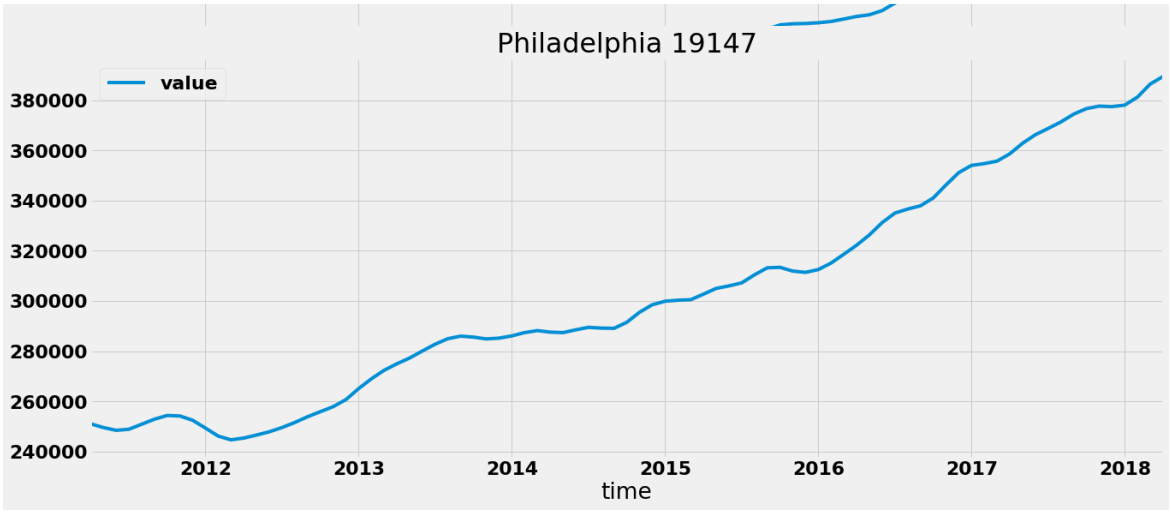
In [28]:
```
# Plotting the timeseries data for the top 10 zipcodes
for zip_code_melt in top_10_melt:
    zip_code_melt['data'].plot(
        title=f"Philadelphia {zip_code_melt['zip_code']}", figsize=(20
```

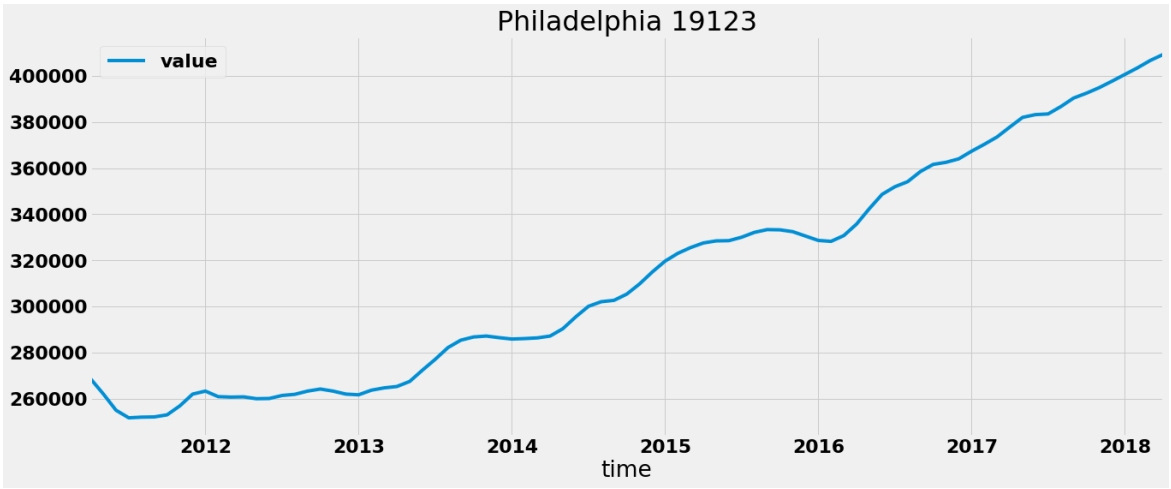


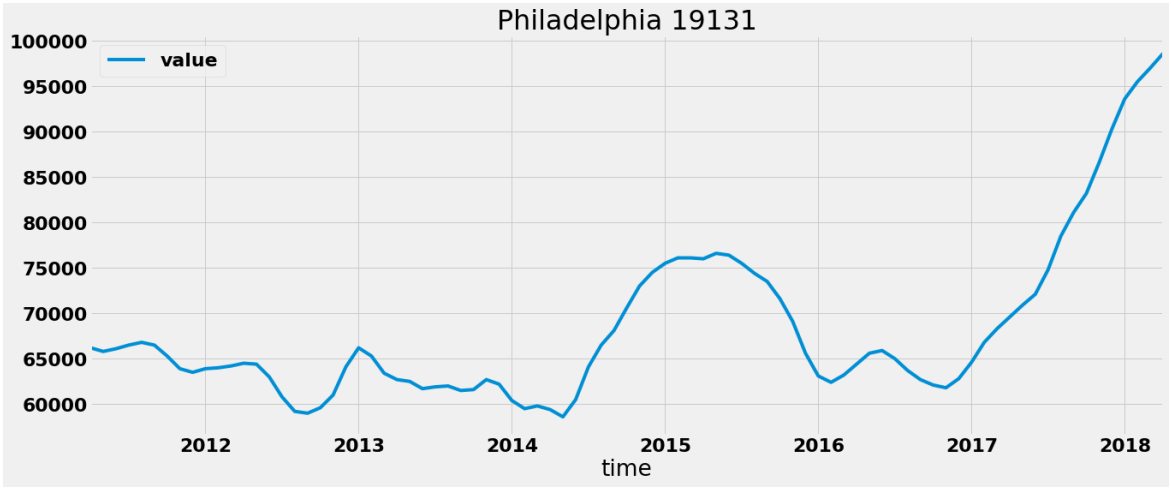


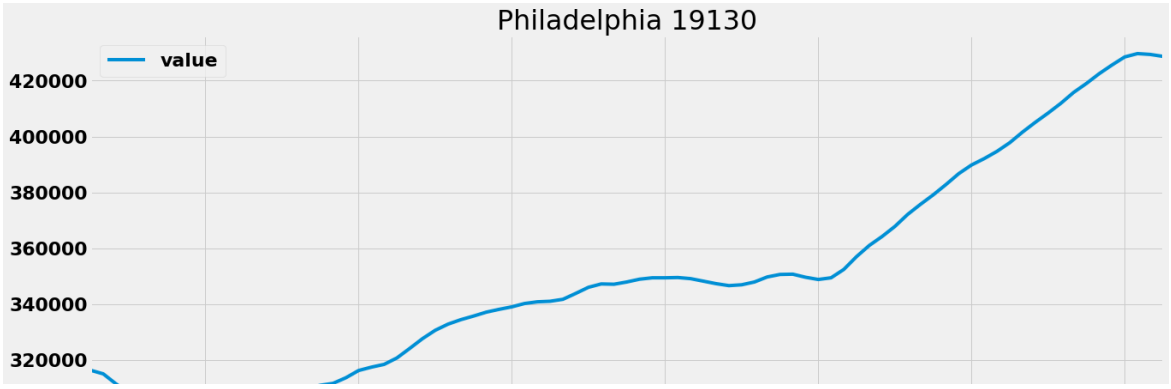


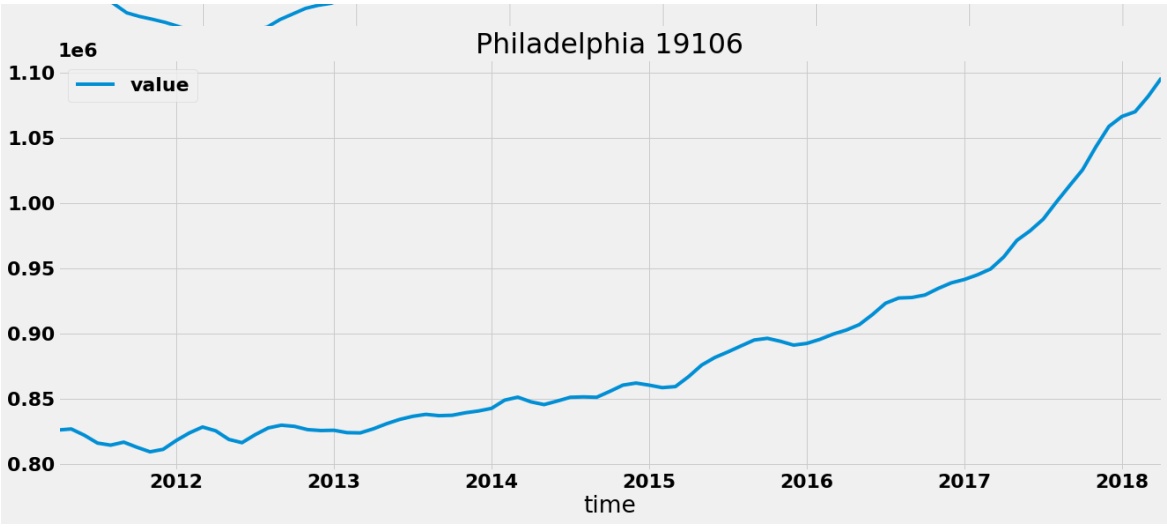


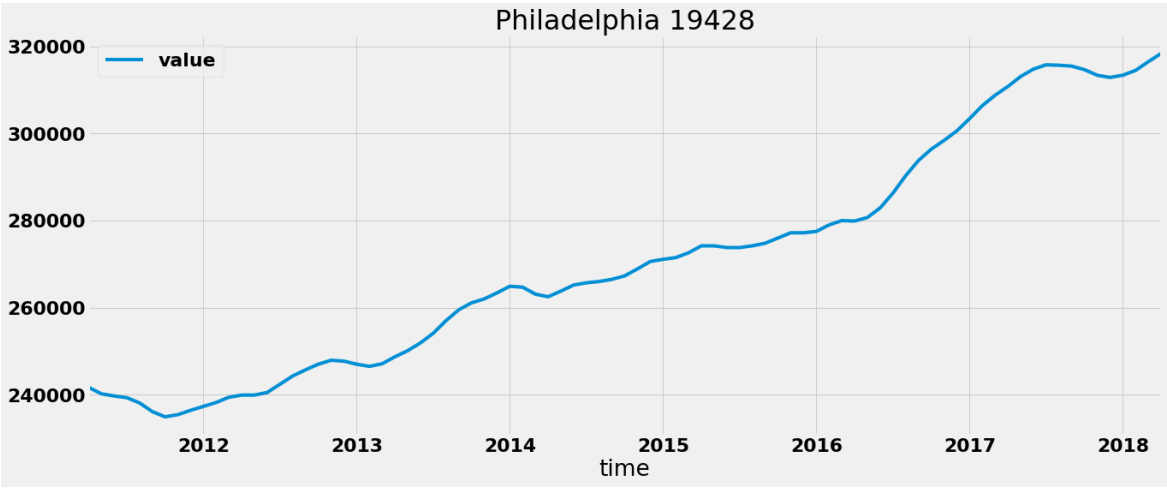

Philadelphia 19147
value
380000
360000
340000
320000
300000
280000
260000
240000
2012
2013
2014
2015
2016
2017
2018
time


Philadelphia 19123
value
400000
380000
360000
340000
320000
300000
280000
260000
2012
2013
2014
2015
2016
2017
2018
time


Philadelphia 19131
value
100000
95000
90000
85000
80000
75000
70000
65000
60000
2012
2013
2014
2015
2016
2017
2018
time


Philadelphia 19130
value
420000
400000
380000
360000
340000
320000

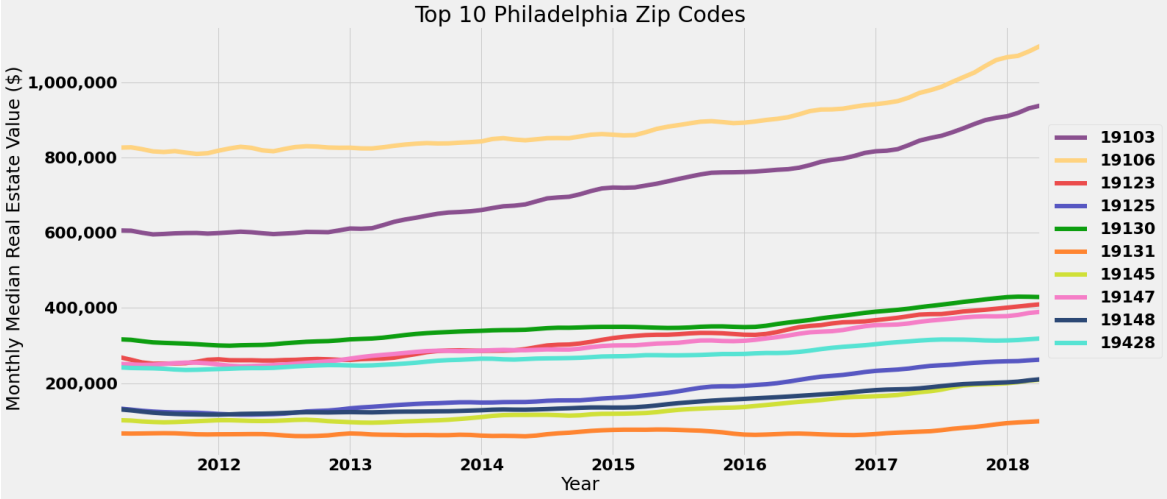




Looking at the graphs above, we can visualize more clearly how and when each zip code increased in value. That being said, it's hard to compare zip codes with them all plotted in disparate graphs. Lets plot them all in one graph now.

In [29]:
```
# creating color map so that values don't repeat
color_list = ["#d0e038", "#5757c2", "#2c4875", "#8a508f",
              "#f57ec7", "#eb4c4c", "#ff8531", "#0d9e10", "#ffd380", "
color_map = dict(list(zip(zip_list, color_list)))

# pivoting so that each zipcode becomes a column for our visualization
plot_df = melt_data(top_10_philly, plot=True).pivot(
    columns='RegionName', values='value')

# getting the zipcodes in the right order and in a list
column_zip = []
for col in plot_df.columns:
    column_zip.append(col)
```

In [30]:
```
# creating a pretty graph that plots all of our zipcodes in one place!

# creating our lineplot, setting the color map and line width size
ax = plot_df.plot.line(color=color_map, figsize=(20, 10), linewidth=6)
ax.set_yticklabels(
    [0, "200,000", "400,000", "600,000", "800,000", "1,000,000"])
ax.set_xlabel("Year", fontsize=25)
ax.set_ylabel(" Monthly Median Real Estate Value ($)", fontsize=25)
# code taken from https://stackoverflow.com/questions/4700614/how-to-p
ax.legend(column_zip, loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('Top 10 Philadelphia Zip Codes', fontsize=30)
plt.show()
```



In the graph above, we see that the `19103` zip code (Rittenhouse Square - a small but very wealthy neighborhood next to center city in Philadelphia) and the `19106` zip code (the Historic District of Philadelphia) start and end at unusually high value real estate for the Philadelphia area. The rest of the zip codes (excluding `19131`) seem to hover in the range starting from 100,000 value to 300,000 in 2011, and finish at around 200,000 to 400,000 in 2018.

5 SARIMA Modeling

Now that we have gotten a good look at our top ten zip codes, let's model them!

5.1 Train-Test Split

To begin, we'll perform a train-test split so we can later validate our predictions and evaluate our models.

Type *Markdown* and LaTeX: α^2

In [31]:
```
# creating a training and validation set of our data

trainvalid_split = []

for melted in top_10_melt:
 # find the index which allows us to split off 20% of the data
    cutoff = round(melted['data'].shape[0]*0.8)
    trainvalid_split.append({
        'train': melted['data'][:cutoff],
        'valid': melted['data'][cutoff:],
    })

# # checking that it worked!
# fig, ax = plt.subplots(figsize=(12, 8))
# ax.plot(trainvalid_split[0]['train'], label='train')
# ax.plot(trainvalid_split[0]['valid'], label='valid')
# ax.set_title(f'Train-Validation Split {top_10_melt[0]["zip_code"]}')
# plt.legend();
```

An important component of time series modeling is making the data stationary - removing any trends or seasonal patterns. One way of doing this in SARIMA (Seasonal Auro-Regression Integrated Moving Average) and similar models (ARIMA and ARMA) is by specifying elements, which tell our model how to handle the data it receives with the goal of making the data as stationary as possible for processing. The goal of the function below, `sarima_elements` is to iterate through possible combinations of these elements and find the optimal combination to stationarize the data.

Further explanation of these elements can be found in the functions descriptive text.

In [32]:
```
# code inspiration taken from https://github.com/learn-co-curriculum/d

def sarima_elements(TS):
    '''
This function will calculate the optimal elements to use in our model.
this function requires both the time series and the order of elements

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

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

The seasonal ARIMA contains these three elements along with a seasonal
    '''
    p = d = q = list(range(0, 2))
    pdq = list(itertools.product(p, d, q))
    pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p,

    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(TS,
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=F
                                                enforce_invertibility=

                output = mod.fit()
                ans.append([comb, combs, output.aic])
            except:
                continue

    ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
    out = ans_df.loc[ans_df['aic'].idxmin()]
    return out
```

Now that we have a function to find the optimal elements for each time series we want to work with, we can create another function to take those elements, run and fit our model, and print out metrics of how well the model performed.

In [33]:
```
# inspiration for this code taken from https://github.com/AnnikaNoren/


def plot_diagnostics(arima_model_output):
    # plotting the diagnostic graphs
    arima_model_output.plot_diagnostics(figsize=(15, 18))
    plt.show(block=False)


def get_arima_model_output(elements, df_train):
    # creating our model
    ARIMA_MODEL = sm.tsa.statespace.SARIMAX((df_train), order=elements
                                            enforce_stationarity=False

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

def get_prediction(arima_model_output):
    # forcast 3 years 2 months into the future (Jan 1st 2020)
    return arima_model_output.get_forecast(steps=38)

def get_prediction_confidence(prediction):
    # getting the confidence interval
    return prediction.conf_int()

def plot_model_and_prediction(prediction, prediction_confidence, df_tr
    #plot our model and it's predicted values
    fig, ax = plt.subplots(figsize=(12, 8))
    ax.plot(df_train, label='Training Data')
    ax.plot(df_valid, label='Validity Data')
    prediction.predicted_mean.plot(ax=ax, label='Forecasted Value')
    ax.fill_between(prediction_confidence.index,
                    prediction_confidence.iloc[:, 0],
                    prediction_confidence.iloc[:, 1], color='k', alpha

    plt.title(f"{zipcode} Forcast", fontsize=30)
    ax.set_xlabel('Years')
    ax.set_ylabel('Home Values')
    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

6 Model Evaluation

Lets look at each of the print outs used to evaluate the models:

- The first print out is a list of the elements used to create the model for each specific zip code, along with the AIC (Akaike Information Criterion) score. This score can be used to compare models, but can only compare ARIMA models if the elements in the models are the same. (For more information about the limitations of AIC in ARIMA models feel free to look here (https://stats.stackexchange.com/questions/4997/can-aic-compare-across-different-types-of-model) and here (https://otexts.com/fpp2/arima-estimation.html))

- The next print out is a summary table about the model. The most important column is the `coef` (coefficient) column, which lets us know how important that component of our model is. The `P>|z|` column then says how significant each component listed is to the model. Most of the p-values of our models are under 0.05, but most of the values above alpha (0.05) are in the seasonal rows - indicating that for some of our models, the seasonal component of the SARIMA model wasn't needed.

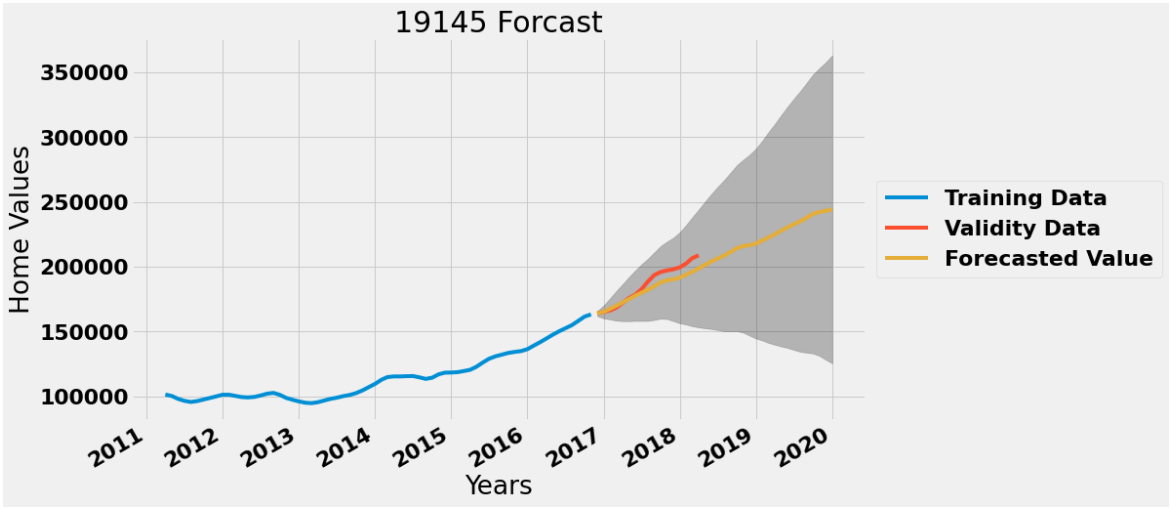
- Following the summary table we see the `plot_diagnostics` . The top left figure is a plot of the standardized residuals. If this figure looks like it's mostly noise, it indicates that the model was well standardized. The top right figure shows how well the KDE (Kernel Density Estimate - the red line) matches a normal distribution (the yellow line) overlaid on a histogram of the model, indicating normality. The bottom left q-q plot shows how well the residuals follow the linear trend, and indicate that the residuals are normally distributed. Finally, the correlogram (autocorrelation) graph on the bottom right shows that residuals have low correlation with lagged versions of itself. All in all, at a glance the `plot_diagnostics` on our models look pretty good!

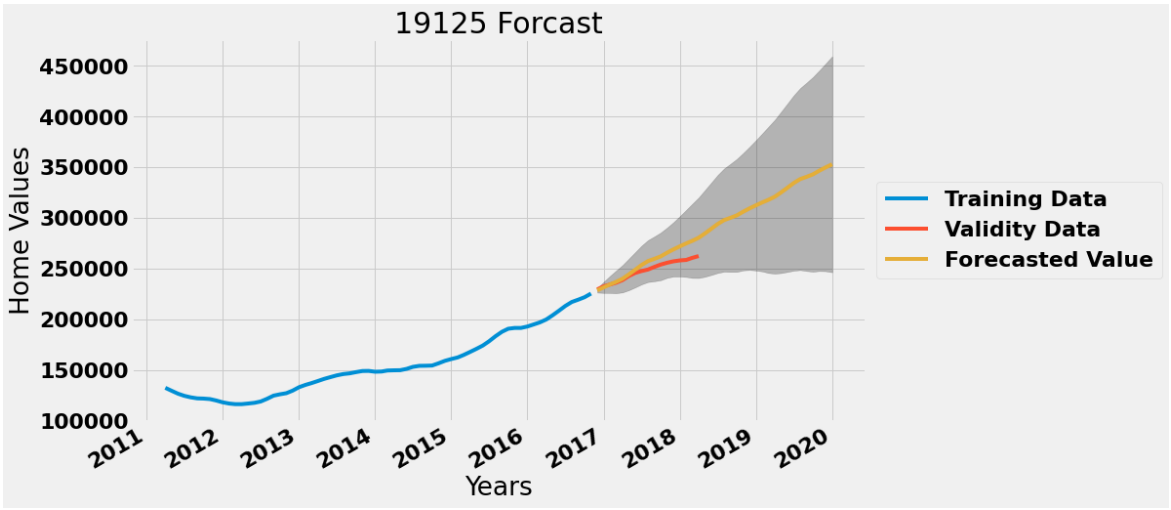
In [34]:
```
# running our model function on our 10 ten zip codes
for i in range(0, len(trainvalid_split)):
    print(zip_list[i])
    elements = sarima_elements(trainvalid_split[i]['train'])
    print(elements)
    output = get_arima_model_output(elements, trainvalid_split[i]['tra
    print((output.summary().tables[1]))
    plot_diagnostics(output)
```

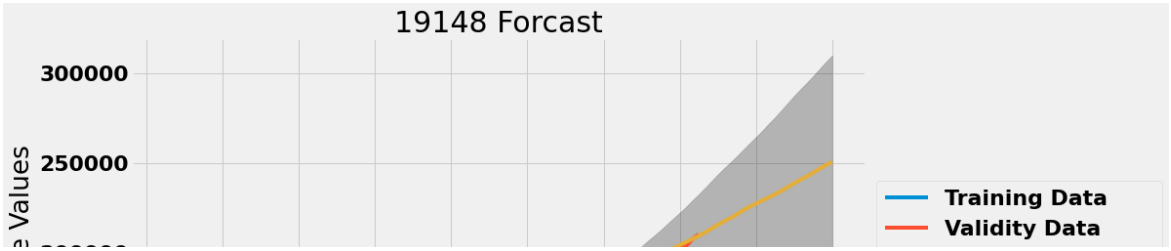
```
19145
pdq          (1, 1, 1)
pdqs     (0, 1, 1, 12)
aic            683.264
Name: 59, dtype: object
==============================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
--------
ar.L1          0.6806      0.180      3.788      0.000       0.328
1.033
ma.L1          0.6872      0.179      3.843      0.000       0.337
1.038
ma.S.L12      -0.1925      0.102     -1.884      0.060      -0.393
0.008
sigma2      1.016e+06   3.19e+05      3.189      0.001    3.91e+05
1.64e+06
```

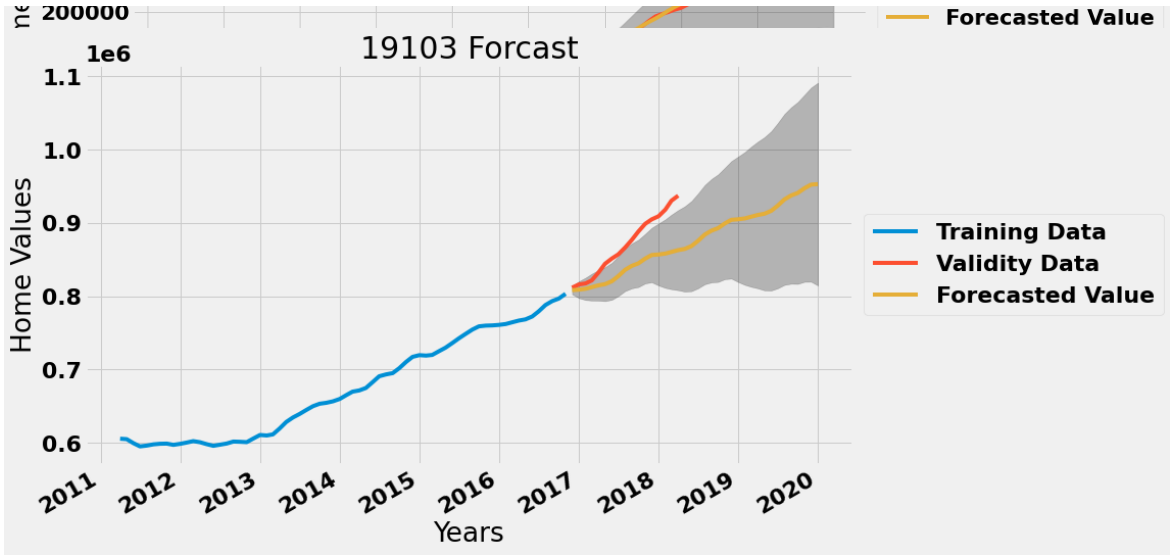
Finally, we';; see a visualization of our models: the training data, validation data, and predicted trend of each zip code. We also see a confidence interval (the gray cone) - we see that for some zip codes, the predicted values are way off, while for others it seems right on point.

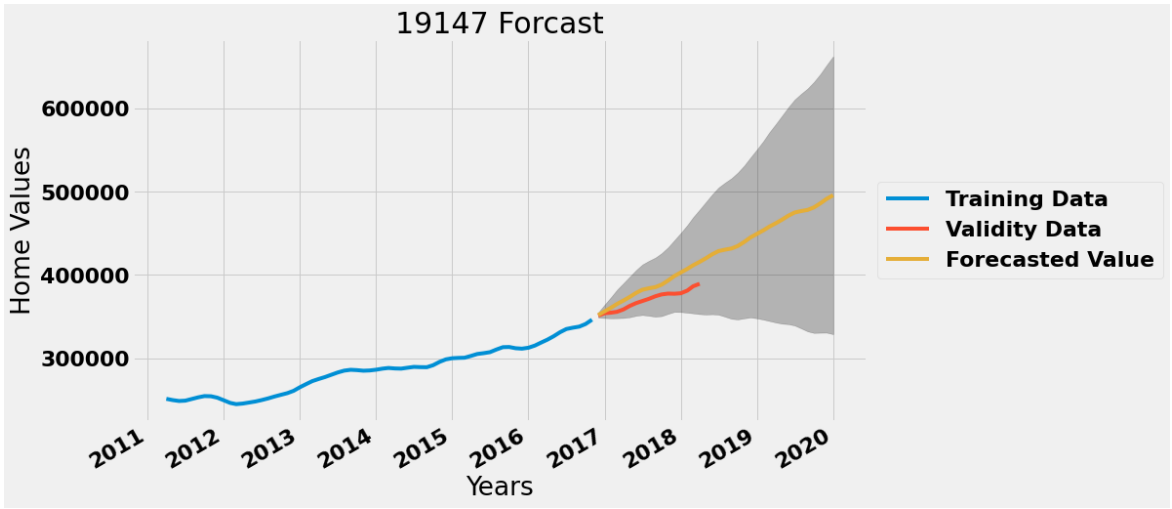
In [35]:
```
arima_models = []

for i in range(0, len(trainvalid_split)):
    elements = sarima_elements(trainvalid_split[i]['train'])
    output = get_arima_model_output(elements, trainvalid_split[i]['tra
    # forcast 3 years 2 months into the future (Jan 1st 2020)
    prediction = get_prediction(output)

    # Confidence intervals of forecasts
    prediction_confidence = get_prediction_confidence(prediction)
    plot_model_and_prediction(prediction, prediction_confidence,
                              trainvalid_split[i]['train'], trainvalid
                              zip_list[i])

    arima_models.append({'model': prediction.predicted_mean,
                         })
```

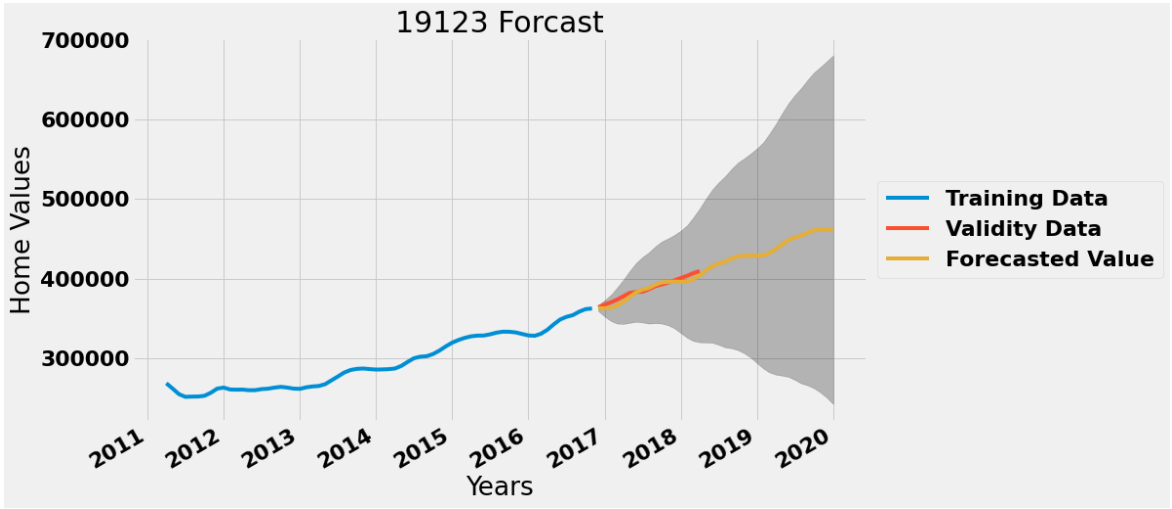


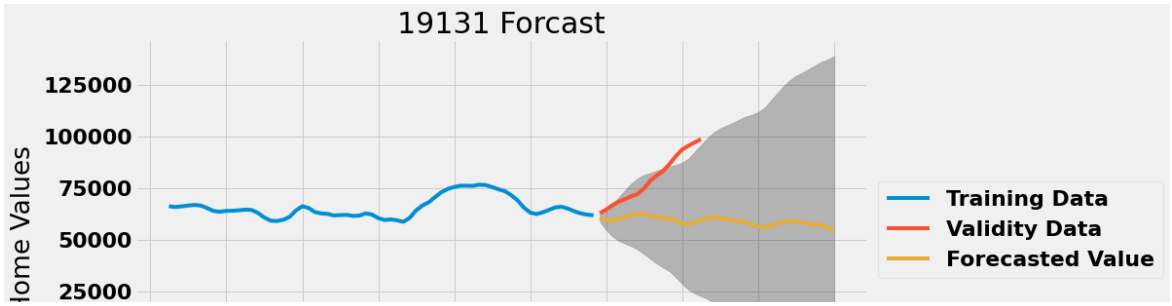


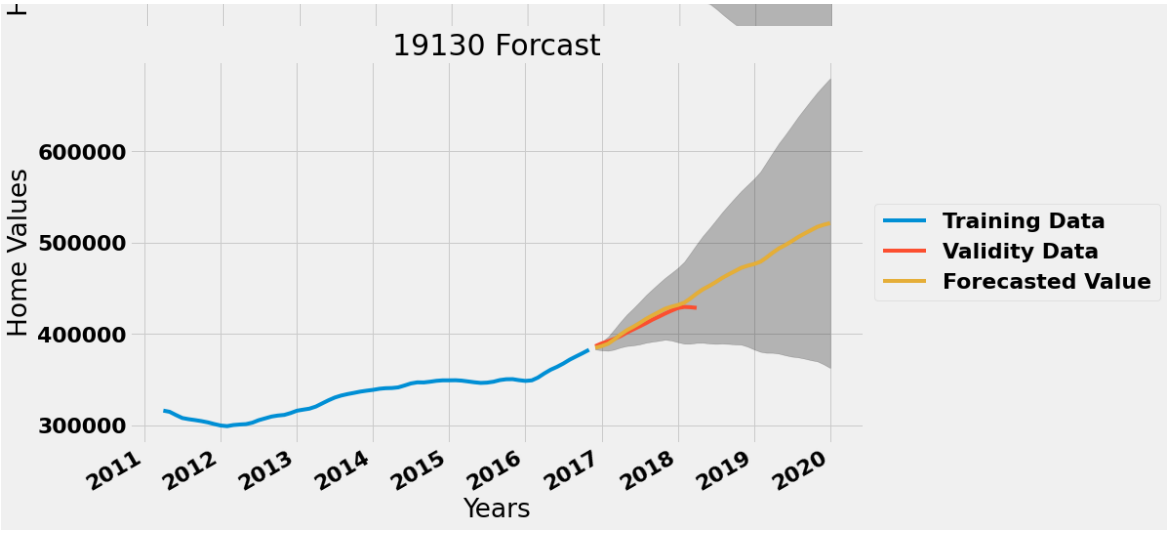


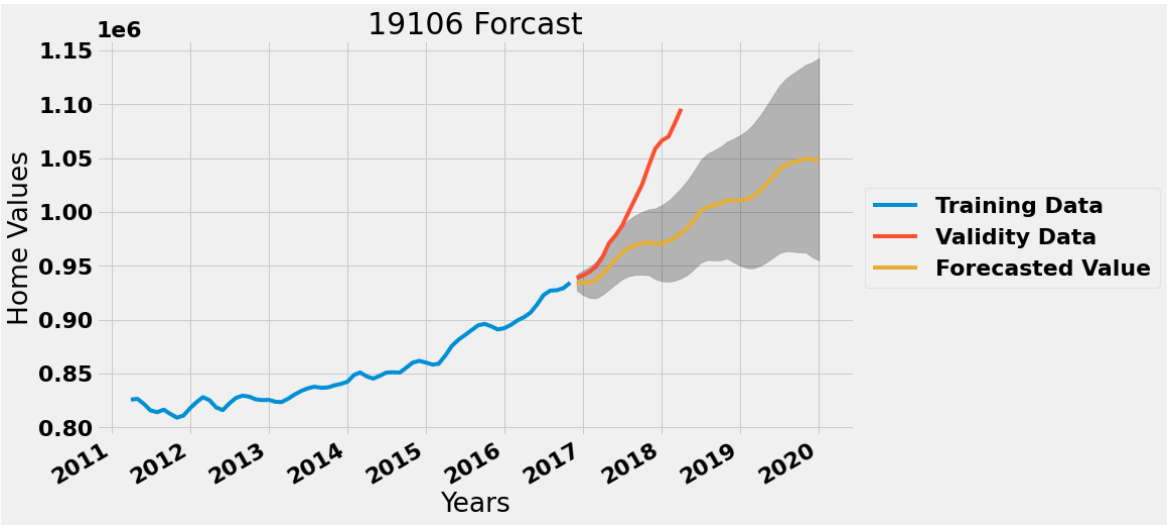

19103 Forcast
1e6
1.1
1.0
0.9
0.8
0.7
0.6
Home Values
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Years
Training Data
Validity Data
Forecasted Value

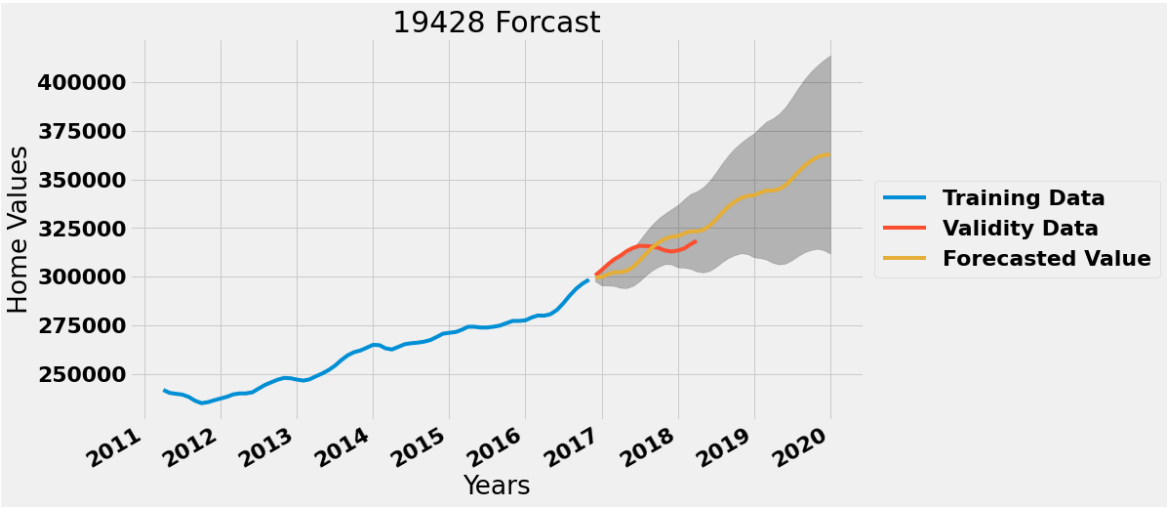

19147 Forcast
600000
500000
400000
300000
Home Values
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Years
Training Data
Validity Data
Forecasted Value


19123 Forcast
700000
600000
500000
400000
300000
Home Values
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Years
Training Data
Validity Data
Forecasted Value


19131 Forcast
125000
100000
75000
50000
25000
Training Data
Validity Data
Forecasted Value







In order to compare our models, lets calculate the RMSE (https://www.statology.org/how-to-interpret-rmse/) (Root Mean Square Error) of each model, and take the 5 models with the lowest RMSE as our best models, as these are the models who's predictions will have most accurately matched the validation data.

Type *Markdown* and LaTeX: α^2

In [36]:
```
# Finding the RMSE for all top 10 zip codes
rmses = []
for i in range(0, len(trainvalid_split)):
    rmses.append({
        'value': sqrt(mean_squared_error(trainvalid_split[i]['valid'],
        'zip_code': zip_list[i]
    })

# Sorting the RMSE's from lowest to highest (lowest indicating that th
sorted_rmses = sorted(rmses, key=lambda d: d['value'])
sorted_rmses
```

Out[36]:
```
[{'value': 1188.1884991178597, 'zip_code': '19148'},
 {'value': 4729.96288976945, 'zip_code': '19123'},
 {'value': 5551.760502149738, 'zip_code': '19130'},
 {'value': 6218.2858413917675, 'zip_code': '19145'},
 {'value': 6498.599797327099, 'zip_code': '19428'},
 {'value': 9557.453530464278, 'zip_code': '19125'},
 {'value': 16187.38275408695, 'zip_code': '19147'},
 {'value': 23286.896136253043, 'zip_code': '19131'},
 {'value': 40962.89267466064, 'zip_code': '19103'},
 {'value': 61282.51774356673, 'zip_code': '19106'}]
```

In [37]:
```
# Our final top 5 models!
top_5_rmses = sorted_rmses[:5]
top_5_rmses
```

Out[37]:
```
[{'value': 1188.1884991178597, 'zip_code': '19148'},
 {'value': 4729.96288976945, 'zip_code': '19123'},
 {'value': 5551.760502149738, 'zip_code': '19130'},
 {'value': 6218.2858413917675, 'zip_code': '19145'},
 {'value': 6498.599797327099, 'zip_code': '19428'}]
```

6.1 Final ROI: 5 Best Zip Codes

Now that we know which models best predicted the validation data, lets calculate the predicted ROI we expect to see by Jan 2020, using our last predicted value and our last observed value in the validation data.

In [38]:
```
# get the final predicted values from the top 10 zip code's data
forcasts = []
for i in range(0, len(arima_models)):
    forcasts.append({
        'value': arima_models[i]['model'].max(),
        'zip_code': zip_list[i]
    })

# forcasts
```

In [39]:
```
# taking the last forcasted value for the zipcodes selected based on t

forcasts_top_5 = []
for i in forcasts:
    for rmse in top_5_rmses:
        if rmse['zip_code'] == i['zip_code']:
            forcasts_top_5.append(i)
#forcasts_top_5
```

In [40]:
```
# get the final observed values of our validity data for our top 10 zi

observed = []
for i in range(0, len(trainvalid_split)):
    observed.append({
        'zip_code': zip_list[i],
        'value': trainvalid_split[i]['valid'].tail(1)
    })

# observed
```

In [41]:
```
# taking only the last observed value for the zipcodes selected based

observed_top_5 = []
for i in observed:
    for rmse in top_5_rmses:
        if rmse['zip_code'] == i['zip_code']:
            observed_top_5.append(i)

observed_top_5
```

Out[41]:
```
[{'zip_code': '19145',
  'value':                value
  time
  2018-04-01  208800.0},
 {'zip_code': '19148',
  'value':                value
  time
  2018-04-01  210100.0},
 {'zip_code': '19123',
  'value':                value
  time
  2018-04-01  409300.0},
 {'zip_code': '19130',
  'value':                value
  time
  2018-04-01  428700.0},
 {'zip_code': '19428',
  'value':                value
  time
  2018-04-01  318400.0}]
```

In [42]:
```
# calculating the ROI
roi_top_5 = []
for i in range(len(top_5_rmses)):
    roi_top_5.append(
        (forcasts_top_5[i]['value'] - observed_top_5[i]['value']) / ob
roi_top_5
```

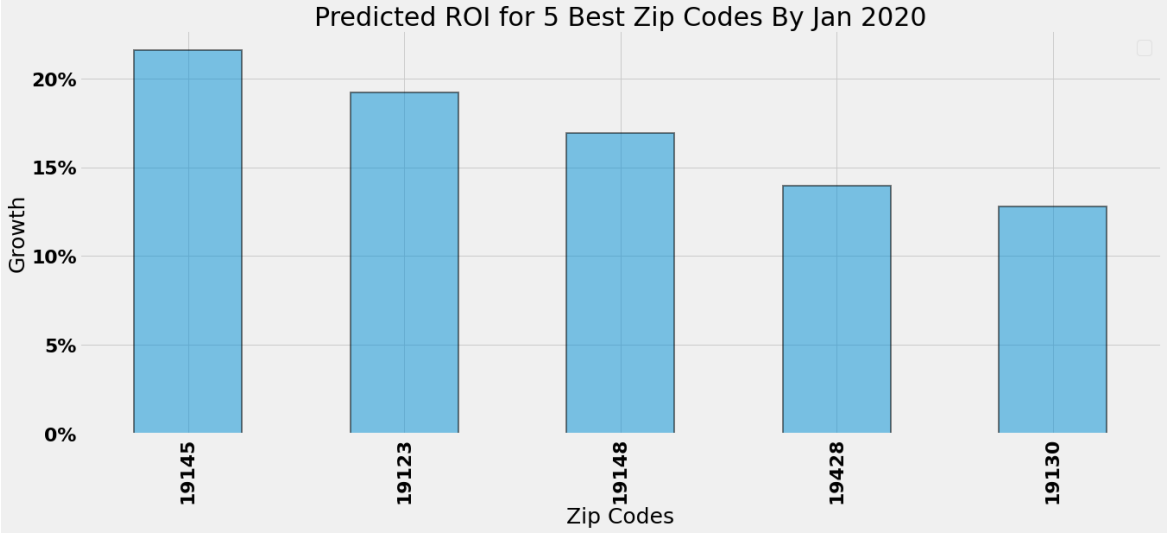
Out[42]:
```
[               value
 time
 2018-04-01  0.169448,
                value
 time
 2018-04-01  0.192087,
                value
 time
 2018-04-01  0.127729,
                value
 time
 2018-04-01  0.216095,
                value
 time
 2018-04-01  0.139479]
```

In [43]:
```
# changing dtype to numpy array
roi_array = []
for i in range(len(top_5_rmses)):
    roi_array.append(roi_top_5[i].to_numpy())

# changing numpy array into a list so we can put it into a data frame
roi_list = [i[0][0] for i in roi_array]

# final list of top 5 zip codes
final_zip = []
for zipcode in top_5_rmses:
    final_zip.append(zipcode['zip_code'])

#creating a DataFrame for predicted ROI and zipcodes
zip_df = pd.DataFrame(roi_list, columns =['Predicted ROI'])
zip_df['Zip Codes'] = final_zip

# sorting DataFrame by highest ROI
zip_df.sort_values('Predicted ROI', axis=0, ascending=False, inplace=T
zip_df
```

Out[43]:

	Predicted ROI	Zip Codes
3	0.216095	19145
1	0.192087	19123
0	0.169448	19148
4	0.139479	19428
2	0.127729	19130

In [44]:
```
# Visualization of ROI for the 5 best zip codes

ax = zip_df.plot.bar(x='Zip Codes', y='Predicted ROI', figsize=(
    20, 8), alpha=0.5, edgecolor="black", linewidth=2)
plt.title("Predicted ROI for 5 Best Zip Codes By Jan 2020", fontsize=3
plt.legend('')
ax.set_yticklabels(["0%", "5%", "10%", "15%", "20%"])
plt.xlabel('Zip Codes', fontsize=25)
plt.ylabel('Growth', fontsize=25);
```



6.2 Final Visualizations

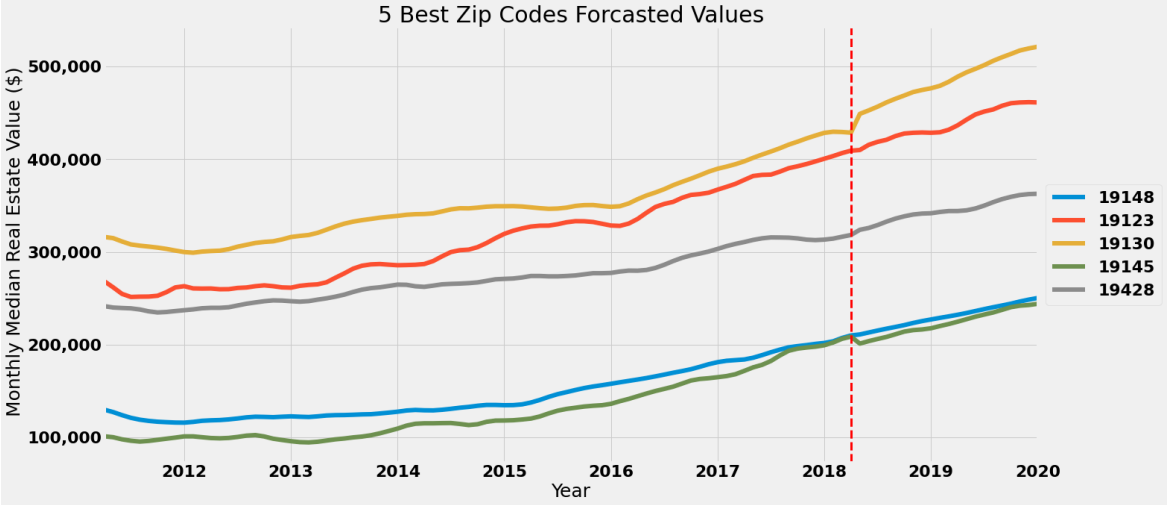
Now that we have the ROI's for the zip codes, lets create a visualization so we can better compare them.

In [45]:
```
# code inspiration from https://stackoverflow.com/questions/56891518/d
# getting the time series data for only our 5 best zipcodes
plot_df_5 = plot_df[plot_df.columns.intersection(final_zip)]
plot_df_5 = plot_df_5.loc[:,final_zip]
```

In [46]:
```
# getting the forcasted data for our top 10 zipcodes
pred_df = pd.DataFrame(arima_models[0])
pred_df.columns= [zip_list[0]]
for i in range(1, len(zip_list)):
    pred_df[zip_list[i]] = [i for i in arima_models[i]['model']]

# getting the forcasted data for our top 5 zipcodes
pred_df_5 = pred_df[pred_df.columns.intersection(final_zip)]
pred_df_5_fin = pred_df_5[-21:]
pred_df_5_fin = pred_df_5_fin.loc[:,final_zip]
```

In [47]:
```
#creating one large dataset with both our original and predicted data
fin_df_pred = pd.concat([plot_df_5, pred_df_5_fin], axis=0)
#fin_df_pred
```

In [48]:
```
# creating our lineplot, setting the color map and line width size
ax = fin_df_pred.plot.line(figsize=(20, 10), linewidth=6)
ax.axvline(pd.to_datetime('2018-04-01'), color='r', linestyle='--', lw
ax.set_yticklabels(
     [0, "100,000", "200,000", "300,000", "400,000", "500,000"])
ax.set_xlabel("Year", fontsize=25)
ax.set_ylabel(" Monthly Median Real Estate Value ($)", fontsize=25)
# code taken from https://stackoverflow.com/questions/4700614/how-to-p
ax.legend(final_zip, loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('5 Best Zip Codes Forcasted Values', fontsize=30)
plt.show();
```



7 Conclusion

7.0.1 Summary of Analysis Process

These zip codes were chosen by sorting the zip codes by highest ROI after the 2008 crash. The top ten zip codes were then split into training and validation sets, and modeled in order to predict how the real estate values would change over 3 years. Using the predicted data and the validation data, we then looked at the RMSE for each model, and picked out the models with the lowest RMSE scores as our best modeled zip codes. Finally, we predicted what our models suggested would be the ROI in Jan, 2020 for real estate purchased at the end of the data set (April, 2018).

Zip Codes	Final Observed Value		Predicted Increase (ROI)		Final Expected 2020 Value
19145	$ 208,800	+	$ 45,100.8 (21.6%)	=	$ 253,900.8
19123	$ 409,300	+	$ 78,585.6 (19.2%)	=	$ 487,885.6
19148	$ 210,100	+	$ 35,506.9 (16.9%)	=	$ 245,606.9
19428	$ 318,400	+	$ 44,257.6 (13.9%)	=	$ 362, 657.6
19130	$ 428,700	+	$ 55,302.3 (12.8%)	=	$ 484,002.3

** Note: This table was created outside of this notebook.

7.0.2 Recommendations

Based on the analysis performed here, we would recommend the following five zip codes as areas to investigate investment options.

Here is some additional information about these areas:

- `19145` : This zip code is located in south Philadelphia (like `19148`), whose urban areas have been gentrifying. It contains both urban and suburban housing, along with the FDR park (https://myphillypark.org/explore/parks/fdr-park/). Additionally, this zip code is right across from the major Philadelphia sports stadiums. As this zip code contains such diverse real estate options, it may be worth further investigation to find exactly which areas would be best to invest in. Alternatively, as prices in this zip code are fairly low, it may be worth investing in properties in different areas of this zip code.
- `19123` : This zip code is located right above center city, which is the area known for the most expensive real estate. It is also abuts `19125` right next to Fishtown - an area known for it's food, bars, and nightlife. It's proximity and good public transportation to both center city and Fishtown may be behind it's rising ROI.
- `19148` : This zip code is also located in south Philadelphia, and includes East Passyunk (https://www.visiteastpassyunk.com/), a funky commercial street nestled in a residential area. The area is also known to be gentrifying.
- `19428` : This is the zip code for Conshohocken, PA. Conshohocken is located 30-40 minutes away from Philadelphia. According to Wikipedia (https://en.wikipedia.org/wiki/Conshohocken,_Pennsylvania) "Historically a large mill town and industrial and manufacturing center, after the decline of industry in recent years Conshohocken has developed into a center of riverfront commercial and residential development."
- `19130` : This zip code is also located right above center city, which is the area known for the most expensive real estate, to the west of `19123`. It's ROI could be due to it's proximity to center city.

  It is important to note that both `19103` and `19106` (Rittenhouse Square (https://www.visitphilly.com/things-to-do/attractions/rittenhouse-square-park/) and the Historic District (https://www.visitphilly.com/articles/philadelphia/must-see-historic-attractions-in-historic-philadelphia/), respectively) models predictions *were lower* than the actual home prices. This indicates that these areas are hot real estate, but may also be more volatile investment. Additionally their buy-in costs are high, making them high risk, high reward - if the markets continue to perform well. `19131`, a neighborhood right next to Fairmount Park (https://www.visitphilly.com/things-to-do/attractions/fairmount-park/) also out performed the model, and should be further investigated in future analyses. Additionally it may be more accurate to model these upward trends in housing prices as an exponential function rather than a time series.

7.0.3 Possible Next Steps

- Further investigate where in `19145` is the best investment.
- Keep an eye on south Philadelphia's neighborhoods (including but not exclusively `19145` and `19148`).
- Keep an eye on the neighborhoods directly north of center city (`19123` and `19130`).
- Further investigate `19121` - it's current value is low, but seems to be rapidly growing