Jeff Levesque

Professor Fox

IST 718

Lab #2

**Introduction**

Property is often thought as one of the safer investment opportunities compared to various alternatives. This often brings to question; how does one analyze or predict investment property? Can linear regression, or time series data be implemented? In this study various questions will try to address particular zip codes can be identified as better investment opportunities. Using forecasting techniques, a model will be generated to help predict mean and median values for successive months in 2017, as well as the following year in 2018.

**Analysis**

Data Preparation:

A single csv file was obtained from Zillow, and version controlled with the codebase:

* <https://github.com/jeff1evesque/ist-718-lab/blob/master/data/Zip_Zhvi_SingleFamilyResidence.csv>

There were 7 columns representing attributes of region, and SizeRank:

* RegionID
* RegionName
* City
* State
* Metro
* CountyName
* SizeRank

Additionally, the dataset had monthly timeseries data ranging from 1996-04 through 2007-09. The composite of these columns contained 15283 rows of data. For preprocessing, column types were generally casted either string, or integer:

df[['City', 'State', 'Metro', 'CountyName']] = df[['City', 'State', 'Metro', 'CountyName']].astype(str)

df[['RegionID', 'RegionName', 'SizeRank']] = df[['RegionID', 'RegionName', 'SizeRank']].astype(int)

df[['City', 'State', 'Metro', 'CountyName']] = df[['City', 'State', 'Metro', 'CountyName']].apply(

lambda x: x.astype(str).str.lower()

)

Additionally, the timezone column was created using the zipcodes package:

def get\_zipcode(city, state):

result = zipcodes.filter\_by(

zipcodes.list\_all(),

active=True,

city=city,

state=state

)

if result and result[0] and result[0]['zip\_code']:

return(result[0]['zip\_code'])

else:

return(0)

df['zip\_code'] = df[['City', 'State']].apply(

lambda x: get\_zipcode(

x['City'].upper(),

x['State'].upper()

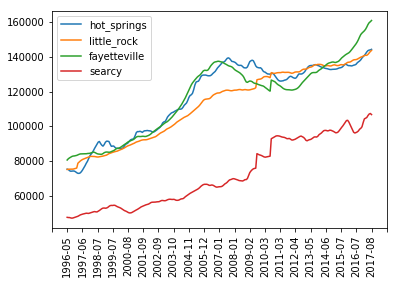
),

axis=1

)

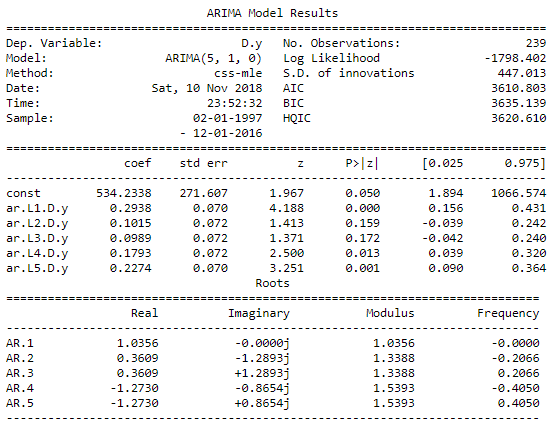
**Results**

Time series models were generated for metro areas in Arkansas:



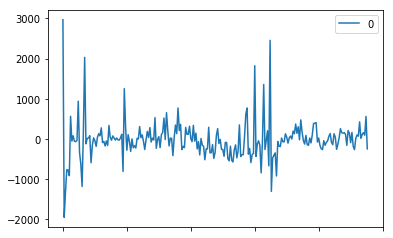
**Figure 1**. Time series plots for metro areas in Arkansas. The code used to generate the plot above can be reviewed in Appendix A below.

An ARIMA model was generated using a train dataset from 01/1997 to 01/2017:

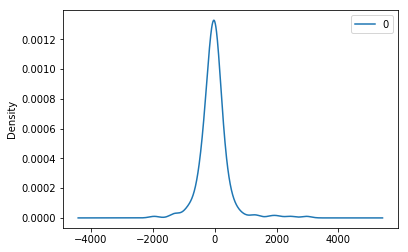


**Figure 2**. Descriptive statistics for an overall ARIMA model between 01/1997 and 01/2017. The code used to generate the plot above can be reviewed in Appendix B below.

An overall residual (Figure 3), and kernel density estimation (kde) plot (Figure 4) were generated using the same train dataset:

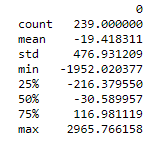


**Figure 3**. Residual plot for an overall ARIMA model. The code used to generate the plot above can be reviewed in Appendix C below.



**Figure 4**. KDE plot for an overall ARIMA model. The code used to generate the plot above can be reviewed in Appendix D below.

Descriptive statistics for the overall ARIMA model:



**Figure 5**. Descriptive statistics for the overall ARIMA model. The code used to generate the plot above, print(residuals.describe()).

The predicted values with error were computed for the remaining months in 2017, which were not in the train dataset (Figure 6), as well as for the 2018 months (Figure 7):

===============================================

date: 2017-2

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

predicted=177232.983241, expected=178300.000000

prediction difference: 0.005984

===============================================

predicted=177232.983241

===============================================

date: 2017-3

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

predicted=178106.353491, expected=179200.000000

prediction difference: 0.006103

===============================================

predicted=178106.353491

===============================================

date: 2017-4

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

predicted=179025.391283, expected=179900.000000

prediction difference: 0.004862

===============================================

predicted=179025.391283

===============================================

date: 2017-5

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

predicted=179963.396485, expected=180600.000000

prediction difference: 0.003525

===============================================

predicted=179963.396485

===============================================

date: 2017-6

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

predicted=180780.276282, expected=181300.000000

prediction difference: 0.002867

===============================================

predicted=180780.276282

===============================================

date: 2017-7

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

predicted=181605.509884, expected=182000.000000

prediction difference: 0.002168

===============================================

predicted=181605.509884

===============================================

date: 2017-8

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

predicted=182440.212917, expected=182500.000000

prediction difference: 0.000328

===============================================

Test MSE: 562014.511865

**Figure 6**. ARIMA prediction and error rate, from the remaining 2017 months not in train. The code used to generate the plot above can be reviewed in Appendix D below.

===============================================

date: 2017-9

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

predicted=183357.985077

===============================================

date: 2017-10

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

predicted=184195.715598

===============================================

date: 2017-11

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

predicted=185004.617198

===============================================

date: 2017-12

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

predicted=185813.748975

===============================================

date: 2018-1

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

predicted=186627.245588

===============================================

date: 2018-2

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

predicted=187427.229121

===============================================

date: 2018-3

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

predicted=188213.890561

===============================================

date: 2018-4

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

predicted=188989.242015

===============================================

date: 2018-5

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

predicted=189759.429189

===============================================

date: 2018-6

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

predicted=190524.215284

===============================================

date: 2018-7

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

predicted=191280.228248

===============================================

date: 2018-8

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

predicted=192027.656753

===============================================

date: 2018-9

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

predicted=192767.630680

===============================================

date: 2018-10

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

predicted=193501.557233

===============================================

date: 2018-11

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

predicted=194229.338516

===============================================

date: 2018-12

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

predicted=194950.383820

===============================================

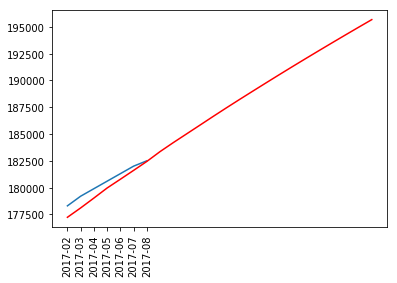
date: 2018-1

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

predicted=195664.981670

**Figure 7**. ARIMA predictions for 2018 months. The code used to generate the plot above can be reviewed in Appendix E below.

To visualize the difference between the predicted and actual values:



**Figure 8**. Comparison of the predicted vs. actual mean value. The code used to generate the plot above can be reviewed in Appendix F below.

Each zipcode data was passed to a custom compute\_arima method. However, not all successfully ran for the full iterative rolling forecasts. Three randomly selected rolling ARIMA forecasts generated for a respective zipcode:

predictions: [array([339759.72842923]), array([166549.71442558]), array([170521.08503644]), array([169660.17688667]), array([170560.93171876]), array([177894.63094519]), array([182695.48339773]), array([183185.48213607]), array([183871.94261669]), array([184557.14197649]), array([185241.97868413]), array([185930.19550277]), array([186620.70063724]), array([187311.33729023]), array([188002.12122458]), array([188693.08684187]), array([189384.17788984]), array([190075.31397832]), array([190766.46689865]), array([191457.63297228]), array([192148.8074253]), array([192839.98593885]), array([193531.16629146]), array([194222.34769062])]

predictions: [array([247853.60493319]), array([74947.97986236])]

predictions: [array([295626.72028987]), array([54935.44916106]), array([90340.41310928])]

**Figure 9**. A 24 month prediction beginning 2017/02 representing the zipcode in the aggregated list.Appendix F and Appendix G demonstrate the code to iterate all zipcode in the provided dataset.

Any rolling forecast with less than the full 24 months prediction, generated an error stating LinAlgError: SVD did not converge. This was handled in the code, by catching the exception, then forcing the iteration to skip to the next. Some obvious possible reasons for this error include an infinite number of elements, or an NaN element supplied to the ARIMA method. However, an explicit attempt was made to fill NaN values with zero’s when reading the initial dataset:

with open('../data/Zip\_Zhvi\_SingleFamilyResidence.csv', 'rb') as f:

df = pd.read\_csv(f).fillna(0)

Upon further review, others have discovered a similar problem linked to the version of anaconda on a windows platform[[1]](#footnote-1). This may be the source of the problem with prediction instances not being fully complete. Additionally, the time associated with running the ARIMA model on the entire dataset, spans no less than 16 hours. Several times, the runtime was prematurely terminated to allow coding enhancements, this greatly presented challenges, especially in the case when a result is accidentally cleared in the jupyter notebook, an uncommon happening.

Better techniques including creating a smaller dataset, could allow faster debugging. Additionally, sampling would be a better approach on a future instance if computing speed was a factor of importance. However, the dataset itself is not large with roughly 15,000 rows. In fact, implementing sampling with or without replacement, could generate a model that under-represents the given dataset population. More specifically, there are 42,000 zipcodes in the United States. Therefore, if sampling is implemented in the future, the sample should be significant enough. However, if the level of significance drives the sample size beyond a time effective limit, then efforts would be no different than attempting to analyze the entire dataset.

An attempt could be made to assess the top zipcode values. This could contain significant error, when discarding the incomplete predicted instances with less than 24 predictions. However, the approach would entail many more hours of compute time on a laptop machine. Other considerations could involve researching alternative algorithms with faster compute times.

**Conclusions**

As stated in the results section, this study discovered some limitations potentially due to technology, and software. In practice, if a business finds value for a given research question, then resources would be allocated to compute the given task. Without the required resources, a small data problem can quickly seem to be a big data task. Likewise, in the same business world, there’s a common notion of doing more with less. This could potentially be achieved, by being clever selecting a robust, yet time efficient model. Therefore, as stated earlier, comparing the selecting the most optimal algorithm is an important step for any research.

In this study, Figure 1 shows amongst four Arkansas metro area, each depicting a similar trend, while Fayetteville having much greater recent value. Next, an overall ARIMA rolling model showed (Figure 6) a highly accurate model. Specifically, at each interval-step, the predicted value contained less than a 1% error. Therefore, this approach was adopted when attempting to aggregate the dataset by zipcode, then predicting the next two years worth of predition.

Though this study did not achieve generating an answer to the top three zipcodes, it has successfully identified steps to be taken on a future study. As stated in the results section, a comparison on the most optimal algorithm needs to be made. Additionally, a consideration of sampling may not be chosen, but using a smaller dataset during training could improve efficiencies during analysis.

**Appendix A**

Arkansas Metro Area

# metro areas

hot\_springs = df.loc[(df['Metro'] == 'hot springs') & (df['State'] == 'ar')]

little\_rock = df.loc[(df['Metro'] == 'little rock') & (df['State'] == 'ar')]

fayetteville = df.loc[(df['Metro'] == 'fayetteville') & (df['State'] == 'ar')]

searcy = df.loc[(df['Metro'] == 'searcy') & (df['State'] == 'ar')]

# timeseries plot

fig, ax = plt.subplots()

ax.plot(hot\_springs[date\_columns].mean(), linestyle='solid')

ax.plot(little\_rock[date\_columns].mean(), linestyle='solid')

ax.plot(fayetteville[date\_columns].mean(), linestyle='solid')

ax.plot(searcy[date\_columns].mean(), linestyle='solid')

# decrease ticks

xmin, xmax = ax.get\_xlim()

ax.set\_xticks(np.round(np.linspace(xmin, xmax, 23), 2))

# rotate ticks + show legend

plt.xticks(rotation=90)

plt.gca().legend(('hot\_springs', 'little\_rock', 'fayetteville', 'searcy'))

# show overall plot

plt.show()

**Appendix B**

Overall ARIMA model descriptive statistics between 01/1997 through 01/2017:

# train: collapse column by median

train\_start = df.columns.get\_loc('1997-01')

train\_stop = df.columns.get\_loc('2017-01')

test\_stop = df.columns.get\_loc('2017-09')

train\_columns = df.iloc[:, train\_start:train\_stop].columns.tolist()

test\_columns = df.iloc[:, (train\_stop + 1):test\_stop].columns.tolist()

# transpose dataframe: left column data, right column value

df\_train = df[train\_columns].median().T

df\_test = df[test\_columns].median().T

# build arima model:

model = ARIMA(df\_train, order=(5,1,0))

model\_fit = model.fit()

print(model\_fit.summary())

**Appendix C**

Residual plot for overall ARIMA model:

# plot residual errors

residuals = DataFrame(model\_fit.resid)

residuals.plot()

plt.show()

**Appendix D**

KDE plot for overall ARIMA model:

# plot kernel density estimation

residuals.plot(kind='kde')

plt.show()

**Appendix D**

Error rate for ARIMA model, with remaining 2017 months through 2018 predictions:

history = [x for x in df\_train]

predictions = list()

iterations = (12-len(df\_test)) + 19

for t in range(iterations):

model = ARIMA(history, order=(5,1,0))

model\_fit = model.fit(disp=0)

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat)

if t > 10:

year = 2018

month = (t+2) % 12

if month == 0:

month = 12

else:

year = 2017

month = t+2

if month == 0:

month = 12

print('date: {}-{:01d}'.format(year, month))

try:

obs = df\_test[t]

print('predicted={:03f}, expected={:03f}'.format(float(yhat), obs))

print('prediction difference: {:03f}'.format(abs(1-float(yhat)/obs)))

error = mean\_squared\_error(df\_test, predictions)

print('Test MSE: {:03f}\n\n'.format(error))

except:

obs = yhat

print('predicted={:03f}'.format(float(yhat)))

history.append(obs)

**Appendix E**

Comparison between predicted vs actual mean value:

# plot rolling prediction

plt.plot(df\_test)

plt.plot(predictions, color='red')

plt.xticks(rotation=90)

plt.show()

**Appendix F**

Generic arima function using rolling prediction:

def compute\_arima(df\_train):

history = [x for x in df\_train]

print(history)

predictions = list()

iterations = (12-len(df\_test)) + 19

for t in range(iterations):

model = ARIMA(history, order=(5,1,0))

try:

model\_fit = model.fit(disp=0)

output = model\_fit.forecast()

yhat = output[0]

except error as e:

print('===========================')

print('skipped output: {}, error: {}'.format(t, e))

print('===========================\n')

continue

try:

obs = df\_test[t]

except:

obs = yhat

history.append(obs)

predictions.append(yhat)

print('predictions: {}'.format(predictions))

return(predictions)

**Appendix F**

Implementation of custom ARIMA rolling forecast model from Appendix G:

# group by zipcode

df\_zipcode = df.groupby('zip\_code').agg(np.median).dropna().T

#

# remove columns: column 0 indicates an NaN column

#

df\_zipcode\_clean = df\_zipcode.drop([

'RegionName',

'RegionID',

'SizeRank'

], axis=0)

df\_zipcode\_clean = df\_zipcode\_clean.drop([0], axis=1)

# iterate columns

results = []

for column in df\_zipcode\_clean:

predictions = compute\_arima(df\_zipcode\_clean[column])

results.append({

'zip\_code': df\_zipcode\_clean[column].name,

'predictions': predictions

})

1. <https://github.com/scipy/scipy/issues/4524> [↑](#footnote-ref-1)