IST 718

LAB 2 ASSIGNMENT

Matthew L. Pergolski

Professor Jillian Lando

Overview

In the IST 718 Lab 2 assignment, the O-S-E-M-IN method will be conducted to recommend the best investment areas (by zip code) for the Syracuse Real Estate Investment Trust (SREIT). In order to provide the analysis, a brief explanation of the data science method will be required.

- O
    - Obtain: In the obtaining section, Data Acquisition will be discussed and referenced.
- S
    - Scrub: In the scrubbing section, Data Cleaning will be discussed and referenced.
- E
    - Explore: In the exploring section, Data Exploration will be discussed and referenced.
- M
    - Model: In the modeling section, Data Modeling techniques will be discussed – the workings of our linear model will be introduced and referenced.
- IN
    - Interpret: In the interpreting section, we will summarize the results and provide the overall recommendation to the stakeholder.

To achieve the recommendation, a modeling technique of the SARIMAX model as well as the Facebook Prophet model will be conducted on the dataset. The SARIMAX model will be dedicated for the filtered data based on the locations described in the assignment instructions (Hot Springs, Little Rock, Fayetteville, Searcy) while the prophet model will be used for predicting each zip code in the dataset.

Data Acquisition

The dataset used in the analysis comes from Zillow:

- Zillow
    - http://files.zillowstatic.com/research/public_csvs/zhvi/Zip_zhvi_uc_sfr_month.csv

The dataset was loaded with the pandas library directly from the web utilizing the 'to_csv' method.

Data Cleaning

In the 'Data Cleaning' phase, each dataset was inspected and transformed as needed.

Zillow

In looking at the 'Zillow' dataset, the .info() method was invoked and provided the following results:

```
# Load    Follow link (cmd + click)
url = "http://files.zillowstatic.com/research/public_csvs/zhvi/Zip_zhvi_uc_sfr_month.csv"
df = pd.read_csv(url)

# Understand the data
df.info(verbose=True, null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29532 entries, 0 to 29531
Data columns (total 336 columns):
 #    Column      Non-Null Count  Dtype
---   ------      --------------  -----
 0    RegionID    29532 non-null  int64
 1    SizeRank    29532 non-null  int64
 2    RegionName  29532 non-null  int64
 3    RegionType  29532 non-null  object
 4    StateName   29532 non-null  object
 5    State       29532 non-null  object
 6    City        28280 non-null  object
 7    Metro       22368 non-null  object
 8    CountyName  29532 non-null  object
 9    1996-02-29  14361 non-null  float64
 10   1996-03-31  14443 non-null  float64
 11   1996-04-30  14457 non-null  float64
 12   1996-05-31  14474 non-null  float64
 13   1996-06-30  14539 non-null  float64
 14   1996-07-31  14558 non-null  float64
 15   1996-08-31  14573 non-null  float64
 16   1996-09-30  14600 non-null  float64
 17   1996-10-31  14623 non-null  float64
 18   1996-11-30  14636 non-null  float64
 19   1996-12-31  14655 non-null  float64
 20   1997-01-31  14678 non-null  float64
 21   1997-02-28  14836 non-null  float64
```

This method allowed me to inspect datatypes as well as any null-values; it also provides an overview of the shape of the data frame – calling out the number of rows and columns, respectively.

After initial inspection, it appears as though there are many nulls in the dataset. It also appears that each date is constructed as a column. This will make analyzing the data within our models difficult, so a transformation will be needed. We will ultimately need to 'normalize' the dates and condense them to one single column, called 'Date.' This feature engineering aspect will be required to fit the data within the SARIMAX and prophet models in the following sections.

Another required step in our cleansing process will be to resolve the null values. There are many possible ways to do this; however, our approach will consist of combining two approaches.

For any column that is corresponds to a date, we will use the 'interpolate' method of resolving null values. Interpolation can be used to estimate missing values in a dataset based on the known values before and after the missing value – which will work favorably for our time series dataset.

Once nulls for dates are interpolated, we'll inspect the nulls for the non-date columns. Since these would be increasingly more difficult to interpolate or estimate, we will simply drop these remaining null values.

<u>Null-Handling:</u>

```
'''
Interpolate the missing values in the dataframe for date columns


Remove nulls from Region, Size, Rank, RegionName, RegionType

'''

interp_re_df = pd.concat([df.iloc[:, :9], df.iloc[:, 9:].interpolate()], axis=1)
print(interp_re_df.shape)
interp_re_df.head()

interp_re_df_nonull = interp_re_df.dropna()
print(interp_re_df_nonull.shape)
interp_re_df_nonull.info(verbose=True, null_counts=True)
```

```
(29532, 336)
(21639, 336)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21639 entries, 0 to 29527
Data columns (total 336 columns):
 #    Column      Non-Null Count  Dtype
---   ------      --------------  -----
 0    RegionID    21639 non-null  int64
 1    SizeRank    21639 non-null  int64
 2    RegionName  21639 non-null  int64
 3    RegionType  21639 non-null  object
 4    StateName   21639 non-null  object
 5    State       21639 non-null  object
 6    City        21639 non-null  object
 7    Metro       21639 non-null  object
 8    CountyName  21639 non-null  object
 9    1996-02-29  21639 non-null  float64
 10   1996-03-31  21639 non-null  float64
 11   1996-04-30  21639 non-null  float64
 12   1996-05-31  21639 non-null  float64
 13   1996-06-30  21639 non-null  float64
 14   1996-07-31  21639 non-null  float64
 15   1996-08-31  21639 non-null  float64
 16   1996-09-30  21639 non-null  float64
 17   1996-10-31  21639 non-null  float64
 18   1996-11-30  21639 non-null  float64
 19   1996-12-31  21639 non-null  float64
```

<u>'Normalization' of Date Columns</u>

```python
# Convert wide format to long format for easier time series analysis
date_columns = [col for col in interp_re_df_nonull.columns if col.startswith('19') or col.startswith('20')]
columns_to_keep = ['RegionName', 'Metro'] + date_columns
interp_re_df_nonull_long = interp_re_df_nonull[columns_to_keep]

df_all_states_long = pd.melt(interp_re_df_nonull_long, id_vars=['RegionName', 'Metro'], var_name='Date', value_name='Value')
df_all_states_long['Date'] = pd.to_datetime(df_all_states_long['Date'], format="%Y-%m")
df_all_states_long = df_all_states_long.sort_values(by=['RegionName', 'Date'])
print(df_all_states_long.shape)
df_all_states_long.head(30)
```

```
(7075953, 4)
```

	RegionName	Metro	Date	Value
7135	1001	Springfield, MA	1996-02-29	123910.992045
28774	1001	Springfield, MA	1996-03-31	125042.323470
50413	1001	Springfield, MA	1996-04-30	125614.049671
72052	1001	Springfield, MA	1996-05-31	125169.953668
93691	1001	Springfield, MA	1996-06-30	126749.059418
115330	1001	Springfield, MA	1996-07-31	128172.831065
136969	1001	Springfield, MA	1996-08-31	128493.068781
158608	1001	Springfield, MA	1996-09-30	128056.102961
180247	1001	Springfield, MA	1996-10-31	128163.832468
201886	1001	Springfield, MA	1996-11-30	128126.012404
223525	1001	Springfield, MA	1996-12-31	127273.502536
245164	1001	Springfield, MA	1997-01-31	127142.463047
266803	1001	Springfield, MA	1997-02-28	126407.858947
288442	1001	Springfield, MA	1997-03-31	125891.886899
310081	1001	Springfield, MA	1997-04-30	126430.977134

As a result, the data set contains no records with null values and a condensed 'Date' column. The shape of the data frame just over 7 million rows, which will be enough data to explore modeling techniques with.

Data Exploration

In the data exploration phase, the assignment instructions suggest to explore the following areas:

- Hot Springs
- Little Rock
- Fayetteville
- Searcy

We begin by filtering the current data frame by the selected metros of interest, as per the instructions:

```python
selected_metros = ['Hot Springs', 'Little Rock', 'Fayetteville', 'Searcy']
df_filtered = interp_re_df_nonull[interp_re_df_nonull['City'].isin(selected_metros)]
print(df_filtered.shape)
df_filtered.head()
```

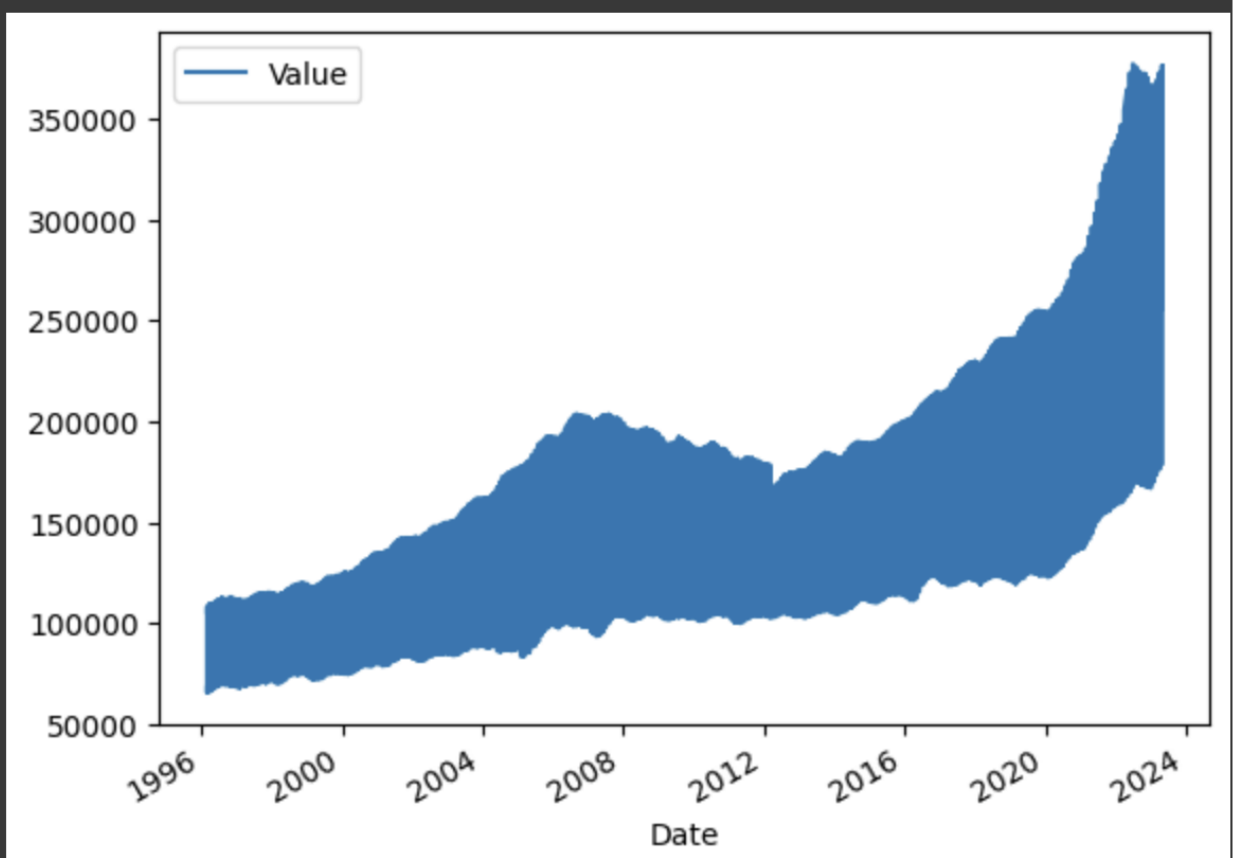
(20, 336)

	RegionID	SizeRank	RegionName	RegionType	StateName	State	City	Metro
1051	89707	1060	72701	zip	AR	AR	Fayetteville	Fayetteville-Springdale-Rogers, AR
1177	89249	1189	71913	zip	AR	AR	Hot Springs	Hot Springs, AR

Before diving into each metro, we can look at all metros combined first:

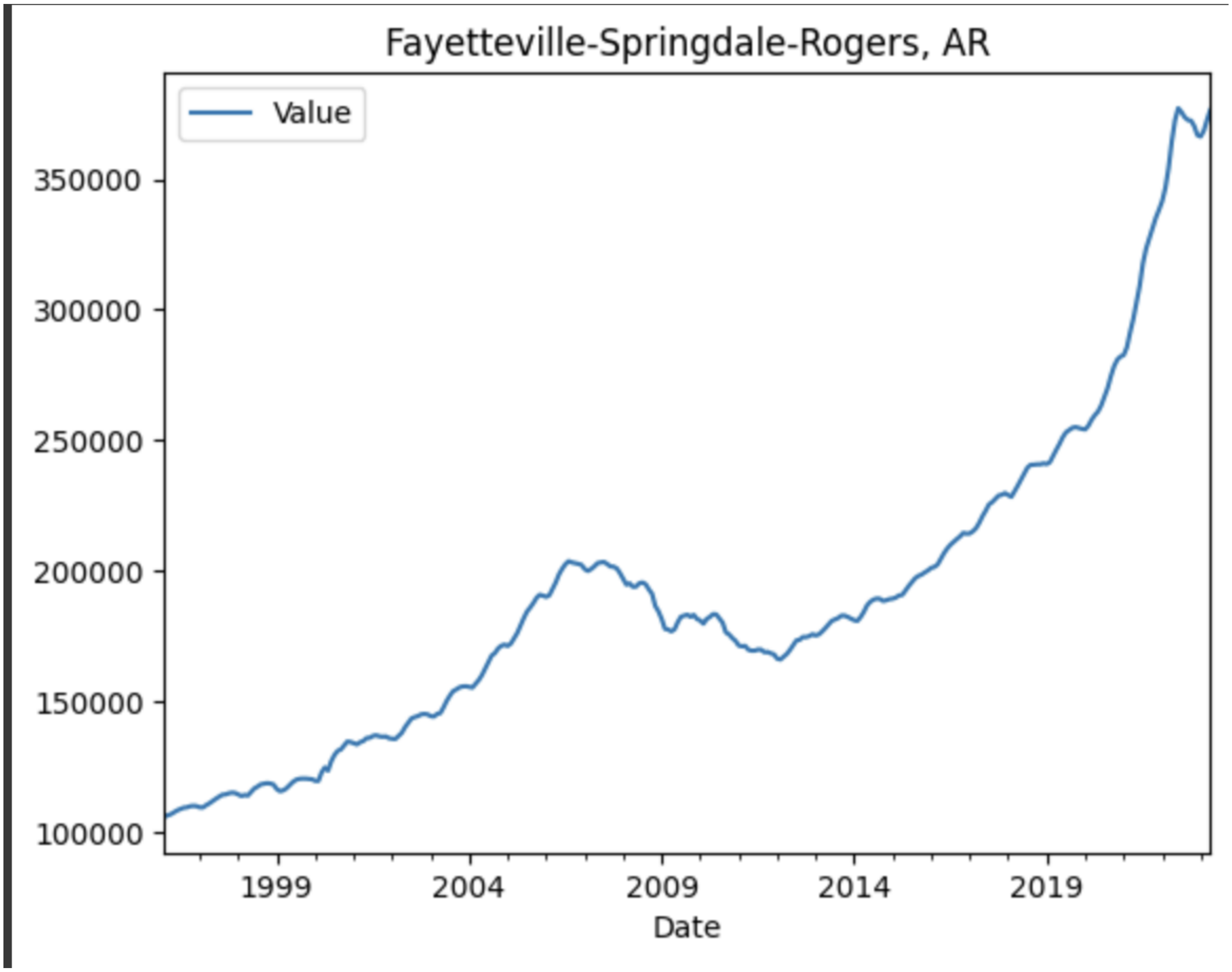
```python
df_long = df_long.groupby(['Metro', 'Date']).mean().reset_index()

df_long.plot(x='Date', y='Value')
plt.show()
```

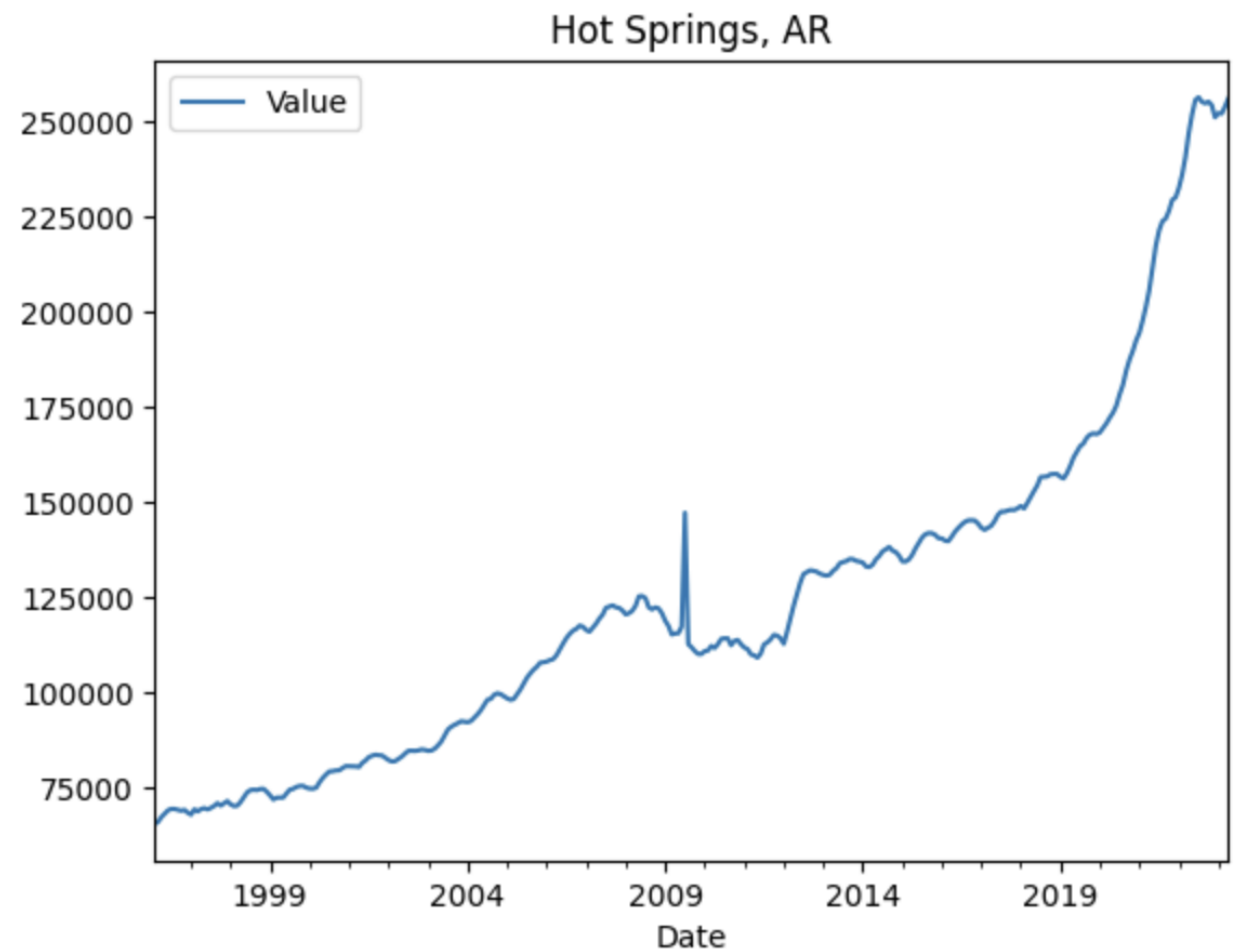


This initial plot gives the audience an idea of what to expect next with each respective metro area. A general trend upward, with seasonality components as well as volatility in the market from 1996 to 2024.

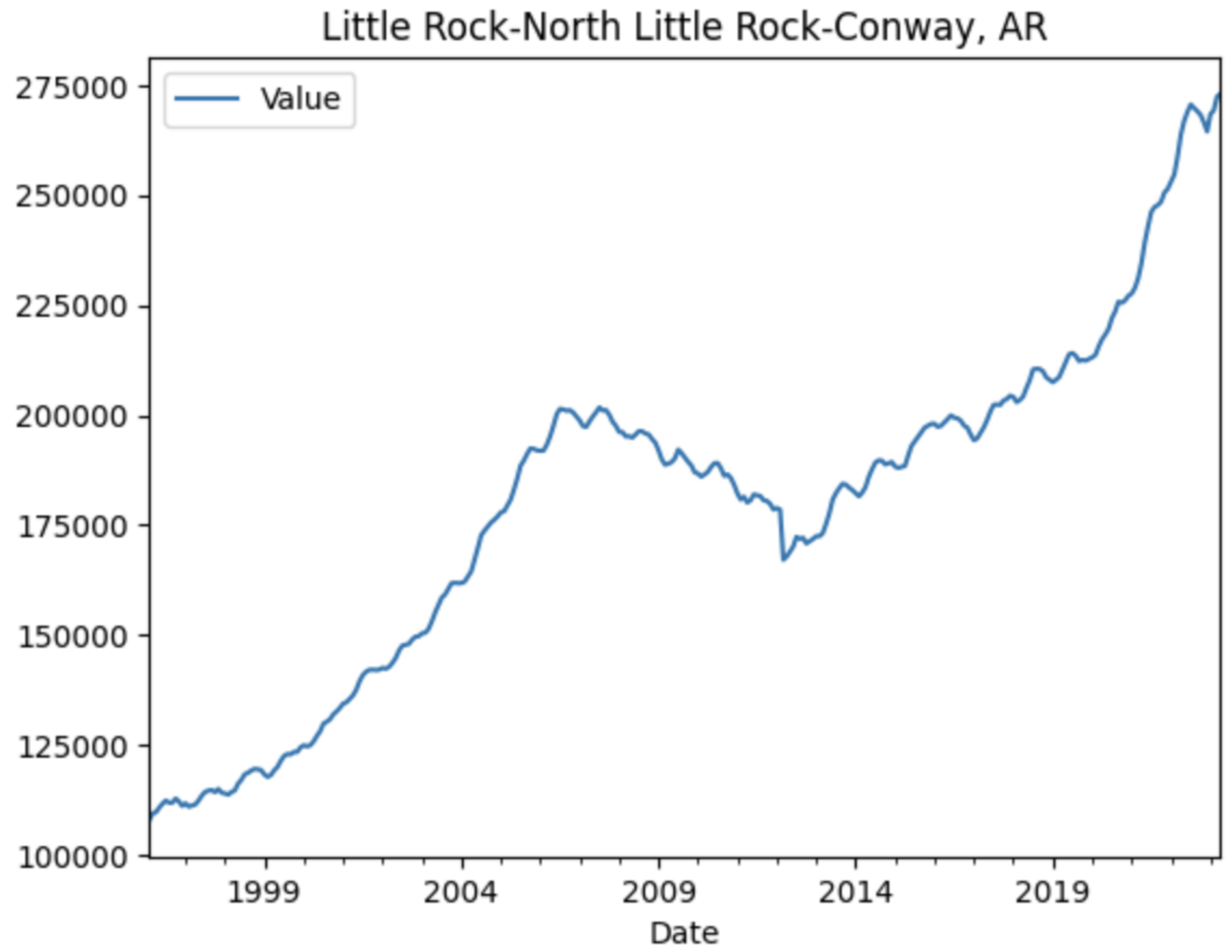
With this in mind, we can begin exploring each metro:



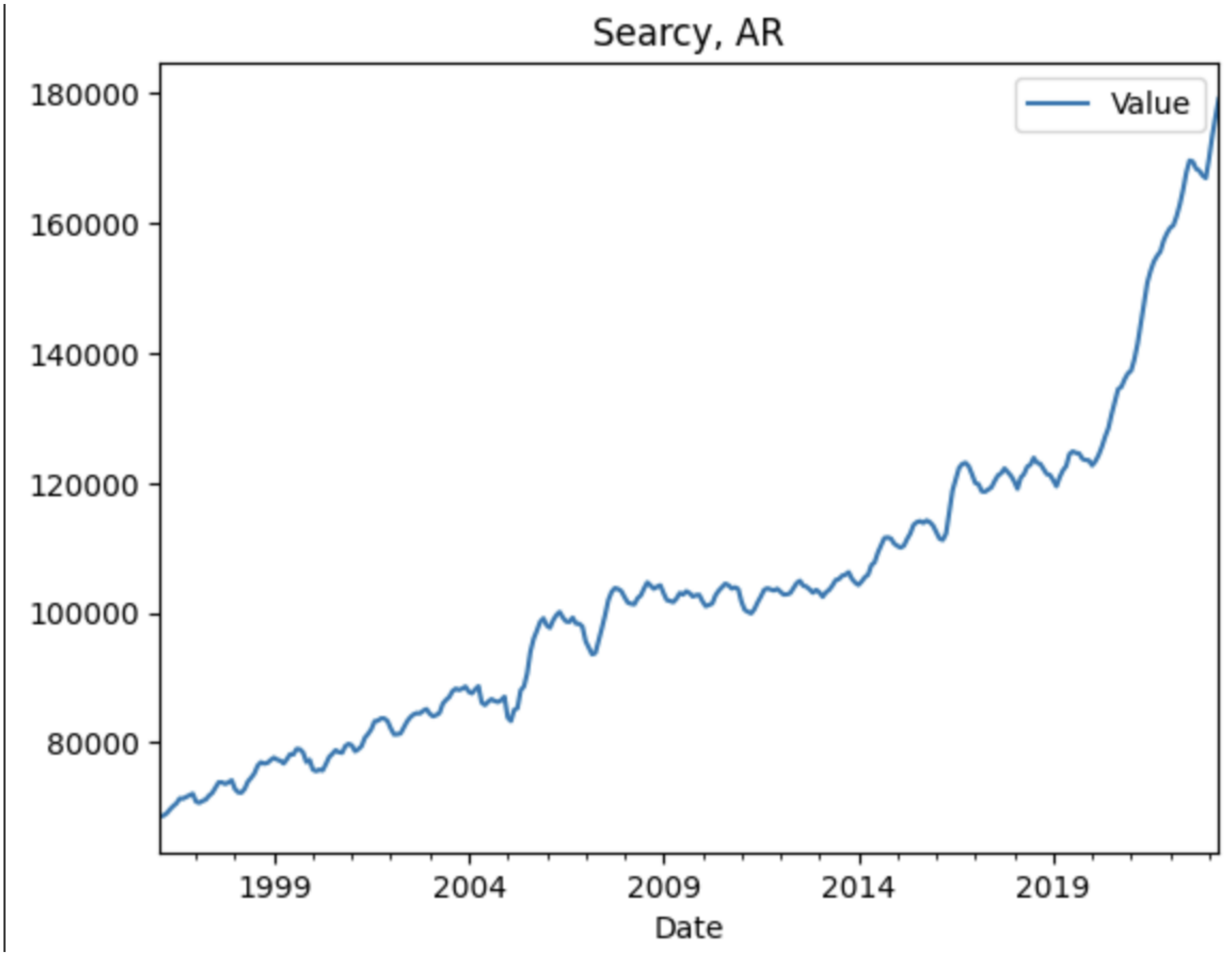
The Fayetteville-Springdale-Rogers, Arkansas area seems to be following a similar trend, albeit with a different price ‘Value’ range as seen in the Y axis.



A similar trend is seen again with the Hot Springs, Arkansas area; however, there seems to be an outlier spike around the 2010 timeframe. In order to specifically understand the reasoning for this spike, additional research would need to be made for this area; perhaps reviewing real estate news articles from this time period would help – or even set up persona interviews with real estate agents from that time period.



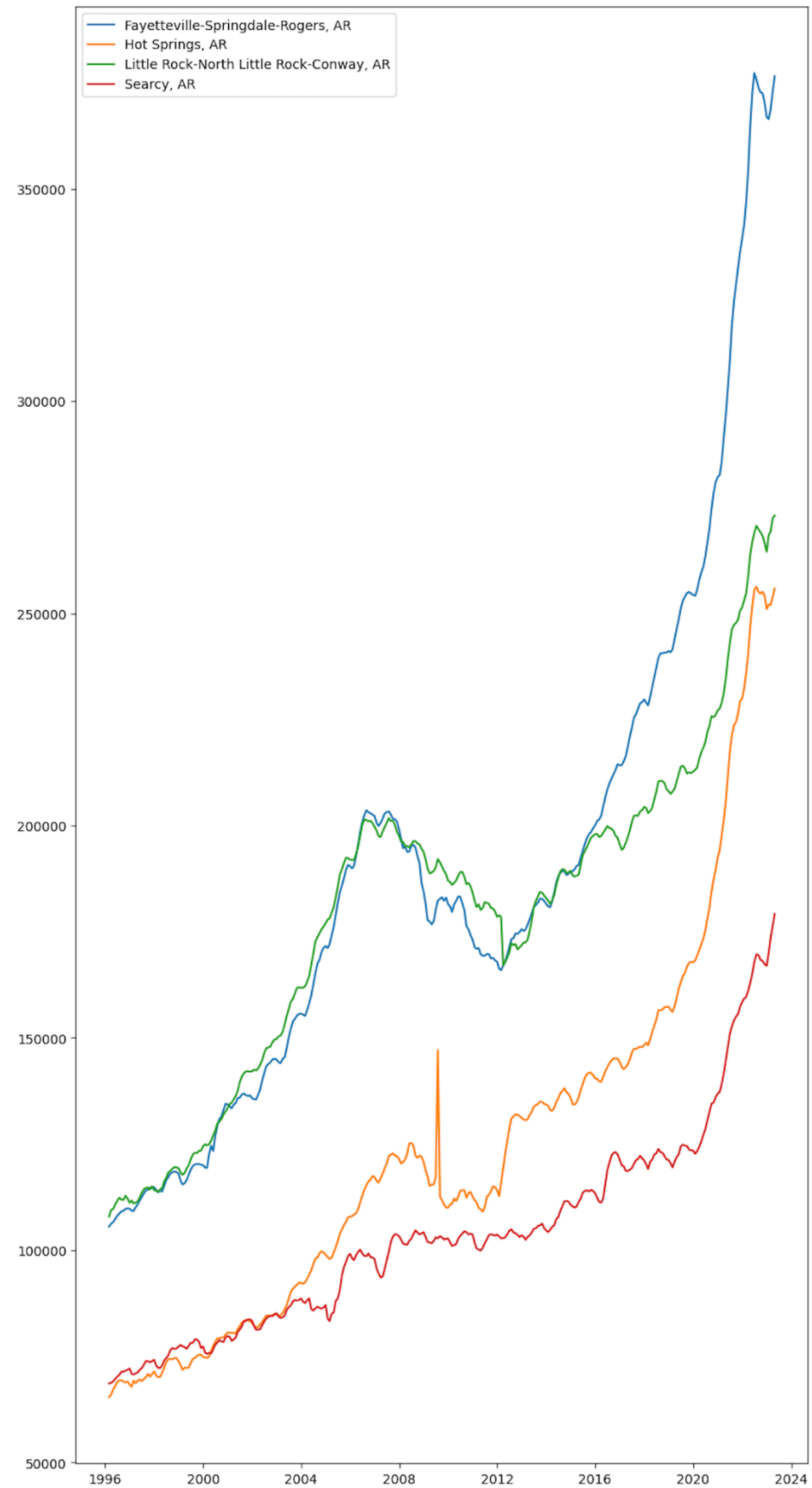
Following a similar trend, the Little rock-North Little Rock-Conway, Arkansas area looks to be on-par with the other mentioned areas.



Lastly, the Searcy, Arkansas area, although follows an upward trend, seems to be more a linear trend when removing seasonality from the dataset; however, the ceiling for prices is lower than the rest, topping out at ~$18,000.

If we were to choose one of these areas based on ceiling, it looks as the investment choice would perhaps depend on how much capital is available for the fund to distribute; more properties could be purchased at lower price-points in the Searcy market, while most likely continuing to see premium appreciation in market-value; however, investment decisions should rely on many factors, not just price.

Another way to look at the data would be to view all graphs on the same visualization chart, as depicted below:



Fayetteville-Springdale-Rogers, AR
Hot Springs, AR
Little Rock-North Little Rock-Conway, AR
Searcy, AR
350000
300000
250000
200000
150000
100000
50000
1996
2000
2004
2008
2012
2016
2020
2024

Data Modeling

For the data modeling portion, two models will be used:

- SARIMAX
    - Testing a random zip code from the dataset
- Prophet
    - Predicting each zip code in the dataset

SARIMAX

For the SARIMAX model, a random zip code was chosen for prediction via the following code:

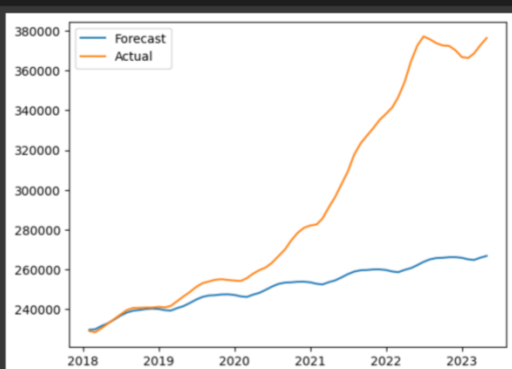
```
# Train the best SARIMAX model
best_model = SARIMAX(train_data['Value'], order=best_order, seasonal_order=best_seasonal_order, enforce_stationarity=False, enforce_invertibility=False)
best_model_fit = best_model.fit()

# Forecasting for 2018
forecast_steps = len(test_data)
forecast = best_model_fit.forecast(steps=forecast_steps)

# Calculate the RMSE for the model
rmse = np.sqrt(mean_squared_error(test_data['Value'], forecast))
print(f"RMSE for the SARIMAX model: {rmse:.2f}")
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
RMSE for the SARIMAX model: 57973.54
```

```
plt.plot(test_data['Date'], forecast, label='Forecast')
plt.plot(test_data['Date'], test_data['Value'], label='Actual')
plt.legend(loc='upper left')
plt.show()
```



The forecasts appear to be relatively accurate for the years of 2018-2019; however, the forecasted values appear to keep the seasonality trend going into the 2023, whereas the actual values spike higher than expected. This conveys to the audience that forecasting real estate is not an exact science, and (quite frankly) anything can happen in the real estate market. The RMSE for the equation appears to be 57973.54, which is not an optimal number. The closer to zero, theoretically the better our equation prediction would be.

Another approach may be needed, and we decide to utilize Facebook/Meta's prophet forecasting model to predict on all zip codes in the dataset:

Prophet

Meta's prophet model, which predicted real estate 'Values' for each zip code in the dataset, took over 1.5 hours to complete. Once executed, the predicted values were stored in a data frame labeled 'results' and contained predictions for the Syracuse Real Estate fund problem statement.

```
[ ] # Create an empty dataframe to store the forecast results
    results = pd.DataFrame(columns=['RegionName', 'Forecast'])

    df = df_all_states_long

    # Loop over each unique zip code in the dataset
    for zip_code in df['RegionName'].unique():
        # Filter the dataframe to only include data for the current zip code
        zip_df = df[df['RegionName'] == zip_code]

        # Rename the columns to fit with Prophet's required format
        zip_df = zip_df.rename(columns={'Date': 'ds', 'Value': 'y'})

        # Initialize and fit the model
        m = Prophet()
        m.fit(zip_df)

        # Create a dataframe for future predictions
        future = m.make_future_dataframe(periods=12)  # predicting for next 12 months

        # Generate the forecast
        forecast = m.predict(future)

        # Store the forecast for the last period (e.g., the next 12 months)
        results = results.append({'RegionName': zip_code, 'Forecast': forecast['yhat'].iloc[-1]}, ignore_index=True)

    # Sort the results by forecasted home price
    results = results.sort_values(by='Forecast', ascending=False)

    # Print the top 3 zip codes with the highest forecasted home prices
    print(results.head(3))
```

An example of the output, as per typical of the prophet model, looked similar to the below:

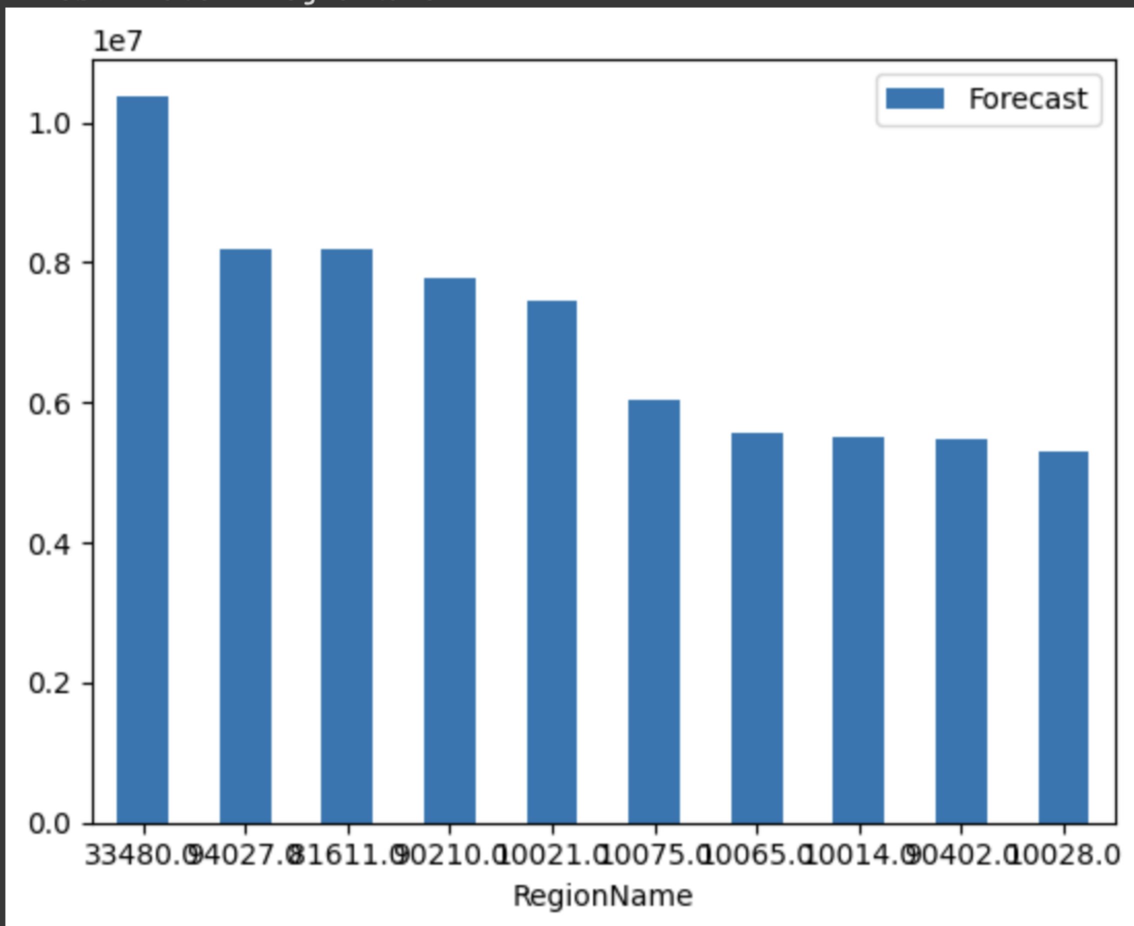
```
Streaming output truncated to the last 5000 lines.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/13bwj_ue.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=92406', 'data', 'file=/tmp/tmpadc3q91a/iay9il2f.json', 'init=/tmp/tmpadc3q91a/13bwj_ue.json', 'output',
01:18:51 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
01:18:51 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<ipython-input-8-35992719aafb>:28: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  results = results.append({'RegionName': zip_code, 'Forecast': forecast['yhat'].iloc[-1]}, ignore_index=True)
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/1lkaftmb.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/7bdzfb6d.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=79390', 'data', 'file=/tmp/tmpadc3q91a/1lkaftmb.json', 'init=/tmp/tmpadc3q91a/7bdzfb6d.json', 'output',
01:18:52 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
01:18:52 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<ipython-input-8-35992719aafb>:28: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  results = results.append({'RegionName': zip_code, 'Forecast': forecast['yhat'].iloc[-1]}, ignore_index=True)
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/lwyo31it.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/ufnxq8xn.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=84556', 'data', 'file=/tmp/tmpadc3q91a/lwyo31it.json', 'init=/tmp/tmpadc3q91a/ufnxq8xn.json', 'output',
01:18:52 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
01:18:52 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<ipython-input-8-35992719aafb>:28: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  results = results.append({'RegionName': zip_code, 'Forecast': forecast['yhat'].iloc[-1]}, ignore_index=True)
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/oidira11.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/_wrxb9n6.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=53079', 'data', 'file=/tmp/tmpadc3q91a/oidira11.json', 'init=/tmp/tmpadc3q91a/_wrxb9n6.json', 'output',
01:18:52 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
01:18:53 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<ipython-input-8-35992719aafb>:28: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  results = results.append({'RegionName': zip_code, 'Forecast': forecast['yhat'].iloc[-1]}, ignore_index=True)
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/i00gtkwi.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpadc3q91a/uy8db42r.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=473', 'data', 'file=/tmp/tmpadc3q91a/i00gtkwi.json', 'init=/tmp/tmpadc3q91a/uy8db42r.json', 'output',
01:18:53 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
```

Conclusions

In our conclusion section, we answer the main question of ‘Which three zip codes offer the best investment opportunity for Syracuse in the future?’ Once the prophet model finished it’s 1.5 hour execution, a bar plot of the highest forecasted zip code values were shown:

```python
top_10 = results.head(10)
top_10.plot.bar(x='RegionName', y='Forecast', rot=0)
```

```
<Axes: xlabel='RegionName'>
```




```python
top_10.head(3)
```

	RegionName	Forecast
8094	33480.0	1.037687e+07
20132	94027.0	8.200469e+06
18264	81611.0	8.185935e+06

Accompanying the visualization is the a table that consists the zip code (RegionName) as well as the forecasted price value (Forecast) that addresses out problem statement.

A join to the original long-formatted table allows the reader to view the Metro area names for the zip codes:

```
merged_df = pd.merge(top_10.head(3), df_all_states_long[['RegionName', 'Metro']], on='RegionName')
merged_df = merged_df.drop_duplicates(subset=['RegionName'])
merged_df
```

	RegionName	Forecast	Metro
0	33480.0	1.037687e+07	Miami-Fort Lauderdale-Pompano Beach, FL
327	94027.0	8.200469e+06	San Francisco-Oakland-Berkeley, CA
654	81611.0	8.185935e+06	Glenwood Springs, CO

As noted above, the Miami/Ft. Lauderdale, San Francisco/Oakland, and Glenwood Springs areas appear to have the highest forecasted values.

Lab Questions

Lab Question 1

What technique/algorithm/decision process did you use to down sample? (BONUS FOR NOT DOWN SAMPLING)

Ultimately, the only down sampling that was conducted in the code was removing nulls for the non-date columns. These would have been to difficult to interpret – although we could have taken the mode of each column for the null-value. Other than this, no other data was removed to make the training and processing time faster.

This decision was felt during the model fitting process. As stated earlier in the report, the prophet model took over 1.5 hours to fit on each zip code in the data set.

Lab Question 2

What three zip codes provide the best investment opportunity for the SREIT? Why?

As shown in the conclusions section, the areas with the best investment opportunity (based on the model's results) were the following:

- Miami-Fort Lauderdale-Pompano Beach, FL
- San Francisco-Oakland-Berkeley, CA
- Glenwood Springs, CO

The decisions were based on forecasted future home value. Some investors may disagree with the model's predictions; which could be a fair argument. Just because these areas may have the highest forecasted values does not mean they necessarily are the best investment opportunity. An investment requires a down payment as well as continued loan payoff (assuming a loan is taken). Some investors may look for areas with cheaper purchase prices; however, the steep appreciation coastal areas appear to get throughout the decades is an aspect that even the stingiest investor cannot immediately criticize.

The only exception to this rule would be the Colorado location. A scenic area by mountain tops can also see impactful appreciation as time goes on. These markets appear to be a great investment choice for the SREIT group. Additional factors such as crime rates, weather patterns, or even future job growth may also adjust the predictions of the model – which should be explored in future studies.