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 attempt to address whether zip codes can be identified as better investment opportunities. Using forecasting techniques, an ARIMA model will be used to predict mean and median values for successive months in 2017, through 2018.

**Analysis**

Data Preparation:

Multiple datasets were implemented, and version controlled with the codebase:

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

Specifically, a main Zip\_Zhvi\_SingleFamilyResidence.csv was collected from Zillow, then used to estimate housing worth per city and state. This dataset covers a time series between 1996-01 through 2017-09. However, additional FBI crime data was aggregated based on state county:

* maryland.xls
* new-hampshire.xls
* virginia.xls
* district-of-columbia.xls

This allowed the former data to be filtered based on specific crime criteria. In the case of this study, the sum of all offense types divided by the county population (i.e. crime ratio). The ratio was not allowed to exceed 3%. Any county that exceed this value was omitted from the data aggregation. However, since there was only one row of data in the DC FBI dataset, its use was omitted, and possibly left as future exercise. Next labor force data was collected from the Department of Labor:

* laucnty17.xlsx

When joined with the latter combined data, the composition allowed further filtering based on unemployment rate. For this study, unemployment was not allowed to exceed 3.5%. Finally, the adjusted data was filtered on select locations to reduce the modeling:

* Maryland
* Virginia
* Washington D.C.
* Massachusetts
* New Hampshire

Furthermore, it is important to note that any NaN cell from the previous datasets were converted to zero. Additionally, to allow our dataframes to be joined together 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()

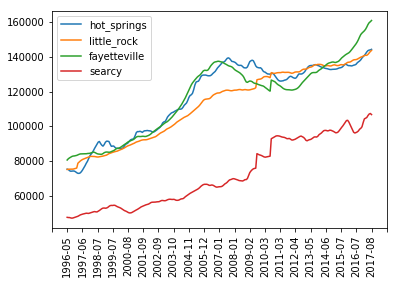
),

axis=1

)

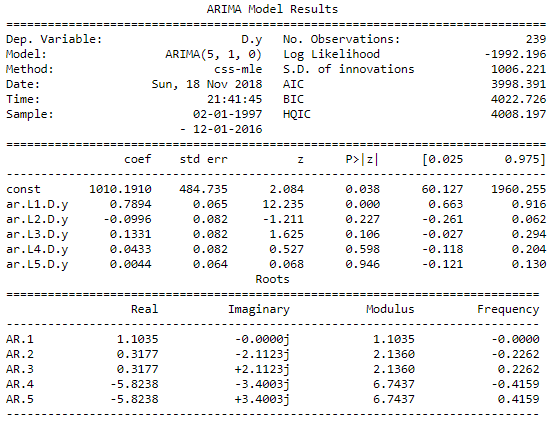
**Results**

Time series models were generated for metro areas in Arkansas:



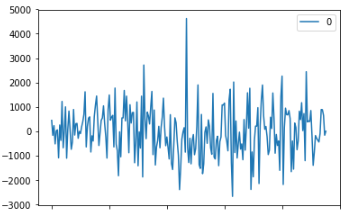
**Figure 1**. Time series plots for metro areas in Arkansas. The code used to generate the plot above can be reviewed in Appendix A below.

It appears that Fayetteville, Arkansas has the greatest increase of property value, followed by a visually difficult decision between Little Rock and Hot Springs Arkansas. Next, an ARIMA model was generated using a train dataset to determine whether a time series model could generalize housing data. This was done from 01/1997 to 01/2017:

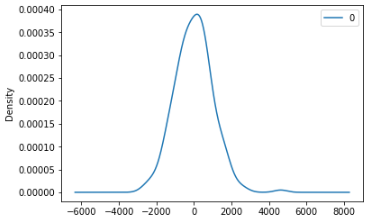


**Figure 2**. Descriptive statistics for an overall ARIMA model between 01/1997 and 01/2017 (train set). The code used to generate the plot above can be reviewed in Appendix B below.

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

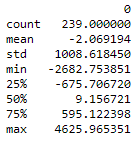


**Figure 3**. Residual plot for an overall ARIMA model. The code used to generate the plot above can be reviewed in Appendix C below.



**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=388813.576701, expected=391050.000000

prediction difference: 0.005719

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

predicted=388813.576701

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

date: 2017-3

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

predicted=390443.689396, expected=392900.000000

prediction difference: 0.006252

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

predicted=390443.689396

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

date: 2017-4

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

predicted=392003.290988, expected=395800.000000

prediction difference: 0.009592

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

predicted=392003.290988

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

date: 2017-5

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

predicted=393503.316512, expected=398850.000000

prediction difference: 0.013405

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

predicted=393503.316512

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

date: 2017-6

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

predicted=394958.772728, expected=400850.000000

prediction difference: 0.014697

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

predicted=394958.772728

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

date: 2017-7

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

predicted=396374.708561, expected=402250.000000

prediction difference: 0.014606

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

predicted=396374.708561

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

date: 2017-8

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

predicted=397752.815045, expected=403400.000000

prediction difference: 0.013999

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

Test MSE: 22164768.684165

**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=409223.452509

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

date: 2017-10

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

predicted=413533.867822

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

date: 2017-11

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

predicted=417515.922014

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

date: 2017-12

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

predicted=421472.969033

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

date: 2018-1

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

predicted=425233.650838

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

date: 2018-2

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

predicted=428722.288873

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

date: 2018-3

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

predicted=431986.176158

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

date: 2018-4

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

predicted=435068.894530

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

date: 2018-5

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

predicted=437986.877565

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

date: 2018-6

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

predicted=440749.383756

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

date: 2018-7

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

predicted=443369.575329

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

date: 2018-8

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

predicted=445861.222357

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

date: 2018-9

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

predicted=448234.798712

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

date: 2018-10

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

predicted=450501.831974

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

date: 2018-11

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

predicted=452672.004600

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

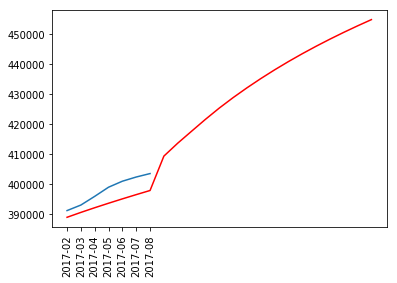
date: 2018-12

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

predicted=454753.750478

**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 E below.

Upon reviewing the above train case, it is evident that none of the difference between the predicted with the actual test data exceed 1.5%. Therefore, the implementation of ARIMA(5,1,0) for the entire population, was extended for each individual zip code. However, since the internal statsmodels.tsa.arima\_model does not allow d > 2[[1]](#footnote-1), the differencing was indirectly applied, by adapting the actual dataset.

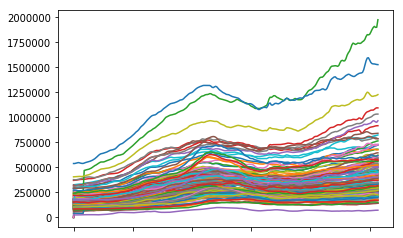
Each zipcode data was passed to a custom compute\_arima function. However, a direct implementation sometimes failed. Specifically, the statsmodels package sometimes failed with LinAlgError: SVD did not converge[[2]](#footnote-2). This was reconciled by incrementing the difference factor until the error was eliminated.

A computed rolling prediction successfully generated n predictions after the train:

predictions: [array([418001.35543155]), array([1052702.7826835]), array([747011.27115869]), array([388385.55041419]), array([803436.78946866]), array([800579.33426588]), array([803408.99540631])]

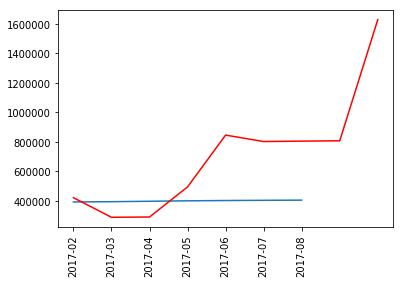
**Figure 10**. Rolling prediction for a given zipcode in the aggregated list. Appendix F and Appendix G demonstrate the code to iterate all zipcode in the provided dataset.

An aggregated plot of the full original dataset duration indicates similar trends:

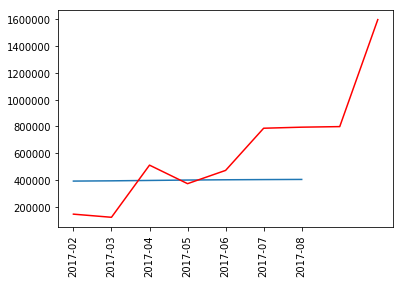


**Figure 11**. Zipcode trend for select zipcodes. The code used to generate the plot above can be reviewed in Appendix H below.

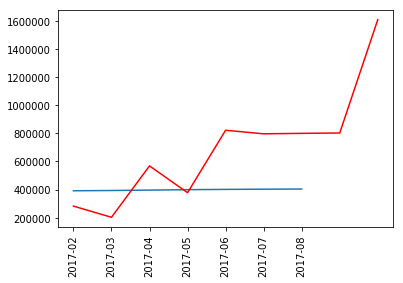
Based on the above plot, an iterative differencing is not appropriate. Specifically, a visually workable difference appears to be at least 1/3 of the data. Due to earlier discussed limitations, differencing was forced until the ARIMA model would successfully fit. Therefore, this approach would be susceptible to error prone predictions.



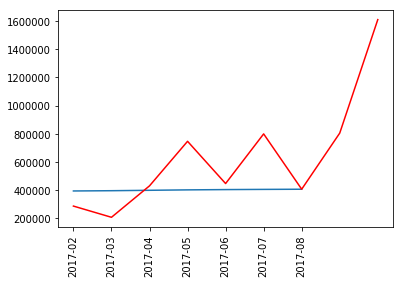
**Figure 11**. Most accurate model (red), against actual (blue) for Milton, NH (03851). The code used to generate the plot above can be reviewed in Appendix I below.



**Figure 12**. Second most accurate model (red), against actual (blue) for Berlin, NH (03570). The code used to generate the plot above can be reviewed in Appendix I below.



**Figure 13**. Third most accurate model (red), against actual (blue) for Troy, NH (03465). The code used to generate the plot above can be reviewed in Appendix I below.



**Figure 14**. Fourth most accurate model (red), against actual (blue) for Bethlehem, NH (03574). The code used to generate the plot above can be reviewed in Appendix I below.

Given the above results, an alternative fbprophet[[3]](#footnote-3) python package was briefly attempted. However, the package requires an additional Microsoft Visual C++ 14.0, and corresponding dependencies. Given it’s rich features, this approach should be further investigated.

**Conclusions**

As stated in the results section, this study discovered some limitations potentially due to technology, and software. One approach would be to investigate and acquire a better understanding of what algorithms works best. Some algorithms will have better performance, while others are more robust. Having this knowledge for a given platform, allows a data scientist to be more constructive at generating accurate predictions.

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 successive months of prediction.

When reviewing results of the top four zipcodes, it was found that the top four zipcodes aggregated by crime, unemployment, and by location (MD, VA, D.C., MA, NH) were found to be New Hampshire cities. However, as found in the results section, the modeling implementation was highly error prone. This is noticed in earlier Figure 10, as the random selected prediction fluctuated drastically.

**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

def rolling\_plot(data, predictions):

plt.plot(data)

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

plt.xticks(rotation=90)

plt.show()

rolling\_plot(df\_test, predictions)

**Appendix F**

Generic arima function using rolling prediction:

def compute\_arima(df\_train, p=5, q=0, d=0, iterations=len(df\_test), alpha=0.05):

history = [x for x in df\_train]

predictions = list()

for t in range(iterations):

delta = 1

model\_fit = False

try:

model = ARIMA(history, order=(p,q,d))

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

print('standard fit used')

except Exception as e:

print('stationary differences will be used')

print('original error: {}'.format(e))

if not model\_fit:

# determine stationarity value: differencing handled with supplied data,

# as an indirect solution, since statsmodel not allow d > 2.

#

# @delta, autoregressive factor.

for delta in range(10):

stationary = difference(history, delta)

stationary.index = history[1:]

result = adfuller(stationary)

print('stationary fit: {}, p: {}'.format(delta, result[1]))

# generate model: use high (10) autoregression, since data is not

# seasonal. Therefore, using previous values is conservative.

if (result[1] <= 0.05):

try:

model = ARIMA(stationary, order=(p,q,d))

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

break

except Exception as e:

print('bad condition {}: stationarity not adequate'.format(delta))

print('original error: {}'.format(e))

continue

# generate forecast: an inverse difference is needed to reverse the earlier

# difference model scaling.

if model\_fit:

output = model\_fit.forecast(alpha=alpha)

print(output)

yhat = inverse\_difference(history, output[0], delta)

# observation: if current value doesn't exist from test, append current

# predition, to ensure successive rolling prediction computed.

try:

obs = df\_test[t]

except:

obs = yhat

history.append(obs)

predictions.append(yhat)

else:

predictions.append(None)

break

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

return(predictions)

**Appendix G**

Implementation of custom ARIMA rolling forecast model from Appendix F:

# iterate columns

results = []

for column in df\_zipcode\_clean:

predictions = compute\_arima(df\_zipcode\_clean[column], p=5)

results.append({

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

'predictions': predictions

})

Associated helper functions for Appendix G:

# stationarity test

def difference(dataset, delta):

diff = list()

for i in range(1, len(dataset)):

value = dataset[i] - dataset[i - delta]

diff.append(value)

return pd.Series(diff)

# invert differenced value

def inverse\_difference(history, yhat, interval=1):

return yhat + history[-interval]

**Appendix G**

Zipcode trend for select zipcodes:

# iterate columns

results = []

for column in df\_zipcode\_clean:

predictions = compute\_arima(df\_zipcode\_clean[column], p=5)

results.append({

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

'predictions': predictions

})

**Appendix I**

Top four zipcodes:

# best 4 models

for model in sorted\_results[:4]:

# get data

zipcode = model['zipcode']

data\_zipcode = df\_zipcode\_clean[[zipcode]]

iterations = (12-len(df\_test)) + 18

data\_train = data\_zipcode.T[date\_columns].T

# compute\_arima

predictions = compute\_arima(data\_train.ix[:,0], iterations=iterations, p=5)

# plot predictions

rolling\_plot(df\_test, predictions)

1. <https://www.statsmodels.org/dev/_modules/statsmodels/tsa/arima_model.html> [↑](#footnote-ref-1)
2. <https://github.com/scipy/scipy/issues/4524> [↑](#footnote-ref-2)
3. <https://facebook.github.io/prophet/docs/quick_start.html> [↑](#footnote-ref-3)