Real Estate Investing

Student name: T.J. Kyner \ Student pace: Full time \ Instructor: Abhineet Kulkarni \ Cohort: 040521

Project Overview

Data Source

Zillow, one of the top real estate listing platforms in the United States, provides access to a variety of data through its research portal and associated APIs. The data used in this project (`zillow_data.csv`) is sourced from the research portal and includes the monthly typical home prices of all homes (inclusive of single-family homes, condominiums, and co-operatives homes) per zip code.

Business Problem

The goal of this project is to act as a consultant to a fictional real estate invement firm and provide an answer to the following question:

> What are the top five best zip codes for us to invest in?

Specifically for this project, the focus will revolve around investment opportunities in **Columbus, OH**. The forecasted five-year return on investment (ROI), defined as the predicted percentage growth of the typical home value in a Columbus metro area zip code in five years, will serve as the evaluation metric for determing which zip codes to recommend.

Imports & Settings

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

# Utility tools
import json
import itertools
import warnings
warnings.filterwarnings('ignore')
from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter('ignore', ConvergenceWarning)

# Visualization
import folium
import seaborn as sns
import missingno as msno
import matplotlib.pyplot as plt
%matplotlib inline

# Modeling
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from pmdarima import auto_arima
```

In [2]:
```
sns.set_theme()
```

Data Preprocessing

Previewing and Summary Info

In [3]:
```
df = pd.read_csv('data/zillow_data.csv', dtype={'RegionName': str})
df.head()
```

Out[3]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2017-07	2017-08	2017-09	2017-10	2017-11	2017-12
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0	...	1005500	1007500	1007800	1009600	1013300	1018700
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700.0	...	308000	310000	312500	314100	315000	316600
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200.0	...	321000	320600	320200	320400	320800	321200
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0	...	1289800	1287700	1287400	1291500	1296600	1299000

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2017-07	2017-08	2017-09	2017-10	2017-11	2017-12
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300.0	...	119100	119400	120000	120300	120300	120300

5 rows × 272 columns

In [4]:
```
df.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
```

Looking at just the non-date columns:

In [5]:
```
df[df.columns[:7]].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   RegionID    14723 non-null  int64 
 1   RegionName  14723 non-null  object
 2   City        14723 non-null  object
 3   State       14723 non-null  object
 4   Metro       13680 non-null  object
 5   CountyName  14723 non-null  object
 6   SizeRank    14723 non-null  int64 
dtypes: int64(2), object(5)
memory usage: 805.3+ KB
```

Checking the number of unique cities, states, etc. included in the data:

In [6]:
```
df[df.columns[:7]].nunique()
```

Out[6]:
```
RegionID      14723
RegionName    14723
City           7554
State            51
Metro           701
CountyName     1212
SizeRank      14723
dtype: int64
```

Column Modifications

The `RegionID` and `SizeRank` columns do not contain useful information for the purposes of this project and will be dropped.

In [7]:
```
df.drop(columns=['RegionID', 'SizeRank'], inplace=True)
```

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

Out[8]:

	RegionName	City	State	Metro	CountyName	1996-04	1996-05	1996-06	1996-07	1996-08	...	2017-07	2017-08	2017-09	2017-10	2017-11	2017-12
0	60657	Chicago	IL	Chicago	Cook	334200.0	335400.0	336500.0	337600.0	338500.0	...	1005500	1007500	1007800	1009600	1013300	1018700
1	75070	McKinney	TX	Dallas-Fort Worth	Collin	235700.0	236900.0	236700.0	235400.0	233300.0	...	308000	310000	312500	314100	315000	316600
2	77494	Katy	TX	Houston	Harris	210400.0	212200.0	212200.0	210700.0	208300.0	...	321000	320600	320200	320400	320800	321200
3	60614	Chicago	IL	Chicago	Cook	498100.0	500900.0	503100.0	504600.0	505500.0	...	1289800	1287700	1287400	1291500	1296600	1299000
4	79936	El Paso	TX	El Paso	El Paso	77300.0	77300.0	77300.0	77300.0	77400.0	...	119100	119400	120000	120300	120300	120300

5 rows × 270 columns

Additionally, I'll rename the `RegionName` column to `ZipCode` for easier interpretation.

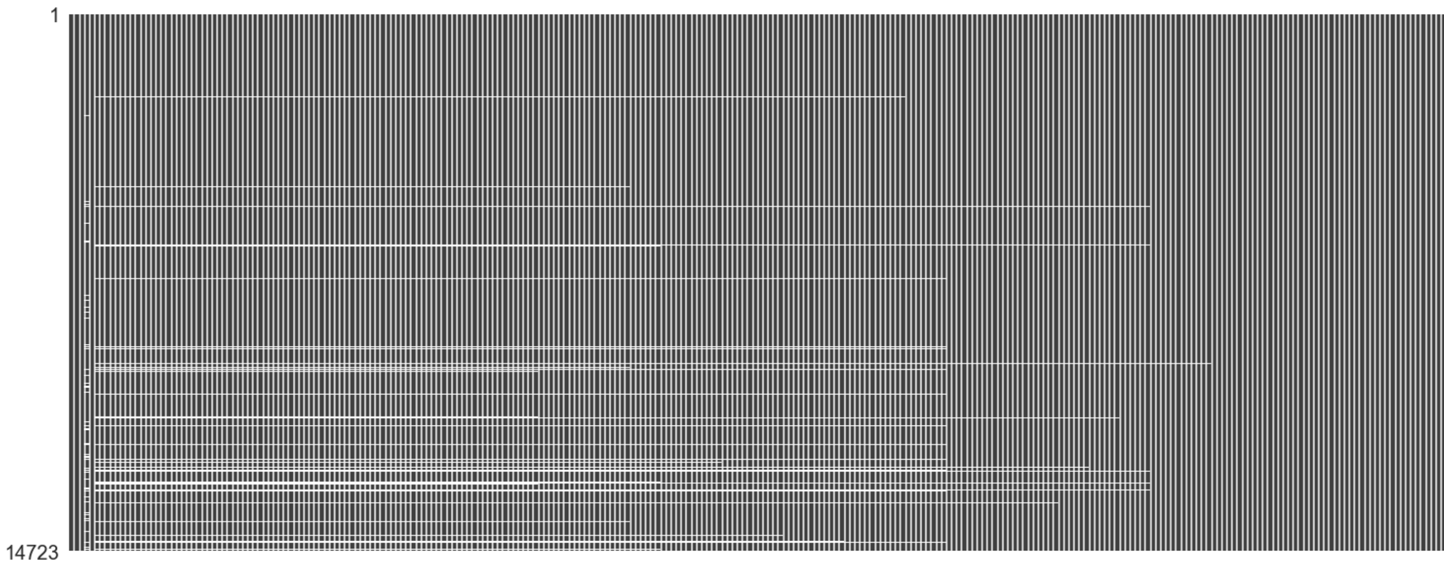
In [9]:
```
df.rename(columns={'RegionName': 'ZipCode'}, inplace=True)
```

In [10]:
```
df.columns[:5]
```

Out[10]:
```
Index(['ZipCode', 'City', 'State', 'Metro', 'CountyName'], dtype='object')
```

Missing Data

In [11]:
```
msno.matrix(df, sparkline=False);
```



In [12]:
```
# Looking only at the non-date columns
df[df.columns[:5]].isna().sum()
```

Out[12]:
```
ZipCode          0
City             0
State            0
Metro         1043
CountyName       0
dtype: int64
```

Observations:

- The `Metro` column is the only categorical feature with missing data. Given that certain cities do not reside within a greater metropolitan area, this is not too concerning.
- The horizontal white lines in the matrix show that certain zip codes are missing a sizable chunk of values for specific date ranges. This is also not cause for concern since the missing data appears to be for consecutive dates for each zip code impacted. This is likely due to one of two reasons:
  1. the zip codes did not exist before the date of the first instance of data or
  2. the data was not tracked for those zip codes until a certain date.

Melting the Data

While the data as it is currently organized (wide format) is useful for reading purposes, transforming it to long format is much more conducive for visualization and modeling purposes. This transformation is known as "melting" the data.

In [13]:
```
def melt_data(df):
    '''Transforms the dataframe from wide format to long format.'''
    melted = pd.melt(df, id_vars=['ZipCode', 'City', 'State', 'Metro', 'CountyName'], var_name='time')
    melted['time'] = pd.to_datetime(melted['time'], infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted

df_melted = melt_data(df)
df_melted.head()
```

Out[13]:

	ZipCode	City	State	Metro	CountyName	time	value
0	60657	Chicago	IL	Chicago	Cook	1996-04-01	334200.0
1	75070	McKinney	TX	Dallas-Fort Worth	Collin	1996-04-01	235700.0
2	77494	Katy	TX	Houston	Harris	1996-04-01	210400.0
3	60614	Chicago	IL	Chicago	Cook	1996-04-01	498100.0
4	79936	El Paso	TX	El Paso	El Paso	1996-04-01	77300.0

In [14]:
```
df_melted.shape
```

Out[14]:
```
(3744704, 7)
```

Adding FIPS Codes

Federal Information Processing System (FIPS) Codes are standardized codes representing unique states and counties within the United States. As defined by the Federal Communications Commission,

> FIPS codes are numbers which uniquely identify geographic areas. The number of digits in FIPS codes vary depending on the level of geography. State-level FIPS codes have two digits, county-level FIPS codes have five digits of which the first two are the FIPS code of the state to which the county belongs.

Including this information in the dataframe will be useful for visualization purposes within the Exploratory Data Analysis section. To do so, I have downloaded county-level FIPS Codes from the Natural Resources Conservation Service of the US Department of Agriculture and will be merging the data with `df_melted` for a final dataframe to work with.

In [15]:
```
# Explicitly setting the FIPS column dtype to `str` to preserve the leading zeroes
fips = pd.read_csv('data/fips_county_level.csv', dtype={'FIPS': str})
fips.head()
```

Out[15]:

	FIPS	Name	State
0	01001	Autauga	AL
1	01003	Baldwin	AL
2	01005	Barbour	AL
3	01007	Bibb	AL
4	01009	Blount	AL

In [16]:
```
df_final = pd.merge(left=df_melted,
                    right=fips,
                    how='left',
                    left_on=['CountyName', 'State'],
                    right_on=['Name', 'State'])

df_final.head()
```

Out[16]:

	ZipCode	City	State	Metro	CountyName	time	value	FIPS	Name
0	60657	Chicago	IL	Chicago	Cook	1996-04-01	334200.0	17031	Cook
1	75070	McKinney	TX	Dallas-Fort Worth	Collin	1996-04-01	235700.0	48085	Collin
2	77494	Katy	TX	Houston	Harris	1996-04-01	210400.0	48201	Harris
3	60614	Chicago	IL	Chicago	Cook	1996-04-01	498100.0	17031	Cook
4	79936	El Paso	TX	El Paso	El Paso	1996-04-01	77300.0	48141	El Paso

Quick check to ensure everything merged correctly:

In [17]:
```
df_final.isna().sum()
```

Out[17]:
```
ZipCode            0
City               0
State              0
Metro         236023
CountyName         0
time               0
value              0
FIPS               0
Name               0
dtype: int64
```

Dropping the redundant `Name` column that was added automatically during the merge process:

In [18]:
```
df_final.drop(columns='Name', inplace=True)
df_final.columns
```

Out[18]:
```
Index(['ZipCode', 'City', 'State', 'Metro', 'CountyName', 'time', 'value',
       'FIPS'],
      dtype='object')
```

Rearranging the columns for readability purposes:

In [19]:
```
df_final = df_final[['ZipCode', 'City', 'Metro', 'CountyName', 'State', 'FIPS', 'time', 'value']]
df_final.head()
```

Out[19]:

	ZipCode	City	Metro	CountyName	State	FIPS	time	value
0	60657	Chicago	Chicago	Cook	IL	17031	1996-04-01	334200.0
1	75070	McKinney	Dallas-Fort Worth	Collin	TX	48085	1996-04-01	235700.0
2	77494	Katy	Houston	Harris	TX	48201	1996-04-01	210400.0
3	60614	Chicago	Chicago	Cook	IL	17031	1996-04-01	498100.0
4	79936	El Paso	El Paso	El Paso	TX	48141	1996-04-01	77300.0

Exploratory Data Analysis

Typical Home Values by County

In this section, I'll be creating a function to plot the typical home values per county for a given date. This function will be able to operate at both a national level to view all available data as well as at a state-specific level for accessing more granular data in a quick manner.

In [20]:
```
with open('data/geojson_counties_fips.json') as f:
    counties = json.load(f)
```

In [21]:
```
state_fips = pd.read_csv('data/fips_state_level.csv', dtype={'FIPS': str})
state_fips.head()
```

Out[21]:

	State	Abbreviation	FIPS
0	Alabama	AL	01
1	Alaska	AK	02
2	Arizona	AZ	04
3	Arkansas	AR	05
4	California	CA	06

In [22]:
```
def get_state_fips(state):
    '''Given a state\'s two letter abbreviation, returns the proper FIPS code.'''
    return state_fips[state_fips.Abbreviation == state].FIPS.values[0]

def get_state_counties(state):
    '''Given a state\'s two letter abbreviation, returns all counties in the state.'''
    return [c for c in counties['features'] if c['properties']['STATE'] == get_state_fips(state)]

# https://stackoverflow.com/questions/12472338/flattening-a-list-recursively
def flatten(S):
    if S == []:
        return S
    if isinstance(S[0], list):
        return flatten(S[0]) + flatten(S[1:])
    return S[:1] + flatten(S[1:])

def get_min_bounds(state):
    '''Given a state\'s two letter abbreviation, returns the southmost and westmost boundaries.'''
    coords = [c['geometry']['coordinates'] for c in get_state_counties(state)]
    flat = flatten(coords)
    long = [val for i, val in enumerate(flat) if i % 2 == 0]
    lat = [val for i, val in enumerate(flat) if i % 2 != 0]
    return [min(lat), min(long)]

def get_max_bounds(state):
    '''Given a state\'s two letter abbreviation, returns the northmost and eastmost boundaries.'''
    coords = [c['geometry']['coordinates'] for c in get_state_counties(state)]
    flat = flatten(coords)
    long = [val for i, val in enumerate(flat) if i % 2 == 0]
    lat = [val for i, val in enumerate(flat) if i % 2 != 0]
    return [max(lat), max(long)]

# Testing the functions
print(get_min_bounds('OH'))
print(get_max_bounds('OH'))
```

```
[38.404338, -84.820157]
[41.977523, -80.518693]
```

In [23]:
```
counties['features'][0]
```

Out[23]:
```
{'type': 'Feature',
 'properties': {'GEO_ID': '0500000US01001',
  'STATE': '01',
  'COUNTY': '001',
  'NAME': 'Autauga',
  'LSAD': 'County',
  'CENSUSAREA': 594.436},
 'geometry': {'type': 'Polygon',
  'coordinates': [[[-86.496774, 32.344437],
    [-86.717897, 32.402814],
    [-86.814912, 32.340803],
    [-86.890581, 32.502974],
    [-86.917595, 32.664169],
    [-86.71339, 32.661732],
    [-86.714219, 32.705694],
    [-86.413116, 32.707386],
    [-86.411172, 32.409937],
    [-86.496774, 32.344437]]]},
 'id': '01001'}
```

In [24]:
```
def plot_choropleth(df, columns, date, geo_data, state=None, key_on='feature.id'):
    '''
    Description:
    ------------
    Creates a choropleth map with the option to automatically zoom to a
    specific state.

    Parameters:
    -----------
```

```
    df : pandas.DataFrame
        A dataframe that contains the values to be plotted.

    columns : str
        The names of the columns in the dataframe containing the data
        to be used. The first column name should reference the
        geographic identifier data such as FIPS codes. The second
        column should reference the actual values.

    date : str (YYYY-MM-DD)
        The date of the values to be plotted. Useful for examining
        changes over time.

    geo_data : str / obj
        A reference to the GeoJSON containing coordinates for
        geographic boundaries (zip codes, counties, states, etc.).

    state : str
        A state's two letter abbreviation. Will automatically zoom the
        choropleth to that state if supplied.

    key_on : str
        The name of the variable in the GeoJSON file that binds the data.

    Returns:
    --------
    Displays a choropleth map.
    '''

    if state:
        df = df[df.State == state]
    df = df[df.time == date]

    map = folium.Map(location=[48, -102],
                     tiles='cartodbpositron',
                     zoom_start=3)

    choro = folium.Choropleth(geo_data=geo_data,
                              data=df,
                              columns=columns,
                              key_on=key_on,
                              bins=6,
                              fill_color='YlOrRd',
                              nan_fill_opacity=0.3,
                              fill_opacity=0.7,
                              line_opacity=0.2,
                              legend_name='Median Home Value')

    choro.add_to(map)
    if state:
        map.fit_bounds([get_min_bounds(state), get_max_bounds(state)])
    return map
```

In [25]:

```
plot_choropleth(df_final,
                columns=['FIPS', 'value'],
                date='2018-04-01',
                geo_data=counties)
```

Out[25]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [26]:
```
plot_choropleth(df_final,
                columns=['FIPS', 'value'],
                date='2018-04-01',
                geo_data=counties,
                state='OH')
```

Out[26]: Make this Notebook Trusted to load map: File -> Trust Notebook

Largest Value Increases in Ohio

This section will only include those zip codes which have data beginning in April of 1996. The original, pre-melted dataframe in wide format will be easier to work with for this calculation and does not need any of the transformations or additions that occurred afterwards such as the FIPS Codes.

In [27]:
```
df_growth = df[['ZipCode', 'City', 'Metro', 'State', '1996-04', '2018-04']].dropna()
df_growth = df_growth[df_growth.State == 'OH']
df_growth.head()
```

Out[27]:

	ZipCode	City	Metro	State	1996-04	2018-04
80	44107	Lakewood	Cleveland	OH	96700.0	174700
92	44035	Elyria	Cleveland	OH	83000.0	91500
127	43081	Westerville	Columbus	OH	131300.0	242300
128	44060	Mentor	Cleveland	OH	131500.0	178400
134	43123	Grove City	Columbus	OH	110800.0	176500

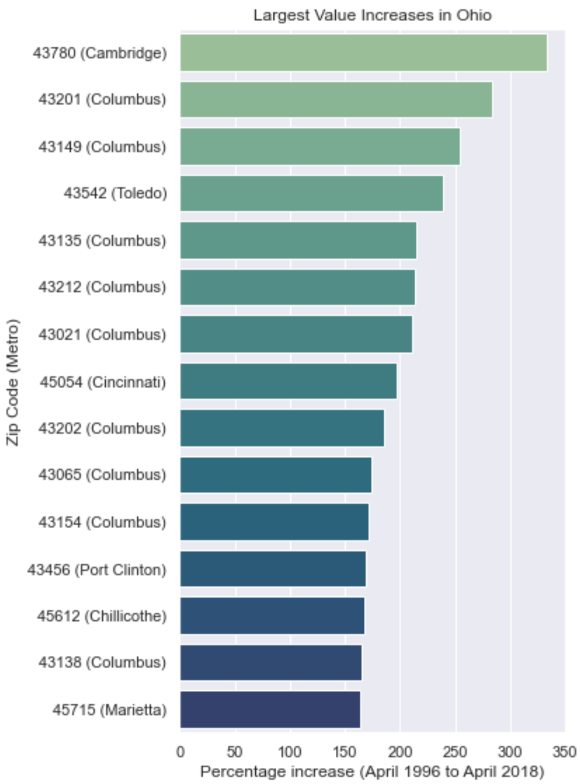
In [28]:
```
df_growth['growth'] = (df_growth['2018-04'] / df_growth['1996-04'] - 1)*100
df_growth.sort_values('growth', ascending=False, inplace=True)
df_growth.head()
```

Out[28]:

	ZipCode	City	Metro	State	1996-04	2018-04	growth
12897	43780	Senecaville	Cambridge	OH	27400.0	118800	333.576642
1492	43201	Columbus	Columbus	OH	70400.0	270300	283.948864
13125	43149	Rockbridge	Columbus	OH	40300.0	143000	254.838710
12080	43542	Monclova	Toledo	OH	80100.0	271400	238.826467
11222	43135	Laurelville	Columbus	OH	35300.0	111000	214.447592

In [113…
```
fig = plt.figure(figsize=(6, 8))
```

```python
sns.barplot(data=df_growth[:15],
            x='growth',
            y='ZipCode',
            orient='h',
            palette='crest')
plt.title('Largest Value Increases in Ohio')
plt.xlabel('Percentage increase (April 1996 to April 2018)')
plt.ylabel('Zip Code (Metro)')
plt.yticks(ticks= [i for i in range(15)],
           labels=[f'{df_growth.iloc[i].ZipCode} ({df_growth.iloc[i].Metro})' for i in range(15)])
plt.tight_layout()
plt.savefig('images/ohio_value_increases.png', facecolor='white', dpi=150);
```



Analysis of Columbus, Ohio

Subsetting a Columbus Metro Area Dataframe

In [30]:
```python
df_cbus_metro = df_final[(df_final.State == 'OH') & (df_final.Metro == 'Columbus')]
df_cbus_metro.head()
```

Out[30]:

	ZipCode	City	Metro	CountyName	State	FIPS	time	value
124	43081	Westerville	Columbus	Franklin	OH	39049	1996-04-01	131300.0
131	43123	Grove City	Columbus	Franklin	OH	39049	1996-04-01	110800.0
162	43130	Lancaster	Columbus	Fairfield	OH	39045	1996-04-01	78400.0
193	43230	Gahanna	Columbus	Franklin	OH	39049	1996-04-01	121700.0
240	43026	Hilliard	Columbus	Franklin	OH	39049	1996-04-01	135800.0

In [31]:
```python
df_cbus_metro.shape
```

Out[31]: (17031, 8)

Visualizing Top Columbus Metro Area Zip Codes

In [32]:
```python
with open('data/geojson_ohio_zip_codes.json') as f:
    ohio_zipcodes = json.load(f)
```

In [33]:
```python
plot_choropleth(df_cbus_metro,
                columns=['ZipCode', 'value'],
                date='2018-04-01',
                geo_data=ohio_zipcodes,
                state='OH',
                key_on='feature.properties.ZCTA5CE10')
```

Out[33]: Make this Notebook Trusted to load map: File -> Trust Notebook

As someone who lives in Columbus, I can confidently say that some of the zip codes shown in the above map that have been lumped into the Columbus metro area are fairly removed from the city in reality. As a result, I will be manually discarding some of the zip codes in order to preserve focus on the city proper limits and its immediate surroundings.

In [34]:
```
df_cbus_metro.ZipCode.nunique()
```

Out[34]: 69

In [35]:
```
zips_to_exclude = ['43138', '43149', '43130', '43112', '43107', '43113',
                   '43135', '43102', '43154', '43164', '43145', '43148',
                   '43150', '43155', '43140', '43143', '43146', '43162',
                   '43045', '43040', '43061', '43344', '43013', '43103',
                   '43105', '43046', '43064']

df_cbus_metro = df_cbus_metro[~df_cbus_metro.ZipCode.isin(zips_to_exclude)]
df_cbus_metro.ZipCode.nunique()
```

Out[35]: 42

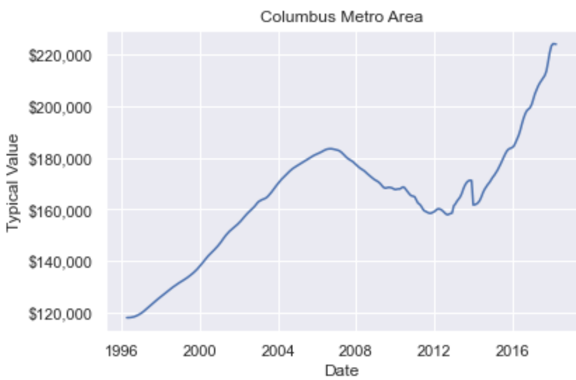
In [36]:
```
plot_choropleth(df_cbus_metro,
                columns=['ZipCode', 'value'],
                date='2018-04-01',
                geo_data=ohio_zipcodes,
                state='OH',
                key_on='feature.properties.ZCTA5CE10')
```

Out[36]: Make this Notebook Trusted to load map: File -> Trust Notebook

Average Value of Zip Codes in the Columbus Metro Area

In [117…

```
df_cbus_avg = df_cbus_metro.groupby('time').mean()
sns.lineplot(data=df_cbus_avg,
             x=df_cbus_avg.index,
             y='value')
plt.title('Columbus Metro Area')
plt.xlabel('Date')
plt.ylabel('Typical Value')
plt.gca().yaxis.set_major_formatter('${x:,.0f}')
plt.tight_layout()
plt.savefig('images/metro_area_value.png', facecolor='white', dpi=150);
```



Largest and Smallest Growth in the Columbus Metro Area

In [38]:

```
df_cbus_growth = df[['ZipCode', 'City', 'Metro', 'State', '1996-04', '2018-04']].dropna()
df_cbus_growth = df_cbus_growth[(df_cbus_growth.Metro == 'Columbus') &
                                (df_cbus_growth.State == 'OH')]
df_cbus_growth = df_cbus_growth[~df_cbus_growth.ZipCode.isin(zips_to_exclude)]

df_cbus_growth['growth'] = (df_cbus_growth['2018-04'] /
                            df_cbus_growth['1996-04'] - 1) * 100

df_cbus_growth.sort_values('growth', ascending=False, inplace=True)

df_cbus_growth.head()
```

Out[38]:

	ZipCode	City	Metro	State	1996-04	2018-04	growth
1492	43201	Columbus	Columbus	OH	70400.0	270300	283.948864
3147	43212	Grandview Heights	Columbus	OH	131500.0	411900	213.231939
8216	43021	Galena	Columbus	OH	128900.0	400600	210.783553
4067	43202	Columbus	Columbus	OH	89000.0	254100	185.505618
1713	43065	Powell	Columbus	OH	132600.0	362900	173.680241

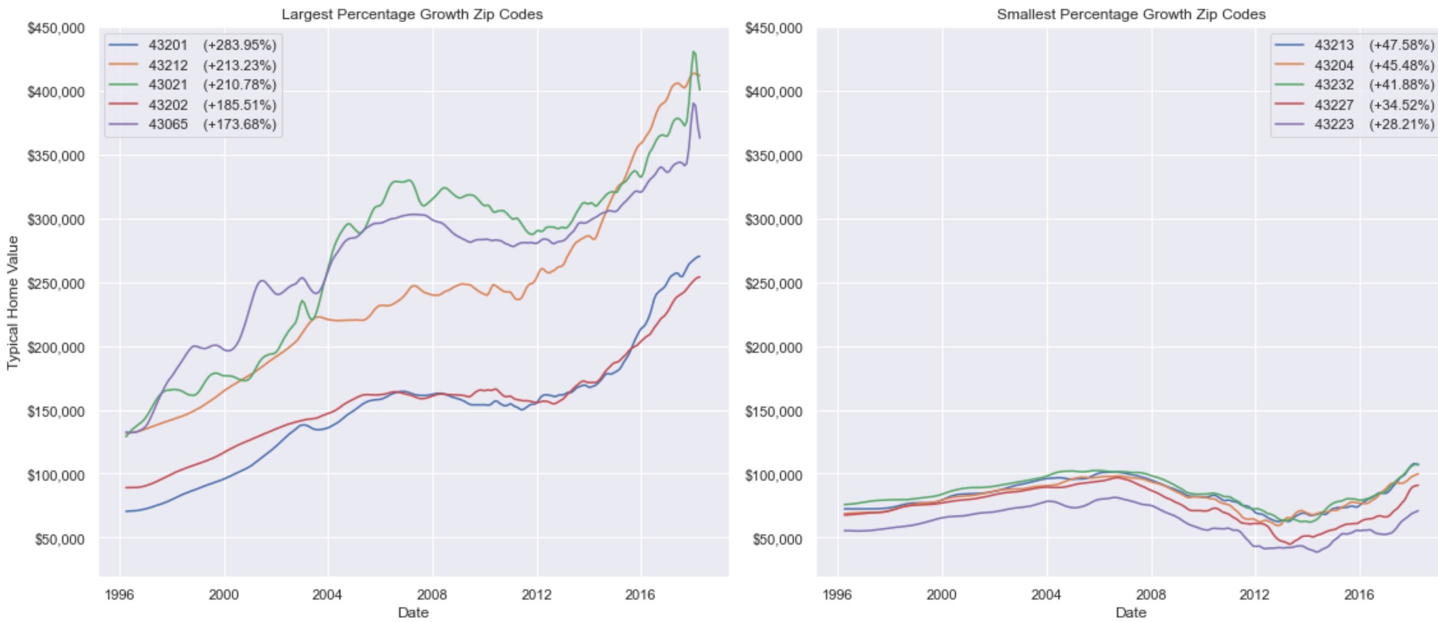
In [39]:

```
top_5 = df_cbus_growth.head()
bot_5 = df_cbus_growth.tail()
```

In [118…

```
fig, axes = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(16, 7))
axes = np.reshape(axes, -1)

for i, group in enumerate([top_5, bot_5]):
    for j, zipcode in enumerate(group.ZipCode.values):
        sns.lineplot(data=df_cbus_metro[df_cbus_metro.ZipCode == zipcode],
                     x='time',
                     y='value',
                     label=f'{zipcode}    (+{round(group.iloc[j].growth, 2)}%)',
                     ax=axes[i])

axes[0].set_title('Largest Percentage Growth Zip Codes')
axes[1].set_title('Smallest Percentage Growth Zip Codes')

for ax in axes:
    ax.set_xlabel('Date')
    ax.set_ylabel('Typical Home Value')
    ax.yaxis.set_tick_params(labelbottom=True)
    ax.yaxis.set_major_formatter('${x:,.0f}')
```

```
    ax.legend()

plt.tight_layout()
plt.savefig('images/top_and_bot_growth.png', facecolor='white', dpi=150);
```



Modeling

Baseline Model

To start the modeling process, I'll begin with creating a baseline model on a specific zip code. This model will then be refined and optimized for better performance. Ultimately, a process will be created to individually evaluate each of the zip codes in the Columbus metro area.

In [41]:
```
df_baseline = df_cbus_metro[df_cbus_metro.ZipCode == '43201']
df_baseline.head()
```

Out[41]:

	ZipCode	City	Metro	CountyName	State	FIPS	time	value
1469	43201	Columbus	Columbus	Franklin	OH	39049	1996-04-01	70400.0
15153	43201	Columbus	Columbus	Franklin	OH	39049	1996-05-01	70500.0
28837	43201	Columbus	Columbus	Franklin	OH	39049	1996-06-01	70600.0
42521	43201	Columbus	Columbus	Franklin	OH	39049	1996-07-01	70800.0
56205	43201	Columbus	Columbus	Franklin	OH	39049	1996-08-01	70900.0

In [42]:
```
df_baseline = df_baseline[['time', 'value']]
df_baseline.set_index('time', drop=True, inplace=True)
df_baseline = df_baseline.asfreq('MS')
df_baseline.head()
```

Out[42]:

	value
time	
1996-04-01	70400.0
1996-05-01	70500.0
1996-06-01	70600.0
1996-07-01	70800.0
1996-08-01	70900.0

In [43]:
```
model = SARIMAX(df_baseline,
                enforce_stationarity=False,
                enforce_invertibility=False)

output = model.fit()
output.summary()
```

Out[43]:

SARIMAX Results

Dep. Variable:	value	**No. Observations:**	265
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-2220.210

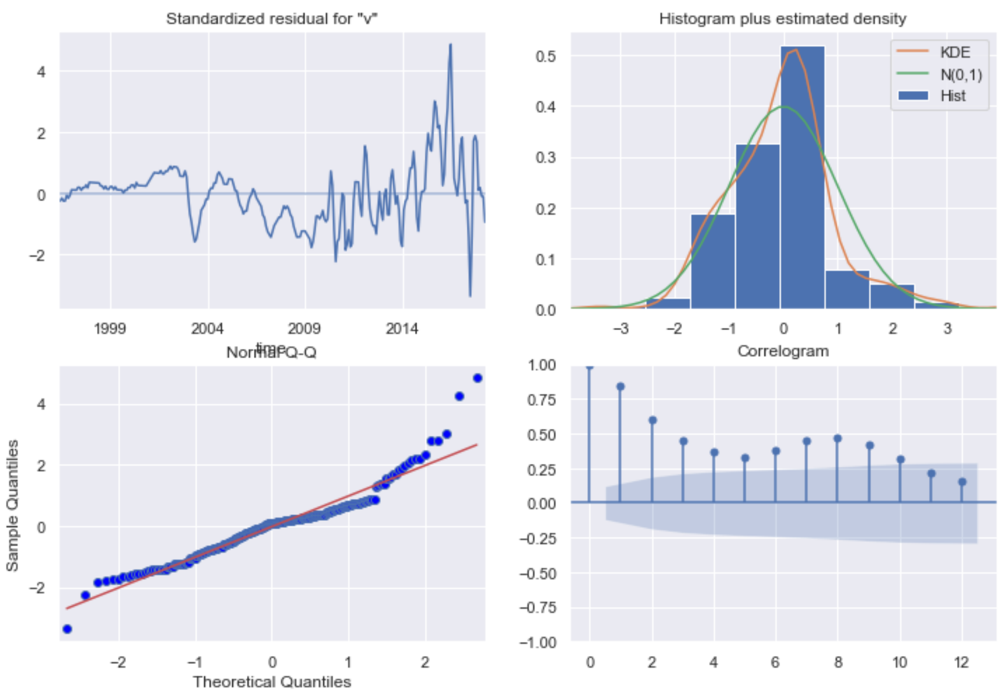
Date:	Tue, 03 Aug 2021	**AIC**	4444.420
Time:	21:51:37	**BIC**	4451.572
Sample:	04-01-1996	**HQIC**	4447.294
	- 04-01-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0053	0.000	2908.002	0.000	1.005	1.006
sigma2	1.181e+06	8.13e-13	1.45e+18	0.000	1.18e+06	1.18e+06

Ljung-Box (L1) (Q):	190.69	**Jarque-Bera (JB):**	151.55
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	7.17	**Skew:**	0.85
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.30

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 8.18e+32. Standard errors may be unstable.

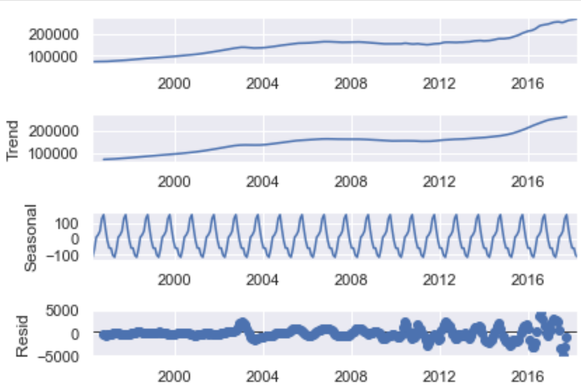
In [44]:
```
output.plot_diagnostics(lags=12, figsize=(12, 8));
```



Decomposition and Transformations

Decomposition

In [45]:
```
decomp = seasonal_decompose(df_baseline)
decomp.plot();
```



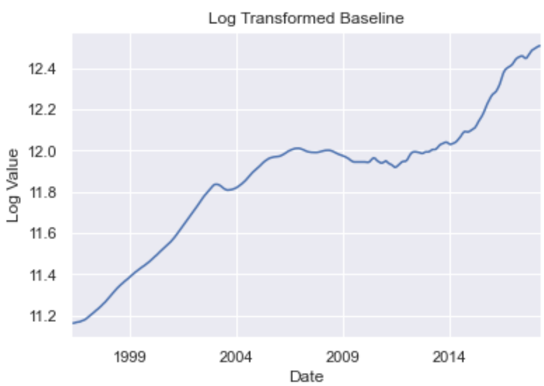
In [46]:
```
def adf_test(df):
    '''Prints the results of a Dickey-Fuller Test for a given pandas.DataFrame object.'''
    adf = adfuller(df)
    print('Results of Dickey-Fuller Test: \n')
    adf_output = pd.Series(adf[0:4],
                           index=['Test Statistic',
                                  'P-value',
                                  '# Lags Used',
                                  '# Observations Used'])

    for k, v in adf[4].items():
        adf_output[f'Critical Value ({k})'] = v

    print(adf_output)

adf_test(df_baseline)
```

```
Results of Dickey-Fuller Test:

Test Statistic            -0.086009
P-value                    0.950814
# Lags Used               13.000000
# Observations Used      251.000000
Critical Value (1%)       -3.456674
Critical Value (5%)       -2.873125
Critical Value (10%)      -2.572944
dtype: float64
```

Based on the results of the Dickey-Fuller test on the unmodified data, the time series as it currently stands is far from stationary. To correct this, I will apply a couple transformations.

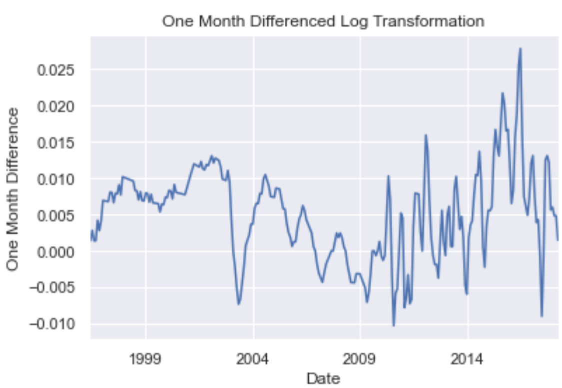
Log Transformation

In [47]:
```
df_log = np.log(df_baseline)
df_log.plot(legend=None)
plt.title('Log Transformed Baseline')
plt.xlabel('Date')
plt.ylabel('Log Value');
```



Differencing Transformation

In [48]:
```
df_log_diff = df_log.diff(periods=1).dropna()
df_log_diff.plot(legend=None)
plt.title('One Month Differenced Log Transformation')
plt.xlabel('Date')
plt.ylabel('One Month Difference');
```



In [49]:
```
adf_test(df_log_diff)
```

```
Results of Dickey-Fuller Test:

Test Statistic            -3.162733
P-value                    0.022246
# Lags Used                9.000000
```

```
# Observations Used     254.000000
Critical Value (1%)      -3.456360
Critical Value (5%)      -2.872987
Critical Value (10%)     -2.572870
dtype: float64
```

While not perfect, the above transformations significantly improved the stationarity of the time series. Now it's time to test its model performance.

Transformed Model

In [50]:
```
transformed_model = SARIMAX(df_log_diff,
                            enforce_stationarity=False,
                            enforce_invertibility=False)

transformed_output = transformed_model.fit()
transformed_output.summary()
```

Out[50]:

SARIMAX Results

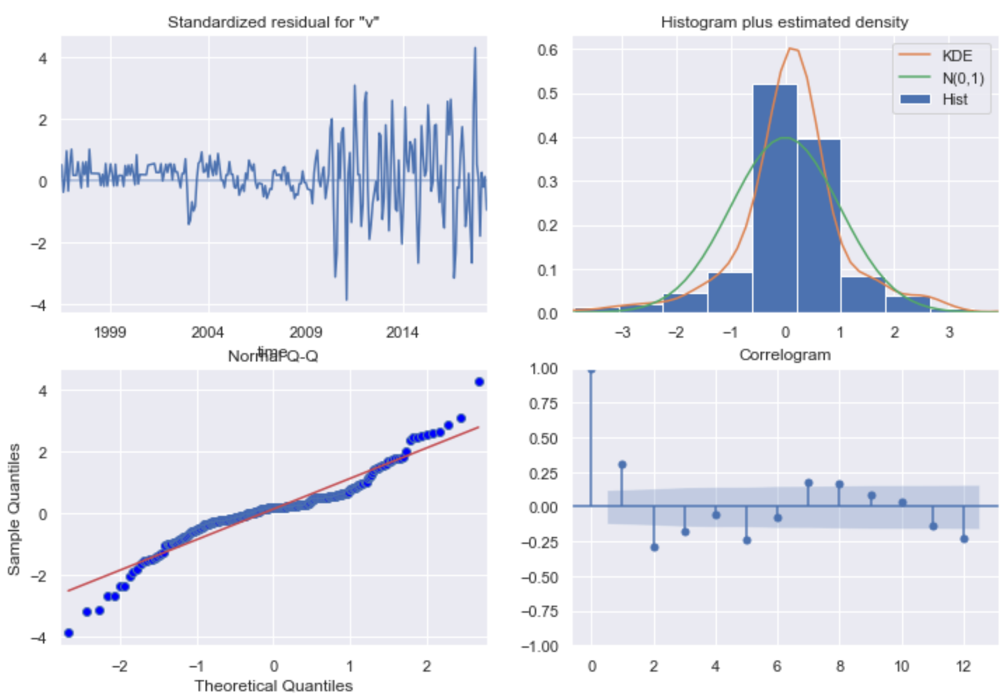
Dep. Variable:	value	**No. Observations:**	264
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	1147.323
Date:	Tue, 03 Aug 2021	**AIC**	-2290.645
Time:	21:51:40	**BIC**	-2283.501
Sample:	05-01-1996	**HQIC**	-2287.774
	- 04-01-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9233	0.020	45.870	0.000	0.884	0.963
sigma2	9.514e-06	5.38e-07	17.692	0.000	8.46e-06	1.06e-05

Ljung-Box (L1) (Q):	24.94	**Jarque-Bera (JB):**	113.31
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	10.53	**Skew:**	-0.08
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.21

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [51]:
```
transformed_output.plot_diagnostics(lags=12, figsize=(12, 8));
```



Hyperparameter Optimization

In [52]:
```
auto_model = auto_arima(df_log_diff,
                        information_criterion='aic',
                        n_jobs=-1,
```

```
                         trace=True,
                         random_state=42)

tuned_model = SARIMAX(df_log_diff,
                      order=auto_model.order,
                      seasonal_order=auto_model.seasonal_order,
                      enforce_stationarity=False,
                      enforce_invertibility=False)

print('\nModel tuning complete.')
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-2368.493, Time=0.29 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-2280.364, Time=0.14 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-2301.922, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-2335.742, Time=0.15 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-2282.363, Time=0.06 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-2326.055, Time=0.31 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-2358.564, Time=0.13 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-2355.154, Time=0.41 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-2373.761, Time=0.45 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-2377.071, Time=0.14 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-2380.908, Time=0.33 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-2327.064, Time=0.17 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=-2379.892, Time=0.20 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=-2377.398, Time=0.23 sec
 ARIMA(0,1,3)(0,0,0)[0]             : AIC=-2382.911, Time=0.07 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=-2329.077, Time=0.07 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=-2379.073, Time=0.10 sec
 ARIMA(0,1,4)(0,0,0)[0]             : AIC=-2381.896, Time=0.09 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-2366.524, Time=0.18 sec
 ARIMA(1,1,4)(0,0,0)[0]             : AIC=-2379.399, Time=0.10 sec

Best model:  ARIMA(0,1,3)(0,0,0)[0]
Total fit time: 3.676 seconds

Model tuning complete.
```

In [53]:

```
results = tuned_model.fit()
results.summary()
```

Out[53]:

SARIMAX Results

Dep. Variable:	value	**No. Observations:**	264
Model:	SARIMAX(0, 1, 3)	**Log Likelihood**	1175.933
Date:	Tue, 03 Aug 2021	**AIC**	-2343.867
Time:	21:51:44	**BIC**	-2329.640
Sample:	05-01-1996	**HQIC**	-2338.147
	- 04-01-2018		
Covariance Type:	opg		

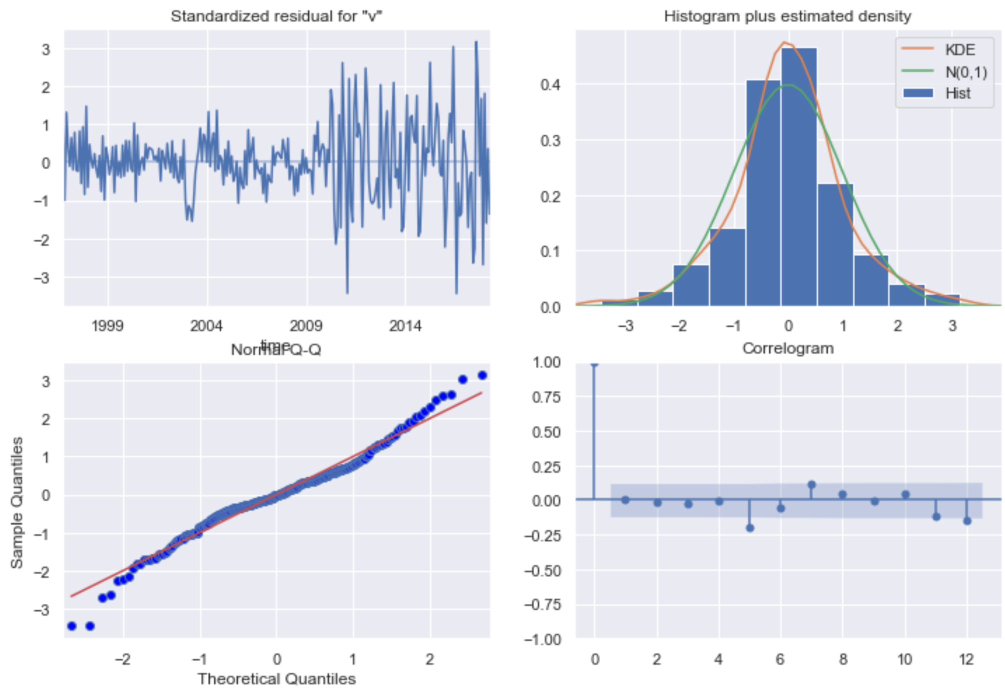
	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.4502	0.044	10.175	0.000	0.363	0.537
ma.L2	-0.3566	0.049	-7.241	0.000	-0.453	-0.260
ma.L3	-0.3721	0.048	-7.818	0.000	-0.465	-0.279
sigma2	6.636e-06	5.11e-07	12.989	0.000	5.63e-06	7.64e-06

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	19.47
Prob(Q):	0.92	**Prob(JB):**	0.00
Heteroskedasticity (H):	5.28	**Skew:**	-0.06
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.34

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [54]:

```
results.plot_diagnostics(lags=12, figsize=(12, 8));
```



Forecasting

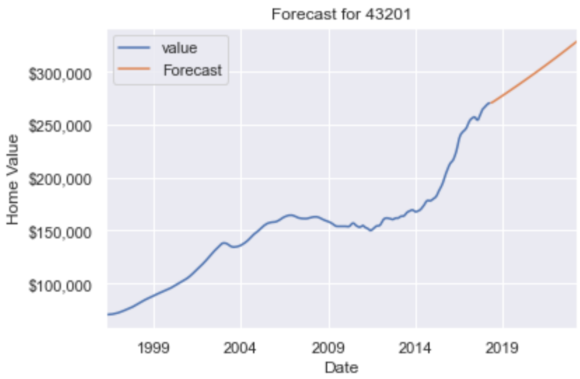
In [55]:
```python
def detransform(start_value, preds):
    '''
    Description:
    ------------
    Detransforms values that have had a log transformation and
    first differencing applied.

    Parameters:
    -----------
    start_value : float
        The last value of the historical data which serves as
        the starting point for the forecasted data. Used to
        rescale the data properly.

    preds : array-like
        Predictions that are in a transformed state (log and
        first differenced).

    Returns:
    --------
    Predictions that are detransformed and able to be visualized
    with historical data accurately.
    '''
    dediff = np.cumsum(preds)  # Undoing the differencing
    delog = np.exp(dediff)  # Undoing the log transformation
    output = delog * start_value
    return output
```

In [56]:
```python
predictions = results.get_forecast(60)
predictions = detransform(start_value=df_baseline.value[-1],
                          preds=predictions.predicted_mean)
predictions.head(10)
```

Out[56]:
```
2018-05-01    270601.144296
2018-06-01    271148.144770
2018-07-01    272049.664500
2018-08-01    272954.181624
2018-09-01    273861.706107
2018-10-01    274772.247950
2018-11-01    275685.817183
2018-12-01    276602.423873
2019-01-01    277522.078118
2019-02-01    278444.790052
Freq: MS, Name: predicted_mean, dtype: float64
```

In [116…
```python
ax = df_baseline.plot()
predictions.plot(ax=ax, label='Forecast')

ax.set_xlabel('Date')
ax.set_ylabel('Home Value')
ax.yaxis.set_major_formatter('${x:,.0f}')

plt.title('Forecast for 43201')
plt.legend()
plt.tight_layout()
```

```python
plt.savefig('images/example_forecast.png', facecolor='white', dpi=150)
plt.show()
```



Forecasted ROI

In [58]:
```python
roi = predictions[-1] / df_baseline.value[-1] - 1
print('Forecasted 5-year ROI for 43201:', f'{round(roi*100, 2)}%')
```

```
Forecasted 5-year ROI for 43201: 21.61%
```

Generating All Forecasted ROI Values

In order to select the top five most attractive zip codes in the Columbus metro area to invest in, each zip code now needs to be run through the modeling process. The following steps will be applied to each zip code:

1. Filter the `df_cbus_metro` dataframe to just the current zip code
2. Apply the log transformation and take the first difference
3. Optimize the hyperparameters for the model
4. Produce a five-year forecast with the optimized model
5. Calculate the forecasted five-year ROI for the current zip code

In addition to gathering the forecasted ROI values, I will also be saving the forecasted values themselves to retain the ability to plot any of the zip codes that may warrant further analysis.

In [59]:
```python
zipcodes = [zipcode for zipcode in df_cbus_metro.ZipCode.unique()]
print('Total number of unique zip codes:', len(zipcodes))
```

```
Total number of unique zip codes: 42
```

Step 1. Filter the df_cbus_metro dataframe to just the current zip code:

In [60]:
```python
def zipcode_filter(zipcode):
    '''Filters the Columbus metro area dataframe to a specific zip code.'''
    df = df_cbus_metro[df_cbus_metro.ZipCode == zipcode]
    df = df[['time', 'value']]
    df.set_index('time', drop=True, inplace=True)
    df = df.asfreq('MS')
    return df
```

Step 2. Apply the log transformation and take the first difference:

In [61]:
```python
def transform(df):
    '''Applies a log transformation and a first difference transformation.'''
    df = np.log(df)
    df = df.diff(periods=1).dropna()
    return df
```

Step 3. Optimize the hyperparameters for the model:

In [62]:
```python
def get_optimized_model(df):
    '''Fits a SARIMAX model to the given data with optimized hyperparameters.'''
    auto_model = auto_arima(df,
                            information_criterion='aic',
                            n_jobs=-1,
                            random_state=42)

    tuned_model = SARIMAX(df,
                          order=auto_model.order,
                          seasonal_order=auto_model.seasonal_order,
                          enforce_stationarity=False,
                          enforce_invertibility=False)

    return tuned_model.fit()
```

Step 4. Produce a five-year forecast with the optimized model:

In [63]:
```python
def get_forecast(model, start_value, n=60):
```

```
    '''Returns the forecasted values for a given model for a specified number of periods (n).'''
    forecast = model.get_forecast(n)
    forecast = detransform(start_value, forecast.predicted_mean)
    return forecast
```

Step 5. Calculate the forecasted five-year ROI for the current zip code:

In [64]:
```
def get_roi(start_value, end_value, return_format='decimal', round_to=4):
    '''
    Description:
    ------------
    Calculates the return on investment (ROI) for a given start and end value.
    Includes options for the format to return and the number of decimals to
    round to.

    Parameters:
    -----------
    start_value : float
        The beginning value.

    end_value : float
        The ending value.

    return_format : str
        Either 'decimal' or 'percentage'. 'decimal' will return the raw
        calculated value (e.g., 0.2051) while percentage will return the
        value multiplied by 100 (e.g., 20.51).

    round_to : int
        The number of decimal places to round to.

    Returns:
    --------
    The return on investment (ROI) for the given values.
    '''
    roi = end_value / start_value - 1
    if return_format == 'percentage':
        return round(roi*100, round_to)
    return round(roi, round_to)
```

Gathering all the forecasts and ROIs:

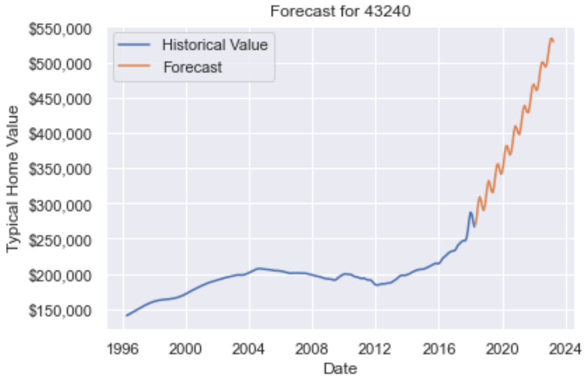
In [65]:
```
forecasts = dict()
rois = dict()

for i, zipcode in enumerate(zipcodes):
    print(f'Working on {i+1}/{len(zipcodes)}:', zipcode, end='\r')

    df = zipcode_filter(zipcode)
    start_value = df.value[-1]
    df = transform(df)
    model = get_optimized_model(df)
    forecast = get_forecast(model, start_value)
    roi = get_roi(start_value, end_value=forecast[-1])

    # Updating the dictionaries
    forecasts[zipcode] = forecast
    rois[zipcode] = roi

print('Finished gathering information.')
```

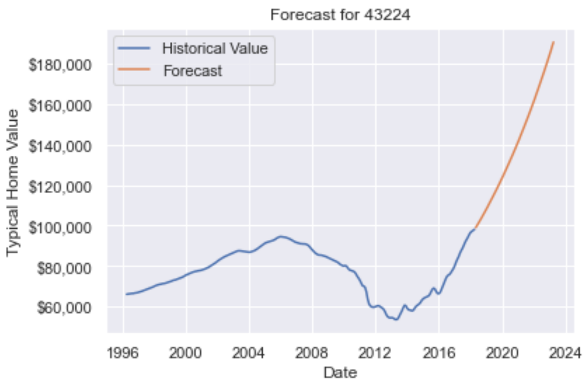
```
Finished gathering information.
```

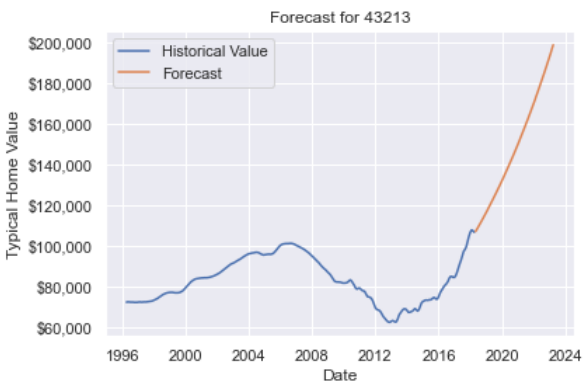
In [66]:
```
top_five = list(sorted(rois.items(), key=lambda x: x[1], reverse=True))[:5]
top_five
```

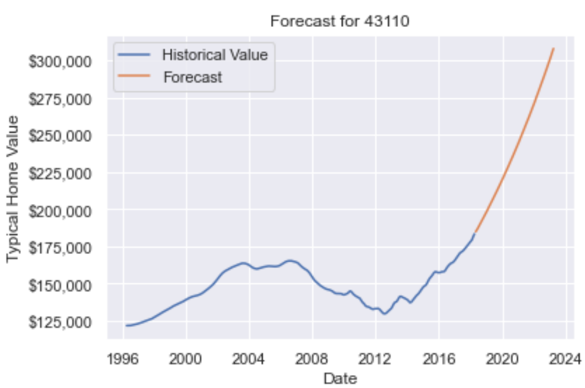
Out[66]:
```
[('43240', 0.9831),
 ('43224', 0.9444),
 ('43213', 0.8631),
 ('43110', 0.6794),
 ('43125', 0.6036)]
```

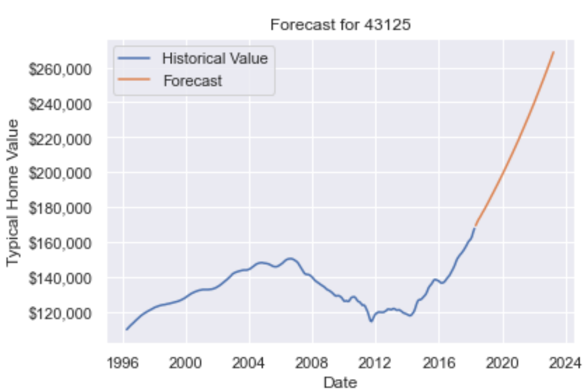
In [72]:
```
def plot_forecast(zipcode):
    '''Plots the historical and forecasted values for a given zip code.'''
    fig = plt.figure()
    plt.plot(zipcode_filter(zipcode), label='Historical Value')
    plt.plot(forecasts[zipcode], label='Forecast')
    plt.xlabel('Date')
    plt.ylabel('Typical Home Value')
    plt.gca().yaxis.set_major_formatter('${x:,.0f}')
    plt.title(f'Forecast for {zipcode}')
    plt.legend()

for z in top_five:
    plot_forecast(z[0])
```


Forecast for 43240
Historical Value
Forecast
Typical Home Value
$550,000
$500,000
$450,000
$400,000
$350,000
$300,000
$250,000
$200,000
$150,000
1996
2000
2004
2008
2012
2016
2020
2024
Date


Forecast for 43224
Historical Value
Forecast
Typical Home Value
$180,000
$160,000
$140,000
$120,000
$100,000
$80,000
$60,000
1996
2000
2004
2008
2012
2016
2020
2024
Date


Forecast for 43213
Historical Value
Forecast
Typical Home Value
$200,000
$180,000
$160,000
$140,000
$120,000
$100,000
$80,000
$60,000
1996
2000
2004
2008
2012
2016
2020
2024
Date


Forecast for 43110
Historical Value
Forecast
Typical Home Value
$300,000
$275,000
$250,000
$225,000
$200,000
$175,000
$150,000
$125,000
1996
2000
2004
2008
2012
2016
2020
2024
Date


Forecast for 43125
Historical Value
Forecast
Typical Home Value
$260,000
$240,000
$220,000
$200,000
$180,000
$160,000
$140,000
$120,000
1996
2000
2004
2008
2012
2016
2020
2024
Date

Conclusion

Results

After modeling, optimizing, and forecasting values for the typical home in different zip codes in the Columbus metropolitan area, the following five zip codes offer the greatest predicted five-year returns on investment:

- 43240: +98.31%
- 43224: +94.44%
- 43213: +86.31%
- 43110: +67.94%
- 43125: +60.36%

Next Steps

While this project offered significant insight, there's a multitude of ways in which it could be enhanced. Some of these ways include:

- Building an interactive dashboard for easier exploration of the data and forecasts
- Factoring in additional data such as new developments in order to better capture trending areas
- Using alternative evaluation metrics for what defines a good investment opportunity such as calculating the capitalization rate given assumed financing and operating costs instead of investing for appreciation