Rajendiran_Lab2

May 16, 2021

- Course: IST 718
- Authors: Sathish Kumar Rajediran
- Task: Lab 2
- Task Item: ***Best Zipcode Recommendation to Invest based on average household value***
- Date: May 9,2021

[ ]:
```
!pip install --upgrade setuptools
```

[ ]:
```
import datetime
import time
starttime = time.time()
s1 = datetime.datetime.now()
print ("Start date and time:", s1.strftime("%Y-%m-%d %H:%M:%S"))

!conda install -y fbprophet

s2 = datetime.datetime.now()
print ("End date and time:", s2.strftime("%Y-%m-%d %H:%M:%S"))
print("...runtime: {:.2f} minutes".format((time.time() - starttime)/60.0))
```

[3]:
```
# s2-s1
```

0.1 Objective

This case study provides an opportunity to demonstrate our ability to combine datasets and produce meaningful analysis. Specifically, we would like to provide a decision maker with more than just data—we want to provide insights, understanding, and wisdom. This exercise allows the student an opportunity to demonstrate progress (or mastery) of learning objectives 1, 2, 3, 4, and 5. * 1) Bold: Obtain data and understand data structures and data elements. * 2) Scrub data using scripting methods, to include debugging, for data manipulation in R and other tools. * 3) Explore data using essential qualitative analysis techniques, including descriptive statistics. * 4) Model relationships between data using the appropriate analytical methodologies matched to the information and the needs of clients and users. * 5) Interpret the data, model, analysis, and findings, and communicate the results in a meaningful way.

0.2 Instructions

- The research question is can we predict which three zip codes provide the best investment opportunity for the Syracuse Real Estate Investment Trust (SREIT)?
- Using the base data available from Zillow (files.zillowstatic.com/research/public/Zip/Zip_Zhvi_SingleFamily
  - Review the data – clean as appropriate
  - Provide an initial data analysis to include (but not limited to):
    - Develop time series plots for the following Arkansas metro areas:
      - Hot Springs, Little Rock, Fayetteville, Searcy
      - Present all values from 1997 to present
      - Average at the metro area level
- Using data from Zillow:
  - Develop model(s) for forecasting average median housing value by zip code for 2018
  - Use the historical data from 1997 through 2017 as your training data
  - Integrate data from other sources (think Bureau of Labor Statistics and Census data) to improve upon your base model(s)
- Answer the following questions:
  - What technique/algorithm/decision process did you use to down sample? (BONUS FOR NOT DOWN SAMPLING)
  - What three zip codes provide the best investment opportunity for the SREIT?
  - Why?
- Bonus: Develop a geographic visualization that in your view best depicts the data and recommendations:
  - By state
  - Median housing for Dec (state average)

0.3 Additional Instructions

- Don't forget what you learned in your previous courses; do your own work, document any assistance, use comments for clarity.
- Use python to conduct your analysis and produce your graphics

0.4 Submission Items

- Report with graphics
- Supporting notebook for the report with final data set

0.5 Loading and Cleaning the Data

[ ]:
```
#
↪************************************************************************
#       import libraries
#
↪************************************************************************

# standard library
```

```python
import os
import sys
import datetime
import time
import timeit
import warnings
import random
warnings.filterwarnings("ignore")

# import packages for analysis and modeling
import pandas as pd  # data frame operations
import numpy as np  # arrays and math functions
import itertools
import types
import math

# Import required packages for time series and model summary

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
from fbprophet import Prophet

# from scipy.stats import uniform  # for training-and-test split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# from  multiprocessing import Pool, cpu_count

#Visualization packages
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt


print('Libraries imported successfully!\n')
os.getcwd()
```

[7]:
```python
# # Show all columns and do not truncate in the data frame
# pd.set_option('display.max_columns', None)
# pd.set_option('display.max_colwidth', None)
```

0.6 Pre-Processing

Obtain & Scrub the data

1. Data set containing data for every city in the U.S. plus average household value between the periods of Januray, 1996 and March, 2020.
2. Data set containing zip codes for every metropolitan area in the U.S.

[122]:
```
#␣
↪***********************************************************************************
#       Working with files
#␣
↪***********************************************************************************

# Read the csv files into dataframes

# !pwd

# fpath = "/Users/sathishrajendiran/ist718-python/Labs/Lab2/"
fpath = "/content/datalab/TimeSeries/"

try:
    zipdata = pd.read_csv(fpath + "Zip_Zhvi_SingleFamilyResidence.csv",␣
↪encoding='ISO-8859-1')
    print('zipdata data - Total Number of rows Processed: ',len(zipdata))
except:
    print("Is the file in correct directory?")
```

```
zipdata data - Total Number of rows Processed:  30464
```

[11]:
```
#Lets start with data until December, 2019; Remaining months will be included␣
↪in the later part
zipdata = zipdata.iloc[:, 0:297]
```

[ ]:
```
# summary statistics
summary = zipdata.describe()
summary.head()
```

Remove NAs

1. Most of the metro sections values are NAs (7144). Lets replace these NAs from using public dataset containing Metro Names by Zipcode + City and County level

[ ]:
```
# Missing values
na_values = zipdata.isna().sum()
# print(na_values)
```

[15]:
```
#␣
↪***********************************************************************************
#       Working with files
#␣
↪***********************************************************************************
# Read the csv files into dataframes
# !pwd
# fpath = "/Users/sathishrajendiran/ist718-python/Labs/Lab2/"
```

```python
try:
    msazip = pd.read_csv(fpath + "fs11_gpci_by_msa-zip.csv")
    print('msazip data - Total Number of rows Processed: ',len(msazip))
except:
    print("Is the file in correct directory?")
```

```
msazip data - Total Number of rows Processed:  43772
```

[18]:
```python
# Keep data with missing metro name.
missing_metro = zipdata[zipdata['Metro'].isna()]
missing_index = missing_metro.index

# Drop observation without Metro data. These observations will then be␣
↪reattached
df_metro_complete = zipdata.drop(index=missing_index)

# Keep zip code and msa name from the metro_area data set
metro_area_simp = msazip[['ZIP CODE', 'MSA Name']]

# Merge missing metro with metro_area_simp
metro_complete = missing_metro.merge(metro_area_simp, how='left',␣
↪left_on='RegionName', right_on='ZIP CODE')
```

[ ]:
```python
# Check to see that there are no NaN variables in the new column.
print(metro_complete['MSA Name'].isna().sum())

# There are no NaN values now. We'll now replace the Metro column with the MSA␣
↪Name column, drop the overlapping columns, and
# reattach the observations to the df_data_1 data set.
metro_complete['Metro'] = metro_complete['MSA Name']
```

[24]:
```python
metro_complete1 = metro_complete.iloc[:, 0:297]
metro_complete1.head()
```

[24]:
```
   RegionID  RegionName RegionType StateName State             City  \
0     58148         926        Zip        LA    LA           Zwolle
1     58011         612        Zip        MI    MI         Frederic
2     58069         727        Zip        AR    AR     Walnut Ridge
3     70178       28734        Zip        NC    NC         Franklin
4     91865       77351        Zip        TX    TX  West Livingston

                      Metro       CountyName  SizeRank  1996-01  …     \
0  San Juan-Caguas-Guaynabo    Sabine Parish        15      NaN  …
1  San Juan-Caguas-Guaynabo  Crawford County       902      NaN  …
2  San Juan-Caguas-Guaynabo  Lawrence County      1341      NaN  …
3   NC NONMETROPOLITAN AREA    Macon County      2724  78118.0  …
4   TX NONMETROPOLITAN AREA     Polk County      3043      NaN  …
```

```
   2019-03  2019-04  2019-05  2019-06  2019-07  2019-08  2019-09  2019-10  \
0   103444   104314   104318   103966   103633   103125   102581   102384
1   124844   126730   128471   129738   129190   127661   126351   125146
2    89964    90545    91114    91324    91745    91521    91633    92007
3   168915   169177   169542   169938   170354   170693   171058   171345
4   130289   130916   131408   131885   132441   133215   133926   134508

   2019-11  2019-12
0   102502   102663
1   124569   123821
2    92444    92555
3   171499   171068
4   134796   135063

[5 rows x 297 columns]
```

[ ]:

```python
# Reattach data to original data.
zipdata = df_metro_complete.append(metro_complete1)
print(zipdata.shape)
print(zipdata.info())
```

[27]:

```python
# Missing values
na_values = zipdata.isna().sum()
# print(na_values)
```

[ ]:

```python
s1 = datetime.datetime.now()
print ("Start date and time:", s1.strftime("%Y-%m-%d %H:%M:%S"))
# Obtain column names of year-month combinations.
year_month = zipdata.iloc[:, 9:].columns.values

df_data_2 = zipdata

# Iterate over year-month combination, replacing NaN with average value for␣
↪metropolitan area
for ym in df_data_2[year_month]:
    df_data_2[ym] = df_data_2.groupby('Metro')[ym].transform(lambda x: x.
↪fillna(x.mean()))

s2 = datetime.datetime.now()
print ("End date and time:", s2.strftime("%Y-%m-%d %H:%M:%S"))
```

[ ]:

```python
# Columns that want to be kept
to_keep = ['RegionID', 'RegionName', 'City', 'State', 'Metro', 'CountyName',␣
↪'SizeRank']

# Columns to be converted to observations
```

```
to_obs = df_data_2.iloc[:, 9:]

# Pivot the data to have on observation for each region for each year-month.
df_data_pivot = pd.pivot_table(df_data_2, values=to_obs, columns=to_keep)

# Reset the index and rename year-month and values columns for easy␣
↪identification.
df_data_pivot = df_data_pivot.reset_index()
df_data_pivot.rename(columns={'level_0':'date', 0:'value'}, inplace=True)

# Convert data column to datetime
df_data_pivot['date'] = pd.to_datetime(df_data_pivot['date'],␣
↪infer_datetime_format=True)

# Print first five rows
df_data_pivot.sort_values(['RegionID', 'date']).head()
```

[ ]:

```
list(df_data_pivot.columns.values)
```

[ ]:

```
# Check the size of the data.
# print(df_data_pivot.shape)

# Check each columns data type so that there aren't any mismatched types (i.e␣
↪RegionID should be an object, not o number)
# print(df_data_pivot.dtypes)

# RegionID and Region name are identification tags and should therefore be␣
↪objects. SizeRank is an ordinal value and should also be converted to an␣
↪object
df_data_pivot['RegionID'] = df_data_pivot['RegionID'].astype(object)
df_data_pivot['RegionName'] = df_data_pivot['RegionName'].astype(object)
df_data_pivot['SizeRank'] = df_data_pivot['SizeRank'].astype(object)
```

[38]:

```
# Print the first and last date of the data
print('Oldest observation: ', df_data_pivot['date'].min())
print('Newest observation: ', df_data_pivot['date'].max(), '\n')

# Print the number of unique states, cities, and zipcodes
print('Number of states: ', len(df_data_pivot['State'].unique()))
print('Number of cities: ', len(df_data_pivot['City'].unique()))
print('Number of zipcodes: ', len(df_data_pivot['RegionID'].unique()))
```

```
Oldest observation:  1996-01-01 00:00:00
Newest observation:  2019-12-01 00:00:00

Number of states:  51
Number of cities:  14862
```

```
Number of zipcodes:  30464
```

[ ]:
```
# Sort the data by state and print the unique values as an array.
df_data_pivot.sort_values('State')['State'].unique()
```

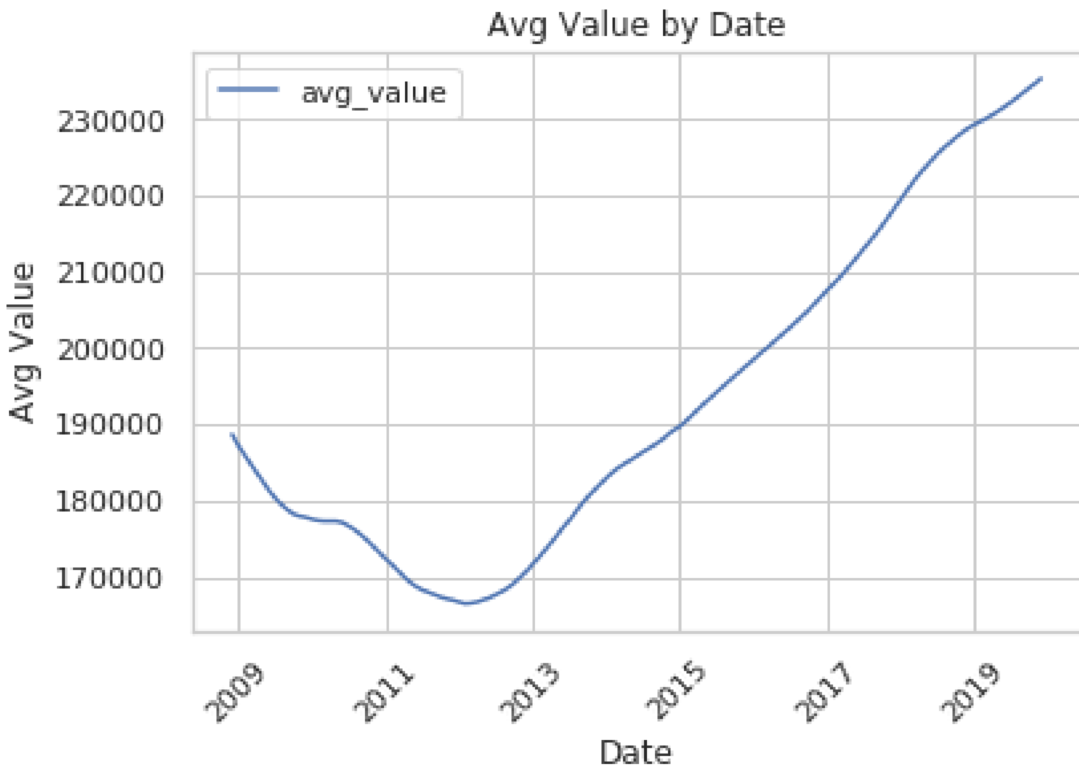
0.7 Exploratory Analysis

> Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone. — > John Tukey

[112]:
```
# Create a new data frame to average value per date. Keep observations starting␣
↪from December 2008 to plot.
national = df_data_pivot.groupby('date')['value'].mean()
national = national.reset_index()
national.columns = ['date', 'avg_value']
national = national[national['date']>='2008-12-01']
print(national.head())

# Plot the data as a line plot.
# Enlarge the plot
plt.figure(figsize=(12,9))
national.plot(x='date', y='avg_value')
_ = plt.xlabel('Date')
_ = plt.ylabel('Avg Value')
_ = plt.title('Avg Value by Date')
# Rotate x-labels
plt.xticks(rotation=45)
plt.show()
```

```
          date      avg_value
155 2008-12-01  188688.365757
156 2009-01-01  187228.194062
157 2009-02-01  186042.240772
158 2009-03-01  184870.897173
159 2009-04-01  183624.110633
```

```
<matplotlib.figure.Figure at 0x7f7d95248b70>
```



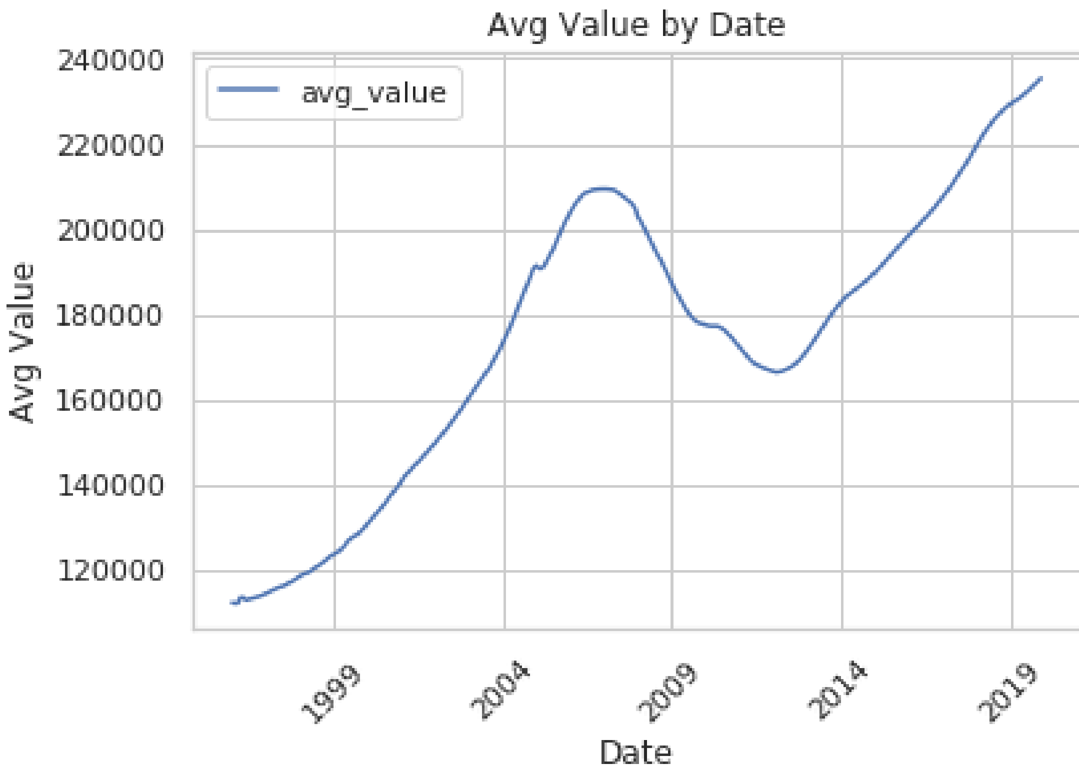
[123]:

```
# Create a new data frame to average value per date. Keep observations starting␣
↪from January 21996008 to plot.
national_all = df_data_pivot.groupby('date')['value'].mean()
national_all = national_all.reset_index()
national_all.columns = ['date', 'avg_value']
national_all = national_all[national_all['date']>='1996-01-01']
print(national_all.head())

# Plot the data as a line plot.
# Enlarge the plot
plt.figure(figsize=(20,12))
national_all.plot(x='date', y='avg_value')
_ = plt.xlabel('Date')
_ = plt.ylabel('Avg Value')
_ = plt.title('Avg Value by Date')
# Rotate x-labels
plt.xticks(rotation=45)
plt.show()
```

```
         date      avg_value
0  1996-01-01  112527.568630
1  1996-02-01  112327.604581
```

```
2 1996-03-01  112358.491573
3 1996-04-01  113714.942722
4 1996-05-01  113733.305915
```

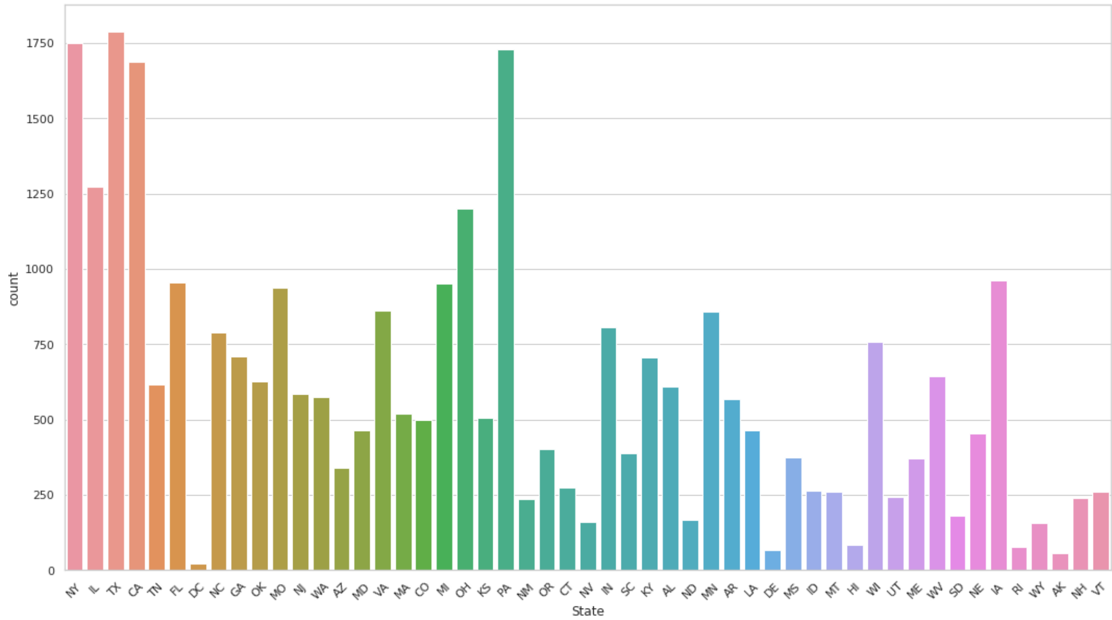
```
<matplotlib.figure.Figure at 0x7f7d9475e160>
```



As this graph presents, between 1996 and 2020 - US real estate market has seen twice hitting the roof and once had downfall. Yes, the downfall was due to the recession that hit the US in late 2008 and lasted until 2012. Ever since the market has been on the rise. Lets look at state level if the story is true across.

[114]:
```
# Enlarge the plot
plt.figure(figsize= (18,10))
sns.set(style = 'whitegrid')
sns.countplot(df_data_2['State'])

# Rotate x-labels
plt.xticks(rotation=45)
```

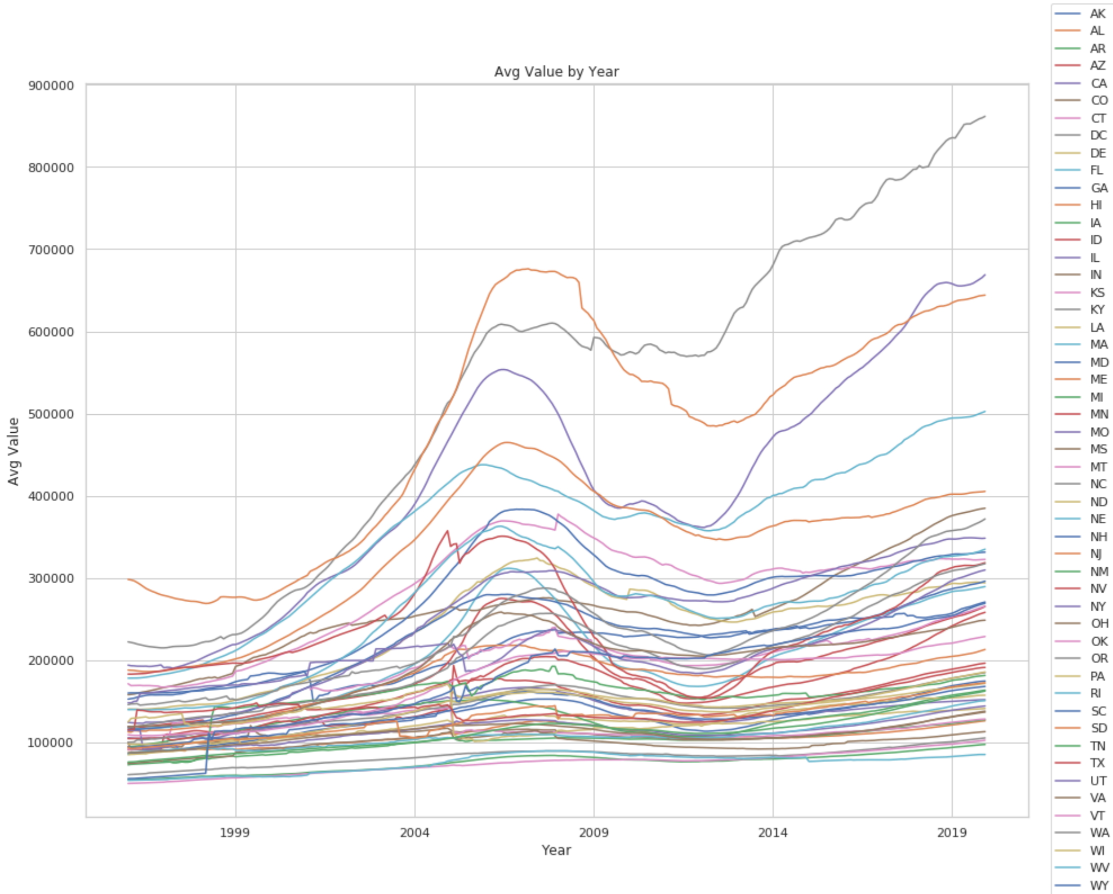
[114]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]),
 <a list of 51 Text xticklabel objects>)
```



[42]:

```
# Create a new data frame starting from December 2008 but looking at values at␣
↪the state level
state = df_data_pivot.groupby(['date', 'State'])['value'].mean()
state = state.reset_index()
state.columns = ['date', 'state', 'avg_value']
state = state[state['date']>='1996-01-01']
print(state.head())

# Pivot the data frame to produce multiple line plots
state_df = state.pivot(index='date', columns='state', values='avg_value')

# Plot the data
state_df.plot(figsize=(15,12))
_ = plt.xlabel('Year')
_ = plt.ylabel('Avg Value')
_ = plt.title('Avg Value by Year')
plt.legend(loc='center right', bbox_to_anchor=[1.1, 0.5])
plt.show()
```

```
         date state      avg_value
0 1996-01-01    AK  152514.153846
1 1996-01-01    AL   92708.648153
2 1996-01-01    AR   54911.793080
3 1996-01-01    AZ  118824.561143
4 1996-01-01    CA  193902.496039
```



Hmmm, Intresting. Its a fact that the real estate pricing has been on the rise after 2012, but the average increase is not consistent across states.Legends on the map is very helpful.But, There are atleast 4 states (DC, CA, HI, MA and NJ) with average price has increased by 400,000 by 2020.

[43]:
```python
# Retain observations for December 2019. Then filter states with values above␣
↪350,000
dec_2019 = state[state['date']=='2019-12-01']
top_three = dec_2019[dec_2019['avg_value']>=350000]

print(top_three[['state', 'avg_value']].sort_values('avg_value',␣
↪ascending=False))
```

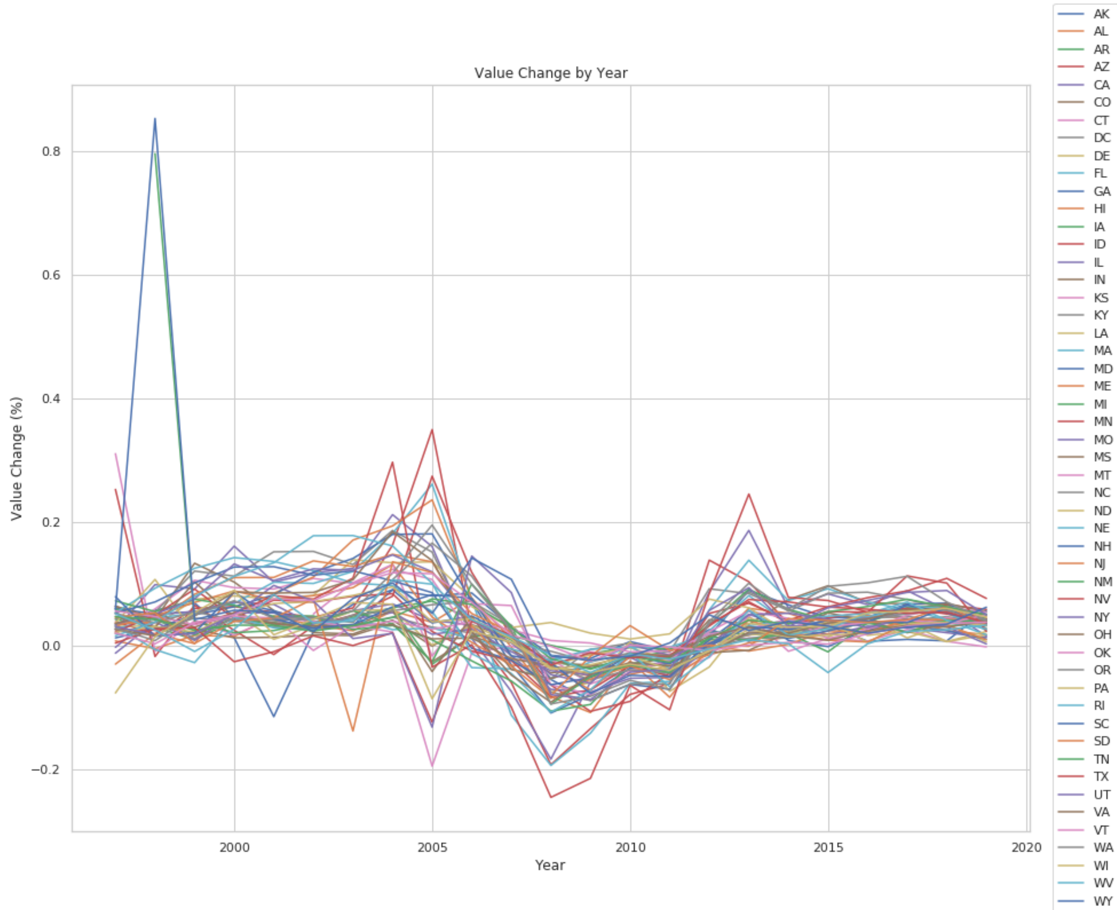
```
      state      avg_value
14644    DC  861657.272727
14641    CA  668897.169431
14648    HI  644084.440476
14656    MA  502607.244701
14668    NJ  405352.664384
```

```
14642    CO  384815.366733
14684    WA  371799.055749
```

[44]:

```python
# Identify December observations
dec_obs = state['date'].map(lambda x: x.month) == 12
state_dec = state[dec_obs]

# Calculate value percentage change for each state
state_dec['pct_change'] = state_dec['avg_value'].div(state_dec.
 ↪groupby('state')['avg_value'].shift(1))-1
state_dec = state_dec.dropna()
print(state_dec.sort_values(['state', 'date']).head())

# Plot the percentage changes for each state.
state_pct_df = state_dec.pivot(index='date', columns='state',␣
 ↪values='pct_change')

# Plot the data
state_pct_df.plot(figsize=(15,12))
_ = plt.xlabel('Year')
_ = plt.ylabel('Value Change (%)')
_ = plt.title('Value Change by Year')
plt.legend(loc='center right', bbox_to_anchor=[1.1, 0.5])
plt.show()
```

```
           date state      avg_value  pct_change
1173 1997-12-01    AK  163082.615385    0.033684
1785 1998-12-01    AK  172434.384615    0.057344
2397 1999-12-01    AK  184291.785714    0.068765
3009 2000-12-01    AK  186616.357143    0.012614
3621 2001-12-01    AK  165166.028061   -0.114943
```



[ ]:
```
dec_2019_pct = state_dec[state_dec['date']=='2019-12-01']
pct_10 = dec_2019_pct[dec_2019_pct['pct_change']>0.05]
print(pct_10[['state', 'pct_change']].sort_values('pct_change',␣
 ↪ascending=False))
```

[ ]:
```
print(pct_10[['state', 'pct_change']].sort_values('pct_change', ascending=True))
```

[48]:
```
# Filter the data, keeping only that for Florida and it's metro area.
florida = df_data_pivot[df_data_pivot['State'] == 'FL']
florida.head()
```

[48]:
```
            date  RegionID  RegionName            City  State  \
1232  1996-01-01     59738        4956       Pensacola     FL
9751  1996-01-01     71730       32003  Fleming Island     FL
9752  1996-01-01     71734       32008        Branford     FL
9753  1996-01-01     71735       32009      Bryceville     FL
9754  1996-01-01     71736       32011        Callahan     FL
```

```
                             Metro       CountyName SizeRank     value
1232  Pensacola-Ferry Pass-Brent  Escambia County    23350   94609.0
9751                Jacksonville      Clay County     4340  146152.0
9752      FL NONMETROPOLITAN AREA  Suwannee County    12313   42010.0
9753                Jacksonville    Nassau County    16025   92317.0
9754                Jacksonville    Nassau County     7549   74670.0
```

Now, lets look at one of the states the real estates available from 1996.Yes, Florida it is. Lets zoom in on Florida around the metro areas to see what kind of an increase it had over the years.

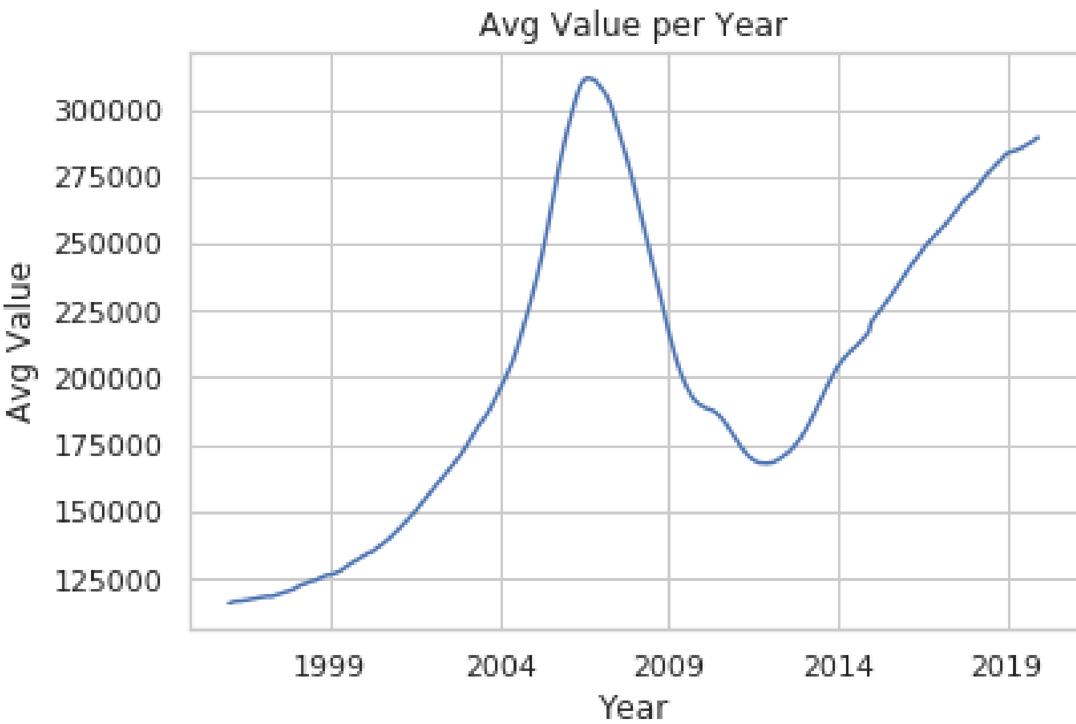
[ ]:
```python
# Let's first look at how the data is distributed and how many NaN there are
print(florida.describe())
print(len(florida))
```

[ ]:
```python
# Keep NaN observations
nan_values = florida[np.isnan(florida['value'])==True]

# Count NaN by Metro
nan_values.groupby('Metro')['Metro'].count()
```

[ ]:
```python
df_data_2[(df_data_2['Metro'] == 'Key West') | (df_data_2['Metro'] ==␣
↪'Sebring')]
```

[ ]:
```python
# Variation over the last five months.
print(florida.groupby('date')['value'].describe().tail())
```

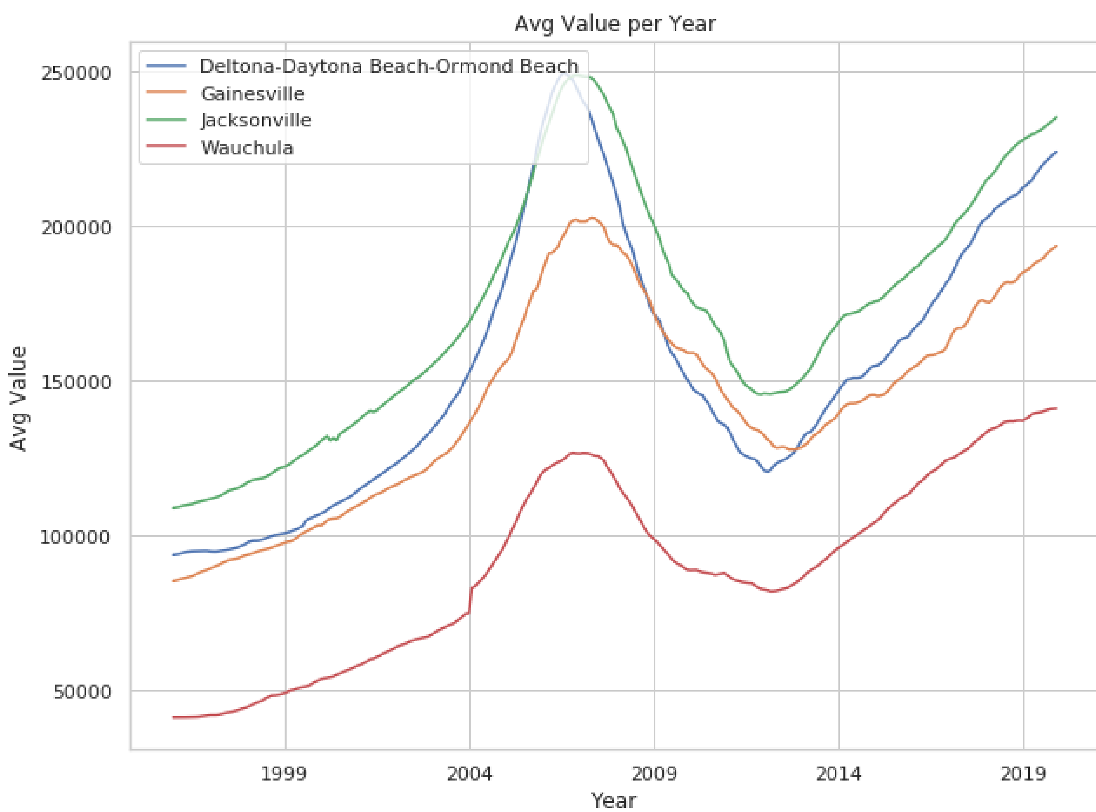
[53]:
```python
florida_avg = florida.groupby('date')['value'].mean()
florida_avg.plot(x='date', y='value')
_ = plt.xlabel('Year')
_ = plt.ylabel('Avg Value')
_ = plt.title('Avg Value per Year')
plt.show()
```




```
[ ]: # Find the top and bottom two florida metro areas, using percentage growth␣
     ↪between 2018 and 1996.
     fl_96 = florida[florida['date']=='1996-01-01']
     fl_19 = florida[florida['date']=='2019-12-01']

     # Get the average household values by metro area
     fl_96_avg = fl_96.groupby('Metro')['value'].mean().reset_index()
     fl_19_avg = fl_19.groupby('Metro')['value'].mean().reset_index()

     # Merge the two sets and find the highest percentage change
     fl_96_19_avg = fl_96_avg.merge(fl_19_avg, on='Metro', how='inner')
     fl_96_19_avg.columns = ['Metro', 'value_96', 'value_19']
     fl_96_19_avg['Pct_change'] = (fl_96_19_avg['value_19']/
     ↪fl_96_19_avg['value_96']) - 1
     fl_96_19_avg.sort_values('Pct_change', ascending=False)
```

```
[55]: # Keep metro areas of Wauchula, Deltona-Daytona Beach-Ormond Beach,␣
      ↪Gainesville, and Jacksonville
      florida_metro = florida[(florida['Metro']=='Wauchula') |
                              (florida['Metro']=='Deltona-Daytona Beach-Ormond␣
      ↪Beach') |
                              (florida['Metro']=='Gainesville') |
                              (florida['Metro']=='Jacksonville')]
```

```
# Calculate average household value of the metro areas and plot
florida_metro_avg = florida_metro.groupby(['date', 'Metro'])['value'].mean()
florida_metro_avg = florida_metro_avg.reset_index()

# Pivot the table
florida_metro_avg_pivot = florida_metro_avg.pivot(index='date',␣
↪columns='Metro', values='value')

# Plot
florida_metro_avg_pivot.plot(figsize=(10,7.5))
_ = plt.xlabel('Year')
_ = plt.ylabel('Avg Value')
_ = plt.title('Avg Value per Year')
plt.legend(loc='upper left')
plt.show()
```

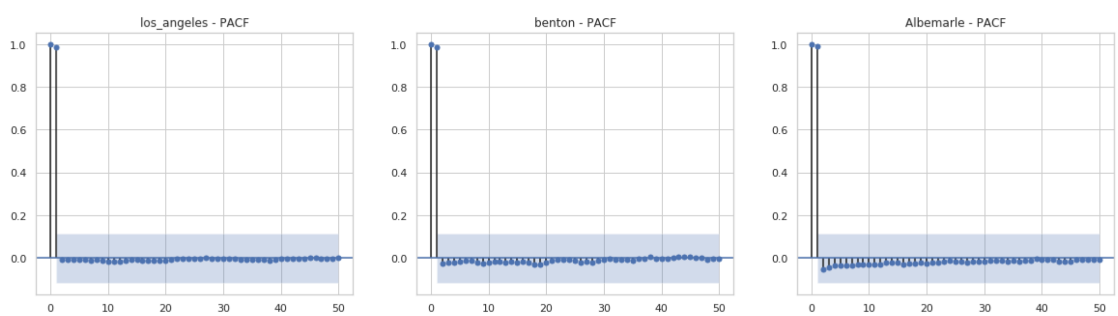


With no surprises, recession had an impact on Florida aswell between 2008 and 2012. However, from there jacksonville and Deltona-Daytona beach-Ormand Beach metro area had seen catching up with its historical rise and alsmot 200% rise in the market. Wauchulla is steadily increasing considering its low investment cost. Risk is low as well.

0.8 Model Building

---

[56]:
```
# Select a zip code to represent our data
fremont = df_data_pivot[df_data_pivot['RegionName']==94536]
benton = df_data_pivot[df_data_pivot['RegionName']==72712]
los_angeles = df_data_pivot[df_data_pivot['RegionName']==90210]
Albemarle = df_data_pivot[df_data_pivot['RegionName']==22911]

# Set figure size
plt.figure(figsize=(20,5))

# Plot autocorrelation plot for 94536, 72712, 90210, and 22911 zip code to␣
↪identify parameter starting point for ARIMA model

ax1 = plt.subplot(131)
plot_pacf(los_angeles.value, lags=50, ax=ax1)
_ = plt.title('los_angeles - PACF')

ax2 = plt.subplot(132)
plot_pacf(benton.value, lags=50, ax=ax2)
_ = plt.title('benton - PACF')

ax3 = plt.subplot(133)
plot_pacf(Albemarle.value, lags=50, ax=ax3)
_ = plt.title('Albemarle - PACF')


plt.show()
```



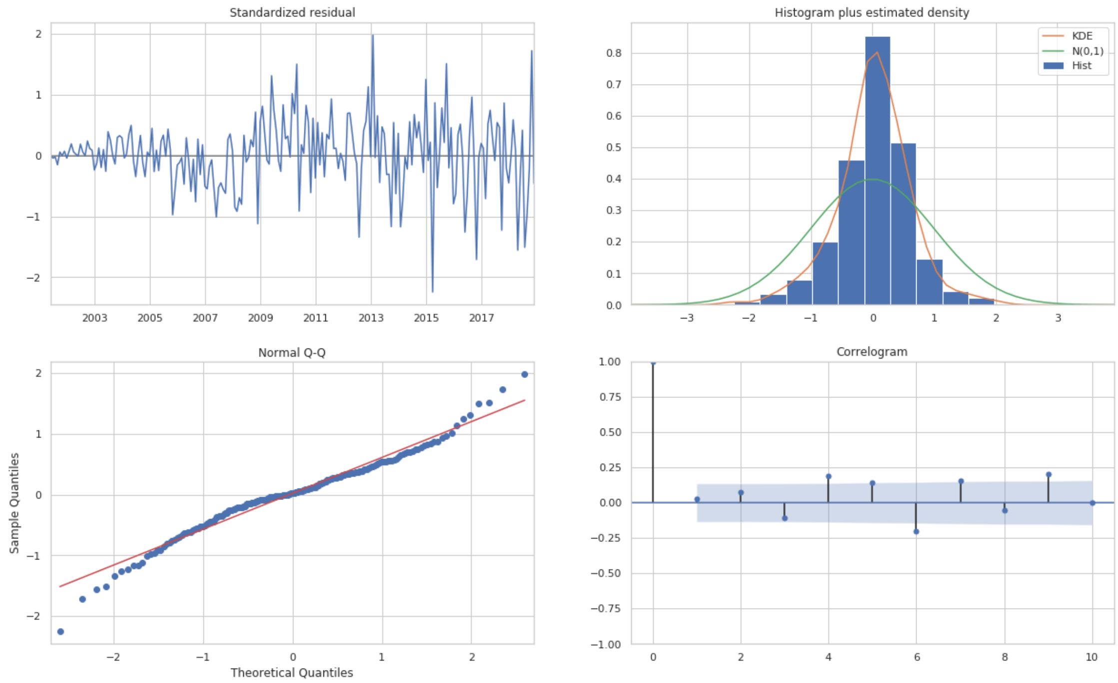
[57]:
```
# Transforming the los_angeles subset
los_angeles_simp = los_angeles[['date', 'value']]
los_angeles_simp = los_angeles_simp.set_index('date')

# Separate into training and test sets
```

```
sm_train = los_angeles_simp.loc[:'2018-12-01']
sm_test = los_angeles_simp.loc['2019-01-01':]
```

[ ]:
```
# Define the p, d and q parameters to take any value between 1 and 10
p = range(1, 5)
d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,␣
↪q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

[ ]:
```
# Run the model with the selected parameters
mod = sm.tsa.statespace.SARIMAX(sm_train,
                                order=(4, 1, 0),
                                seasonal_order=(4, 1, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])
```

[61]:
```
# Plot the model diagnostics.
results.plot_diagnostics(figsize=(20, 12))
plt.show()
```

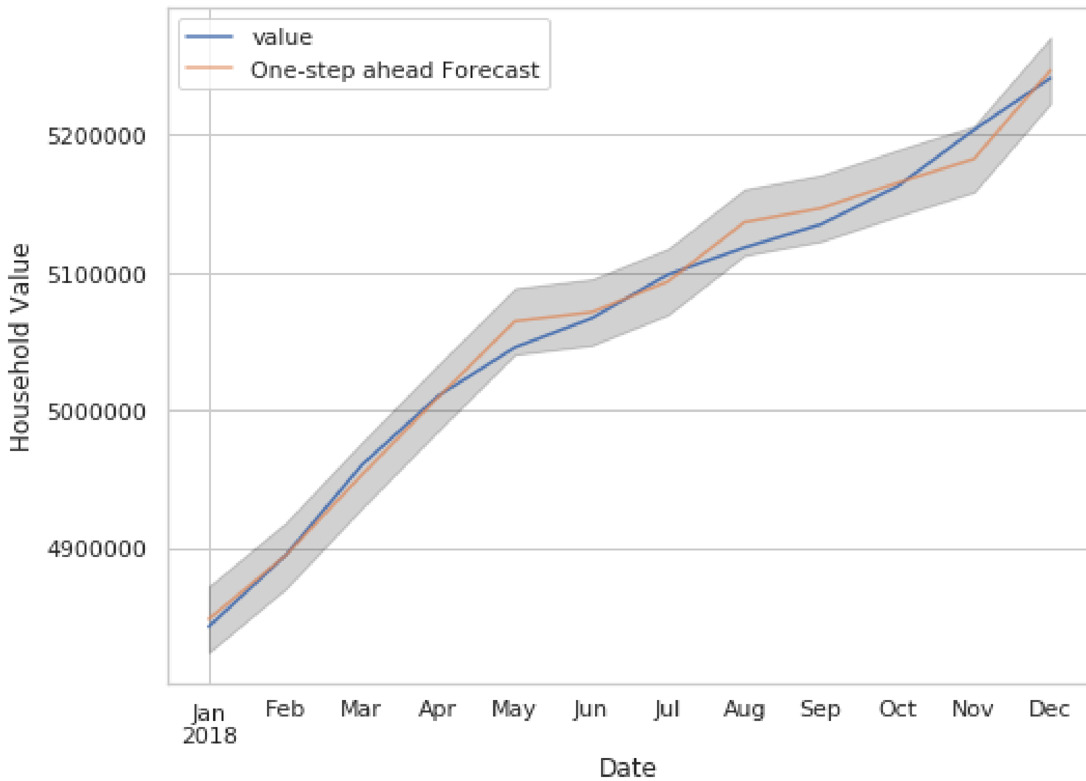


[62]:
```python
# Build a first validation using data from 2018 within the training set.

s1 = datetime.datetime.now()
print ("Start date and time:", s1.strftime("%Y-%m-%d %H:%M:%S"))

pred = results.get_prediction(start=pd.to_datetime('2018-01-01'), dynamic=False)
pred_ci = pred.conf_int()

s2 = datetime.datetime.now()
print ("End date and time:", s2.strftime("%Y-%m-%d %H:%M:%S"))
```

```
Start date and time: 2021-05-16 01:47:32
End date and time: 2021-05-16 01:47:32
```

[63]:
```python
# Overlay the time-series plot with the predicted validation data.
# We'll restrict showing the graphed data starting from 2000 to zoom into the␣
 ↪time-series
ax = sm_train['2018-01-01':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7,␣
 ↪figsize=(8,6))

# Shade confidence intervals
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
```

```
ax.set_xlabel('Date')
ax.set_ylabel('Household Value')
plt.legend()
plt.show()
```



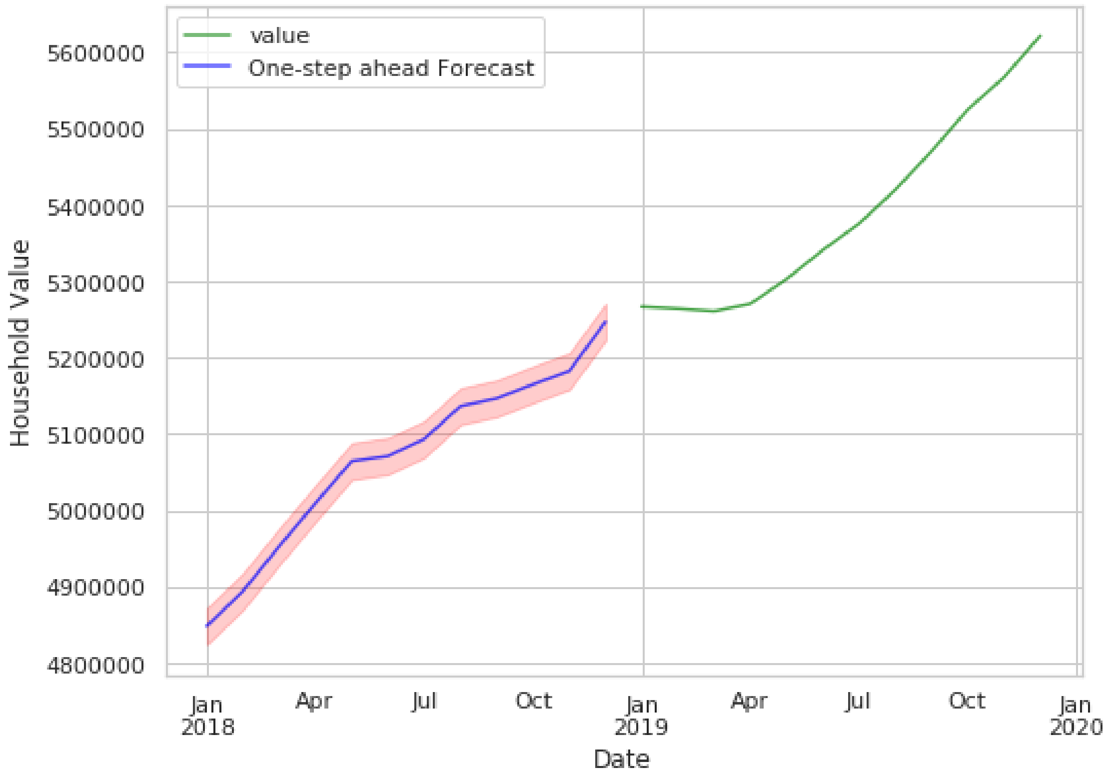
[64]:
```
# Obtain the predicted mean, merge with the actual values, and compute the MSE.
y_forecasted = pred.predicted_mean.reset_index()
y_truth = sm_train['2000-01-01':].reset_index()

y_forecasted.columns = ['date', 'value_pred']
y_merge = y_forecasted.merge(y_truth, how='inner', on='date')

# Compute the mean square error
mae = mean_absolute_error(y_merge.value, y_merge.value_pred)
mse = mean_squared_error(y_merge.value, y_merge.value_pred)
r2 = r2_score(y_merge.value, y_merge.value_pred)

# Print the metric results.
print('The Mean Absolute Error of our forecast is {}'.format(round(mae, 2)))
print('The Mean Squared Error of our forecast is {}'.format(round(mse, 2)))
```

```
print('The R-squared of our forecast is {}'.format(round(r2, 2)))
```

```
The Mean Absolute Error of our forecast is 8495.62
The Mean Squared Error of our forecast is 121380932.45
The R-squared of our forecast is 0.99
```

[ ]:
```
# Use the model to forecast fifteen months into 2018 and store the confidence␣
↪intervals

s1 = datetime.datetime.now()
print ("Start date and time:", s1.strftime("%Y-%m-%d %H:%M:%S"))

forecast = results.get_prediction(start=pd.to_datetime('2018-01-01'),end=pd.
↪to_datetime('2018-12-01'), dynamic=False)
forecast_ci = forecast.conf_int()

s2 = datetime.datetime.now()
print ("End date and time:", s2.strftime("%Y-%m-%d %H:%M:%S"))
```

[68]:
```
# Plot the actual results for 2018 vs the projected results.
ax = sm_test['2018-01-01':].plot(label='observed',color='green', alpha=.7)
forecast.predicted_mean.plot(ax=ax, label='One-step ahead␣
↪Forecast',color='blue', alpha=.7, figsize=(8,6))

ax.fill_between(forecast_ci.index,
                forecast_ci.iloc[:, 0],
                forecast_ci.iloc[:, 1], color='red', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Household Value')
plt.legend(loc='upper left')
plt.show()
```



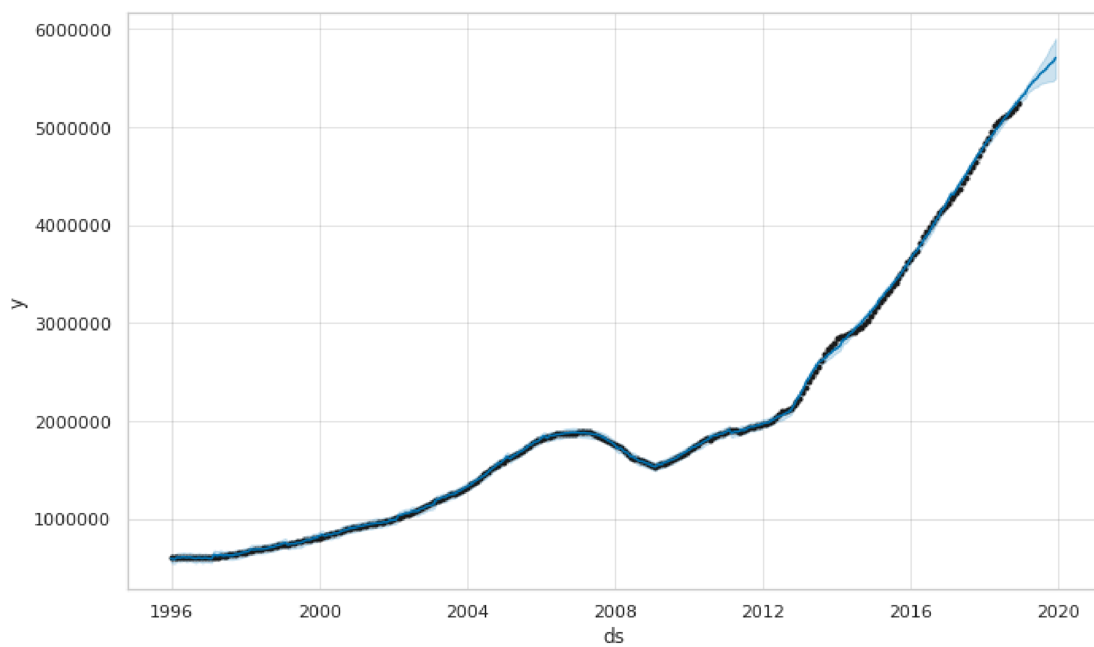
[71]:
```python
# Obtain the predicted mean, merge with the actual values, and compute the MSE.
y_forecasted = forecast.predicted_mean.reset_index()
y_test = sm_test.reset_index()

y_forecasted.columns = ['date', 'value_pred']

# Compute the mean square error
mae = mean_absolute_error(y_test.value, y_forecasted.value_pred)
mse = mean_squared_error(y_test.value, y_forecasted.value_pred)
r2 = r2_score(y_test.value, y_forecasted.value_pred)

# Print the results.
print('The Mean Absolute Error of our forecasts is {}'.format(round(mae, 2)))
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
print('The R-squared of our forecasts is {}'.format(round(r2, 2)))
```

```
The Mean Absolute Error of our forecasts is 322949.91
The Mean Squared Error of our forecasts is 107338698678.34
The R-squared of our forecasts is -6.04
```

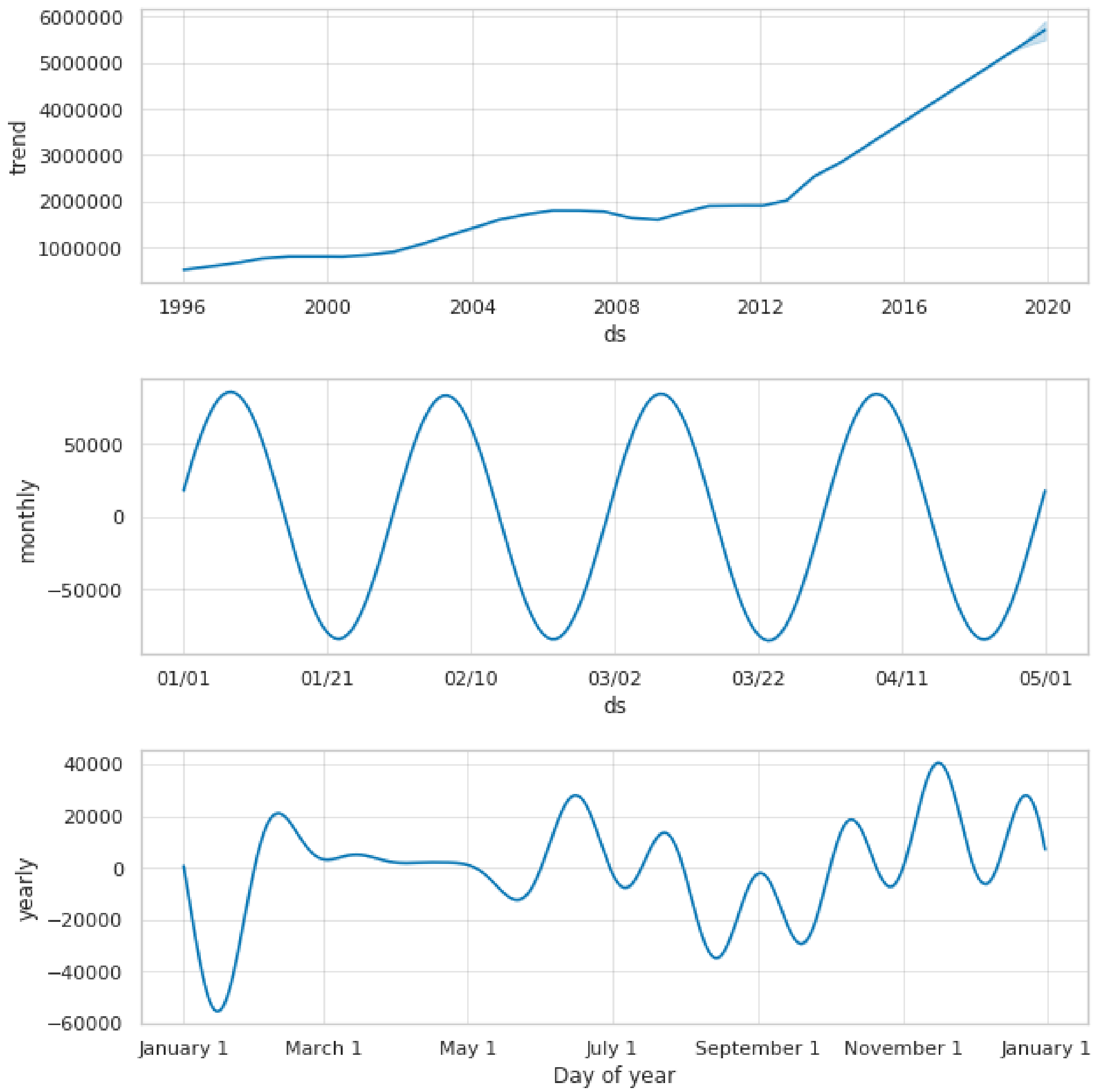
[ ]:
```
# Build prophet model with 95% confidence interval, looking at yearly␣
↪seasonality with prior changepontis
# set to 6 months and seasonality change point set to 1 month.
m1 = Prophet(interval_width=.95, changepoint_prior_scale=6,␣
↪yearly_seasonality=True,
             seasonality_prior_scale=1)
m1.add_seasonality(name='monthly', period=120, fourier_order=4)
sm_train2 = sm_train.reset_index()
sm_train2.columns = ['ds', 'y']
m1.fit(sm_train2)

# Look at the next 12 months.
future_dates = m1.make_future_dataframe(periods=12, freq='MS')
```

[ ]:
```
# Forecast next 12 months with confidence intervals
forecast = m1.predict(future_dates)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

[76]:
```
# Plot model with forecast trend values and uncertainty.
m1.plot(forecast, uncertainty=True);
```



[77]:
```
# Plot trend, yearly, and monthly seasonilities.
m1.plot_components(forecast);
```



[78]:
```
# Calculate MAE, MSE, and R-squared.
mae_prophet = mean_absolute_error(sm_test.value,␣
 ↪forecast[forecast['ds']>='2019-01-01'].yhat)
mse_prophet = mean_squared_error(sm_test.value,␣
 ↪forecast[forecast['ds']>='2019-01-01'].yhat)
r2_prophet = r2_score(sm_test.value, forecast[forecast['ds']>='2019-01-01'].
 ↪yhat)

# Print metrics
print('The Mean Absolute Error of our forecasts is {}'.
 ↪format(round(mae_prophet, 2)))
print('The Mean Squared Error of our forecasts is {}'.format(round(mse_prophet,␣
 ↪2)))
```

```
print('The R-squared of our forecasts is {}'.format(round(r2_prophet, 2)))
```

```
The Mean Absolute Error of our forecasts is 123593.17
The Mean Squared Error of our forecasts is 16877268088.49
The R-squared of our forecasts is -0.11
```

[79]:
```
# Calculate MAE, MSE, and R-squared with the lower CI.
mae_prophet = mean_absolute_error(sm_test.value,␣
↪forecast[forecast['ds']>='2019-01-01'].yhat_lower)
mse_prophet = mean_squared_error(sm_test.value,␣
↪forecast[forecast['ds']>='2019-01-01'].yhat_lower)
r2_prophet = r2_score(sm_test.value, forecast[forecast['ds']>='2019-01-01'].
↪yhat_lower)

# Print metrics
print('The Mean Absolute Error of our lower CI forecasts is {}'.
↪format(round(mae_prophet, 2)))
print('The Mean Squared Error of our lower CI forecasts is {}'.
↪format(round(mse_prophet, 2)))
print('The R-squared of our lower CI forecasts is {}'.format(round(r2_prophet,␣
↪2)))
```

```
The Mean Absolute Error of our lower CI forecasts is 65346.61
The Mean Squared Error of our lower CI forecasts is 5755875401.23
The R-squared of our lower CI forecasts is 0.62
```

0.9 Model Evaluation and Summary

---

[80]:
```
#select only few states
states_focused = ['CA', 'DC', 'NV', 'IN', 'AZ', 'FL']
df_data_pivot2 = df_data_pivot[df_data_pivot.State.isin(states_focused)]
```

[82]:
```
# Select train and test sets.
train = df_data_pivot2[(df_data_pivot2['date'] >= '2010-01-01') &␣
↪(df_data_pivot2['date'] <= '2018-12-01')]
test = df_data_pivot2[df_data_pivot2['date']>='2019-01-01']

# Keep date, value, and RegionName; rename to ds, y, and zipcode.
to_keep = ['date', 'value', 'State', 'RegionName']
train = train[to_keep]
train.columns = ['ds', 'y', 'State', 'zipcode']
test = test[to_keep]
test.columns = ['ds', 'y', 'State', 'zipcode']
```

[83]:
```
train_pivot = train.pivot(index='ds', columns='zipcode', values='y')
test_pivot = test.pivot(index='ds', columns='zipcode', values='y')
```

[84]:
```
# Build function that creates Prophet model
def prophet_train(data):
    """
    Creates a forecast based on the inputted zipcode.
    1. Filters the zipcode from the train set
    2. Builds the model
    3. Runs the model
    4. Creates a list of the next 12 months and produces a forecast for said␣
 ↪months
    5. Appends zipcode for identification
    """

    model = Prophet(interval_width=.95, changepoint_prior_scale=6,␣
 ↪yearly_seasonality=True,
              seasonality_prior_scale=1, weekly_seasonality=False,␣
 ↪daily_seasonality=False)
    model.add_seasonality(name='monthly', period=120, fourier_order=4)

    model.fit(data)

    future_dates = model.make_future_dataframe(periods=12, freq='MS')
    forecast = model.predict(future_dates)
    forecast = forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
    results = forecast[forecast['ds']>='2019-01-01']
    return(results)
```

[86]:
```
s1 = datetime.datetime.now()
print ("Start date and time:", s1.strftime("%Y-%m-%d %H:%M:%S"))

import prophet_train

starttime = time.time()
zipcodes = train.zipcode.unique()
forecasts = []
num = 1

for i in zipcodes:
    data = train[train['zipcode']==i]
    result = prophet_train.prophet_train(data)
    result = result[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
    result['zipcode'] = i
    forecasts.append(result)
    if len(zipcodes) % num == 0:
```

```
        print("{} percent completed".format(np.round(num/len(zipcodes) * 100),␣
 ↪2))
    num += 1

print("...runtime: {:.2f} minutes".format((time.time() - starttime)/60.0))

s2 = datetime.datetime.now()
print ("End date and time:", s2.strftime("%Y-%m-%d %H:%M:%S"))
```

```
Start date and time: 2021-05-16 01:48:03
0.0 percent completed
0.0 percent completed
0.0 percent completed
0.0 percent completed
1.0 percent completed
1.0 percent completed
1.0 percent completed
2.0 percent completed
2.0 percent completed
4.0 percent completed
5.0 percent completed
11.0 percent completed
14.0 percent completed
33.0 percent completed
100.0 percent completed
…runtime: 178.37 minutes
End date and time: 2021-05-16 04:46:25
```

[87]:
```
### Converts the results from the previous block to csv. This csv will be␣
 ↪imported in the next block, while this block is commented out.
## Initiate empty data frame
zipcodes2 = train.zipcode.unique()

train_results = pd.DataFrame()

## Iterate over the zipcodes and append them to the corresponding forecasts.
for i in range(0, len(zipcodes2)):
    result = forecasts[i]
    result['zipcode'] = zipcodes2[i]
    train_results = pd.concat([train_results, result], axis=0,␣
 ↪ignore_index=True)

# Convert train_results to dataframe and export to csv
train_results = pd.DataFrame(train_results)
train_results.to_csv('train_results.csv', index=False)
```

[ ]:
```python
# Load csv with forecasted results
train_results = pd.read_csv('train_results.csv', index_col=False)

# Convert ds to datetime and zipcode to object to merge with the test results.
train_results['ds'] = pd.to_datetime(train_results['ds'])
train_results['zipcode'] = train_results.zipcode.astype(object)
# train_results.head()
```

[89]:
```python
# Merge forecasts with test set.
forecast_results = test.merge(train_results, on=['ds', 'zipcode'], how='left').
↪drop_duplicates()

# Obtain unique zipcodes to filter and obtain MAE and R-squared for each␣
↪forecast
zc = forecast_results.zipcode.unique()

# Initiate empty array to store results
evaluation_results = pd.DataFrame()

# Iterate over every zipcode, calculate MAE, R-Squared, and percentage change␣
↪in forecasted value
for i in zc:
    data = forecast_results[forecast_results['zipcode']==i].reset_index()
    mae = mean_absolute_error(data.y, data.yhat)
    rsq = r2_score(data.y, data.yhat)
    first_value = data.iloc[0, 5]
    last_value = data.iloc[-1, 5]
    pct_change = np.round(((last_value/first_value) - 1) * 100, 2)
    data_evaluation = pd.DataFrame({'zipcode':i, 'MAE':mae, 'R2':rsq,␣
↪'Pct_change':pct_change}, index=[0])
    evaluation_results = pd.concat([evaluation_results, data_evaluation],␣
↪axis=0, ignore_index=True)
```

[90]:
```python
# Print first rows of evaluation_results to check that the data was calculated␣
↪correctly
evaluation_results.head()
```

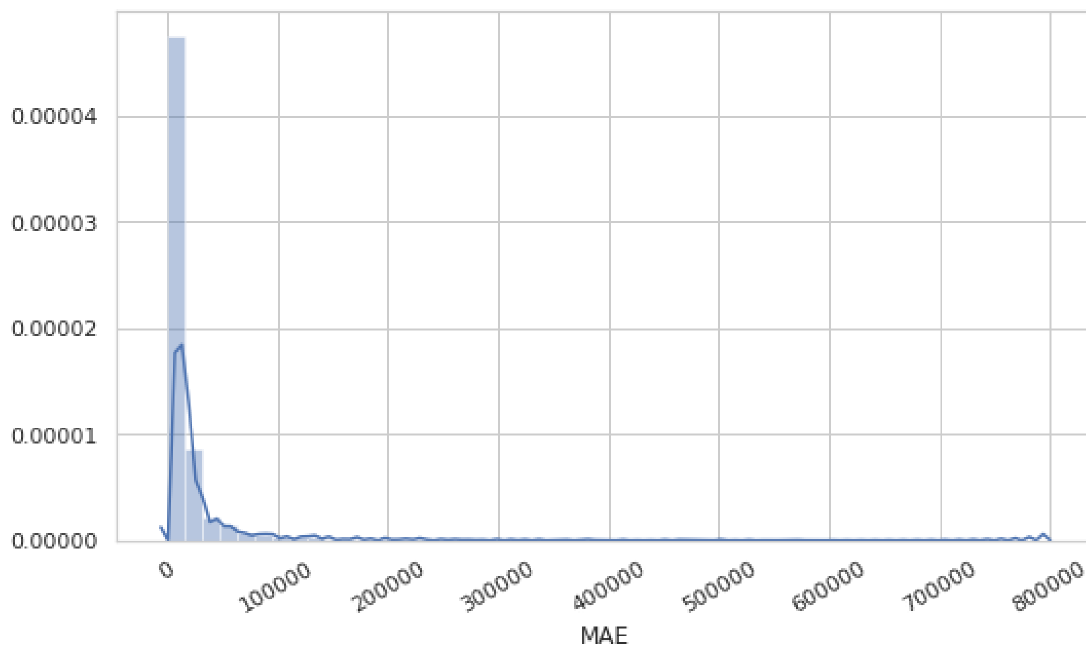
[90]:

	MAE	Pct_change	R2	zipcode
0	3167.603278	-4.47	-34.500037	4956
1	8343.748634	-2.09	-0.979198	7961
2	16094.093448	6.39	-7.996225	20001
3	21969.840190	7.28	-64.628050	20002
4	13713.333741	5.00	-7.595141	20003

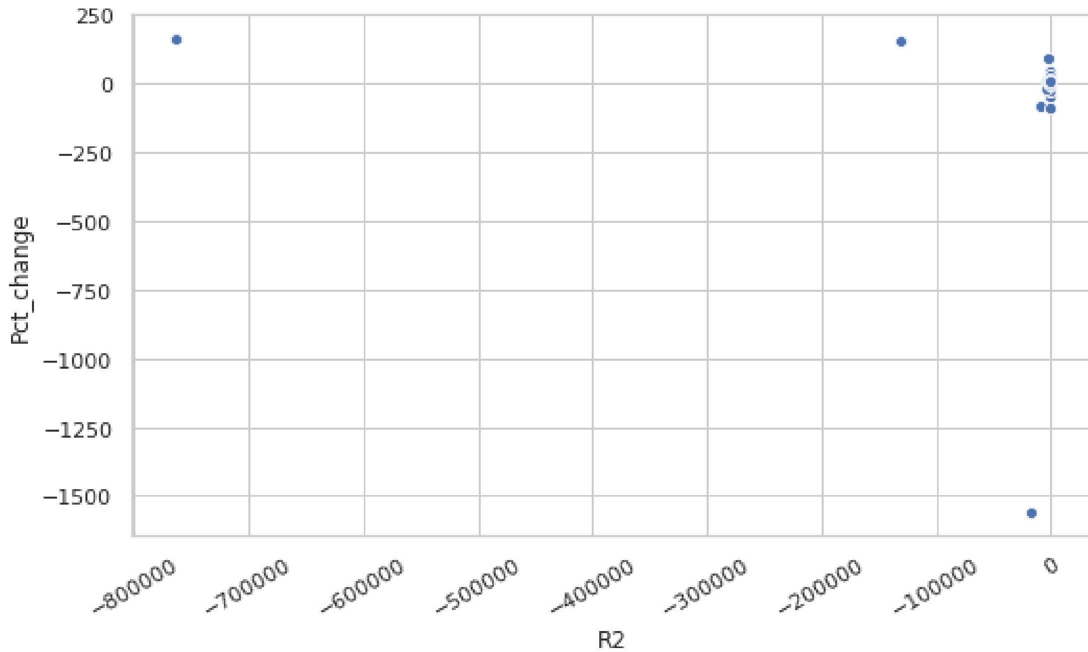
[125]:
```python
# Enlarge the plot

plt.figure(figsize=(9,5))
```

```
sns.distplot(evaluation_results.MAE)
# Rotate x-labels
plt.xticks(rotation=30)
plt.show()
```

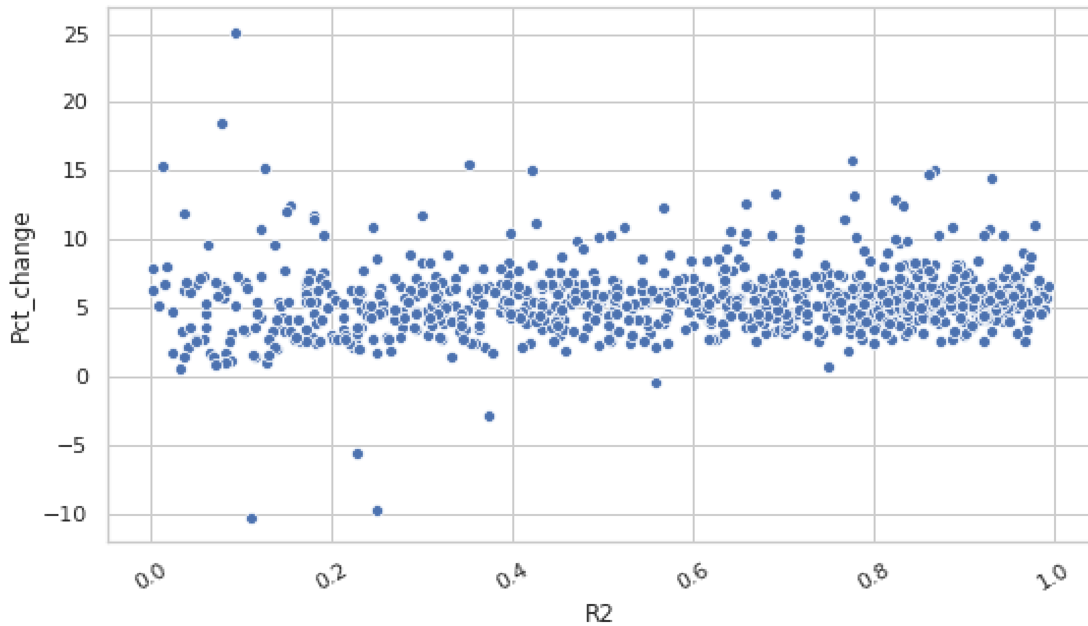
```
/usr/local/envs/py3env/lib/python3.5/site-
packages/statsmodels/nonparametric/kde.py:475: DeprecationWarning: object of
type <class 'numpy.float64'> cannot be safely interpreted as an integer.
  grid,delta = np.linspace(a,b,gridsize,retstep=True)
```



[126]:
```
# Enlarge the plot

plt.figure(figsize=(9,5))
sns.scatterplot(x='R2', y='Pct_change', data=evaluation_results)

# Rotate x-labels
plt.xticks(rotation=30)
plt.show()
```



[127]:
```python
# Limit the data to R-squared greater than or equal to zero
limit_results = evaluation_results[evaluation_results['R2']>=0]

# Enlarge the plot
plt.figure(figsize=(9,5))
sns.scatterplot(x='R2', y='Pct_change', data=limit_results)
# Rotate x-labels
plt.xticks(rotation=30)
plt.show()

# How many models are we losing?
print("There are {} zipcodes in the original data".
 ↪format(len(evaluation_results)))
print("Now we're limiting to {} zipcodes".format(len(limit_results)))
```




```
There are 3969 zipcodes in the original data
Now we're limiting to 943 zipcodes
```

[94]:
```
# Calculate high risk - low reward and low risk - high reward
hr_lr = limit_results.groupby(['R2', 'Pct_change'])[['zipcode', 'R2',␣
↪'Pct_change']].min().iloc[0]
lr_hr = limit_results.groupby(['R2', 'Pct_change'])[['zipcode', 'R2',␣
↪'Pct_change']].max().iloc[-1]
print(hr_lr)
print(lr_hr)
```

```
zipcode       86332.000000
R2                0.001461
Pct_change        6.270000
Name: (0.001460612368360592, 6.27), dtype: float64
zipcode       47971.00000
R2                0.99301
Pct_change        6.62000
Name: (0.9930101854267513, 6.62), dtype: float64
```

[ ]:
```
# Calculate high risk - high reward and low risk - low reward
hr_hr = limit_results[limit_results['Pct_change']>=0].groupby('R2')['zipcode',␣
↪'Pct_change'].min().sort_values('Pct_change')
lr_lr = limit_results[limit_results['Pct_change']>=0].groupby('R2')['zipcode',␣
↪'Pct_change'].max().sort_values('Pct_change', ascending=False)
print(hr_hr.head(20))
```

```python
print(lr_lr.head())
```

[ ]:
```python
# Obtain the four markets by risk-reward
zipcodes_final = [86332, 47971,85172,96034]
cities = df_data_pivot[df_data_pivot.RegionName.
 ↪isin(zipcodes_final)][['RegionName', 'City', 'State', 'Metro',␣
 ↪'CountyName']].drop_duplicates()
# cities
```

[105]:
```python
# Obtain the top three best investment markets zipcodes
best_three = limit_results.groupby(['R2', 'Pct_change'])[['zipcode']].max().
 ↪reset_index()
best_three = best_three.sort_values(['R2', 'Pct_change'], ascending=False)
best_three_array = np.asarray(best_three.iloc[0:3, 2])

# Filter the zipcodes from df_data_pivot
best_cities = df_data_pivot[df_data_pivot.RegionName.
 ↪isin(best_three_array)][['RegionName', 'City', 'State', 'Metro',␣
 ↪'CountyName']].drop_duplicates()
best_cities.merge(best_three, left_on='RegionName', right_on='zipcode',␣
 ↪how='left').sort_values('Pct_change', ascending=False)
```

[105]:
```
  RegionName              City State                     Metro  \
2      47971            Oxford    IN  Lafayette-West Lafayette
0      46065         Rossville    IN                 Frankfort
1      46962  North Manchester    IN                    Wabash

       CountyName        R2  Pct_change  zipcode
2   Benton County  0.993010        6.62    47971
0  Clinton County  0.991205        5.78    46065
1   Wabash County  0.990880        5.77    46962
```

0.10 Summary and Conclusion

- OMG! - Yes, thats how I feel now. It started as a simple exercise with less data sources.However, the complexity was in the form of having 300+ temporal data at city, state, county, metro and zipcode level on all 50 US States.
- After relatively simpe Observe, Scrub processes; exploratotry involved slicing data by metro level, state level, percentage of increase over time. There was a time between 2008 to 2012 the market was real down (recession) - the housing was seen sharp fall in price. However, from 2009 onwards the overall pricing has been on a steady rise. Intrestingly, Indiana, Florida and Arizona had best runs considering the growth percentage from 1996. We have alos looked at Florida to understand how the pricing has been over period of time. Comparing the real estate in 1996 vs 2019; Wauchulla, Lake Ciry and Miami-Fort Laurerdale-West Palm Beach had seen higher percentage of growth (>200%).

- Modeling - Its more of tug of war on the compute resource vs modeling. I almost ran into 3 differnt environments, Jupyter Notebook on Local Mac book (~16GB RAM), IBM Watson Studio and finally GCP Datalab isntance with 64vCPUs,416 GB memory. On the modeling, It stated with understanding the autocorrelation to find the best lag for the model.Based on the initial model ,ran the prediction on the training data set using SARIMAX algortithm; Model has returned and R-Squared of 0.99, MAE as 8495.62 and Mean Squared Error as 121380932. Yes, its still using the training set - so, cant be excited about the R^2 value.
- Then, built a prophet model with 95% confidence interval with monthly & yearly seasonality. This model had R^2 as -0.11 showing strong relationship between actual and predicted values but with the value being negative, we can consider this as well.MAE and Mean Squared Error having high values as well.
- Next prophet model was run with > 2019 values with lower Confidence interval, R^2 value was at 0.62. Comparatively better than the earlier model.
- Now, its time to run the prophet model for the entire dataset,It took about 3 hours to complete with 'CA', 'DC', 'NV', 'IN', 'AZ', 'FL' states in selection.In addition, the process was run with different scenarios based on the R^2, Percent Change by Zipcode across 2010 to 2020.
  - High Risk - High Reward
  - Low Risk - Low Reward
  - Low Risk - High Reward
  - High Risk - Low Reward
- Based on various scenarios, below are the recommendations for Syracuse Real Estate Investment Trust to invest. These may look high reward investements for SREIT.

  Low Risk - High Reward
  - Oxford, IN from Benton County around Lafayette-West Lafayette Metro with R^2 0.99 and Percent Change 6.62
  - Rossville, IN from Clinton County around Frankfort Metro with R^2 0.99 and Percent Change 5.78 |
  - North Manchester,IN from Wabash County around Wabash Metro with R^2 0.99 and Percent Change 5.71

0.11 Next Steps

- This process was run on a GCP Datalab instance with Machine type n1-highmem-64 (64 vCPUs, 416 GB memory). However, It was taking more than 6 hours to complete for 8 states combined.Often with session issues, the code had to be executed mutilple times. Eventually, I had decided to reduce the number of states and duration starting 2010 to run the final prediction. This itself took ~180 minutes. So, for future work, I would try to run this on a GPU instance with higher compute. Definitly, with higher compute, will be able to include all the states from 1996 to run the predictions.
- I have also learned about Interactive choropleath maps using geoplot, geopandas libraries and geoJSON and Shape files. However, this would be integrated in our final project.

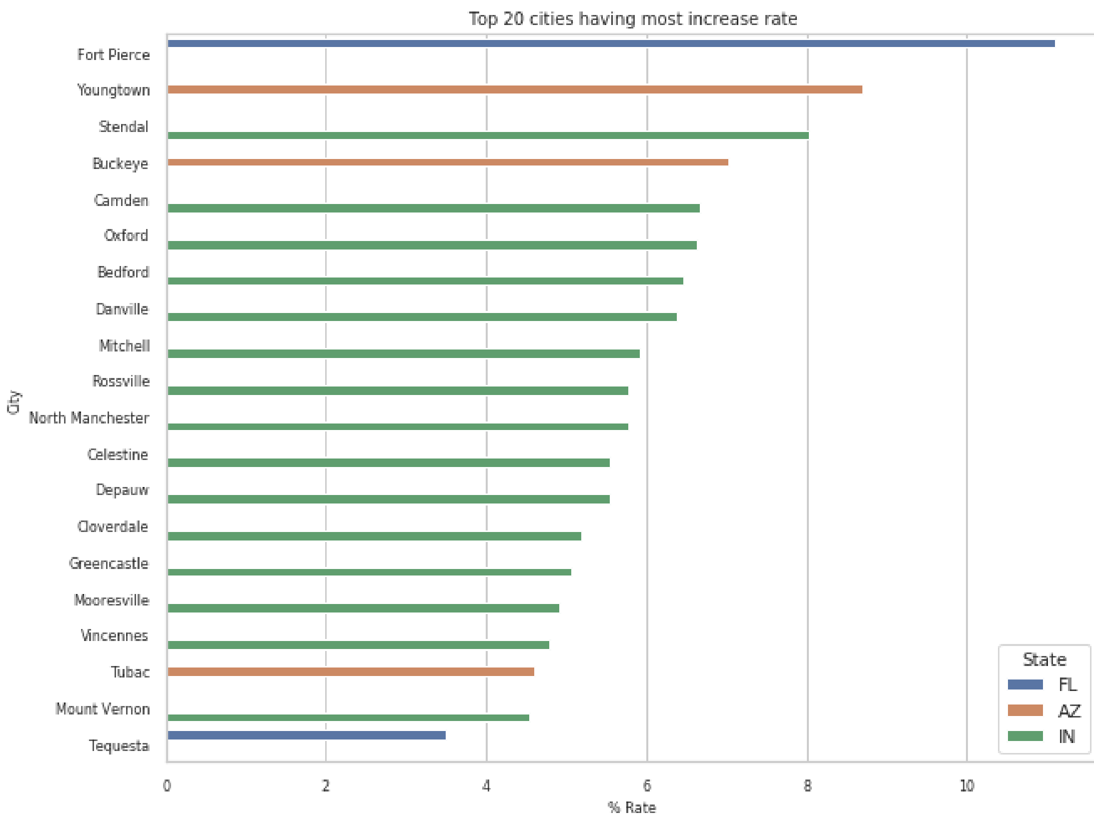
Appendix

[107]:
```
# Filter the zipcodes from df_data_pivot

top_20_array = np.asarray(best_three.iloc[0:20, 2])

best_20_cities = df_data_pivot[df_data_pivot.RegionName.
 ↪isin(top_20_array)][['RegionName', 'City', 'State', 'Metro', 'CountyName']].
 ↪drop_duplicates()
best_20_cities = best_20_cities.merge(best_three, left_on='RegionName',␣
 ↪right_on='zipcode', how='left').sort_values('Pct_change', ascending=False)
```

[ ]:
```
best_20_cities.reset_index(drop=True, inplace=True)
best_20_cities = best_20_cities[['City', 'State', 'Metro','zipcode',␣
 ↪'CountyName','R2','Pct_change']]
# best_20_cities
```

[128]:
```
plt.figure(figsize= (10,8))
plt.xticks(fontsize = 8)
plt.yticks(fontsize = 8)
plt.xlabel("Total cases",fontsize = 8)
plt.ylabel('City',fontsize = 8)
plt.title("Top 20 cities having most increase rate" , fontsize = 10)
ax = sns.barplot(x = best_20_cities['Pct_change'], y = best_20_cities.City,hue␣
 ↪= best_20_cities.State)
ax.set(xlabel='% Rate', ylabel='City')
```

[128]:
```
[Text(0,0.5,'City'), Text(0.5,0,'% Rate')]
```

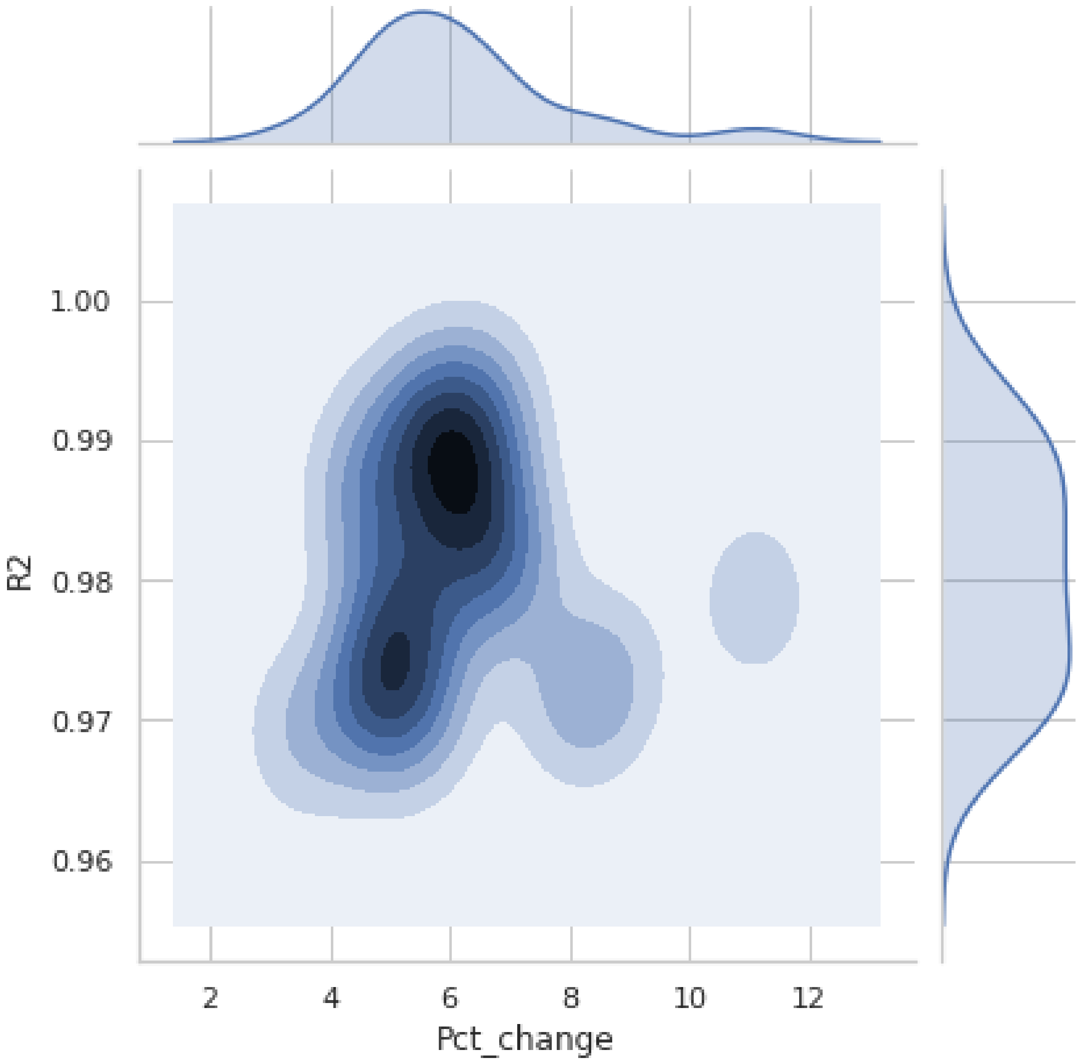


[129]:
```
# Enlarge the plot
plt.figure(figsize=(9,4))

sns.jointplot(
    data=best_20_cities,
    x=best_20_cities['Pct_change'], y=best_20_cities['R2'], hue=best_20_cities.
 ↪State,
    kind="kde"
)
```

```
/usr/local/envs/py3env/lib/python3.5/site-
packages/statsmodels/nonparametric/kde.py:475: DeprecationWarning: object of
type <class 'numpy.float64'> cannot be safely interpreted as an integer.
  grid,delta = np.linspace(a,b,gridsize,retstep=True)
```

[129]:
```
<seaborn.axisgrid.JointGrid at 0x7f7d943b5208>
```

```
<matplotlib.figure.Figure at 0x7f7d943afe80>
```



0.12 References

---

1. https://www.fortunebuilders.com/syracuse-ny-real-estate-market-trends-2016/
2. https://ingeh.medium.com/markdown-for-jupyter-notebooks-cheatsheet-386c05aeebed
3. http://files.zillowstatic.com/research/public/Zip/Zip_Zhvi_SingleFamilyResidence.csv
4. https://www.dol.gov/owcp/regs/feeschedule/fee/fee11/fs11_gpci_by_msa-zip.xls
5. https://stackoverflow.com/questions/60145006/cannot-import-name-easter-from-holidayshttps://stackoverflow.com/questions/60145006/cannot-import-name-easter-from-holidays
6. https://anaconda.org/conda-forge/fbprophet
7. https://www.rate.com/research/fort_pierce-fl-34950/market-trends
8. https://www.rate.com/research/youngtown-az-85363

9. https://www.rate.com/research/oxford-in-47971
10. https://towardsdatascience.com/14-data-visualization-plots-of-seaborn-14a7bdd16cd7
11. https://www.python-graph-gallery.com/choropleth-map-geopandas-python