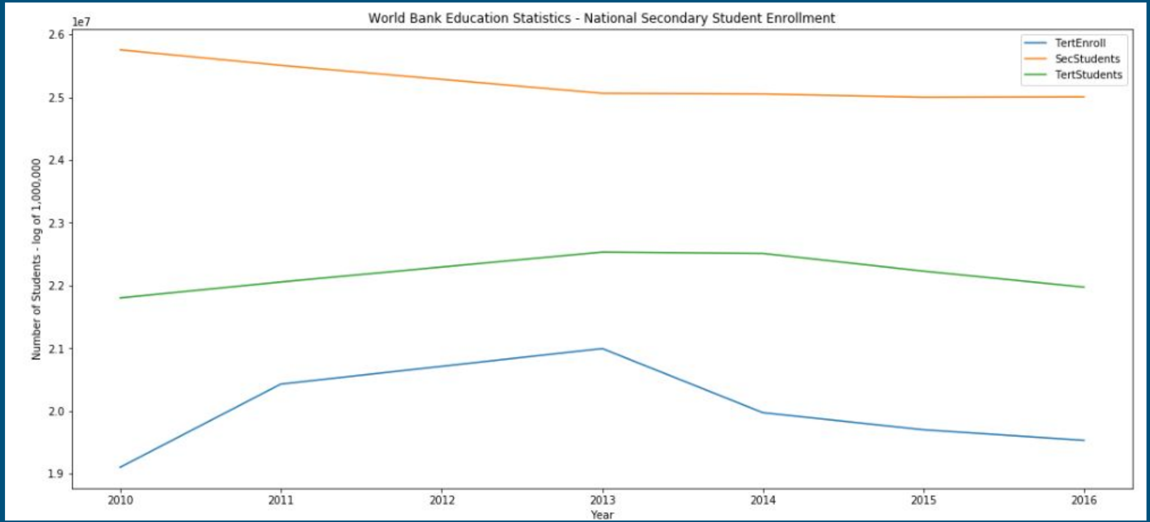
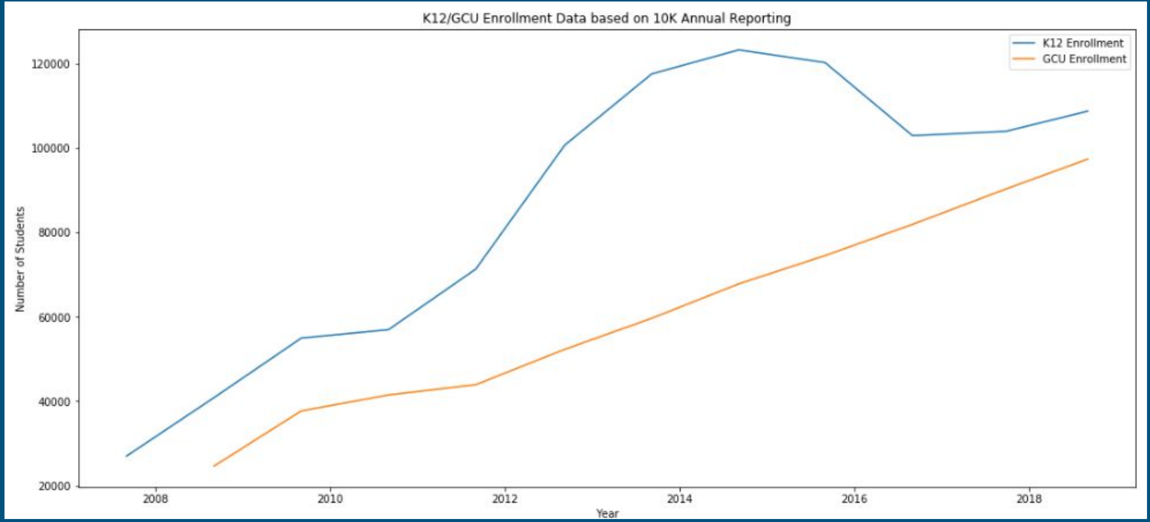
Thinkful Final Capstone

Presented by Jason Paik

Can student enrollment data affect stock price prediction of for-profit education companies?

- Passion for education in understanding the attractiveness of for-profit educational institutions - are these legitimate?
- Curious to know whether these companies were “successful” - way to measure this is through analyzing the trends in the stock price on market
- What features drive this “success”? Does enrollment affect this?
- K-12 (online high school) & Grand Canyon University (“GCU”; for-profit university) were my companies of focus



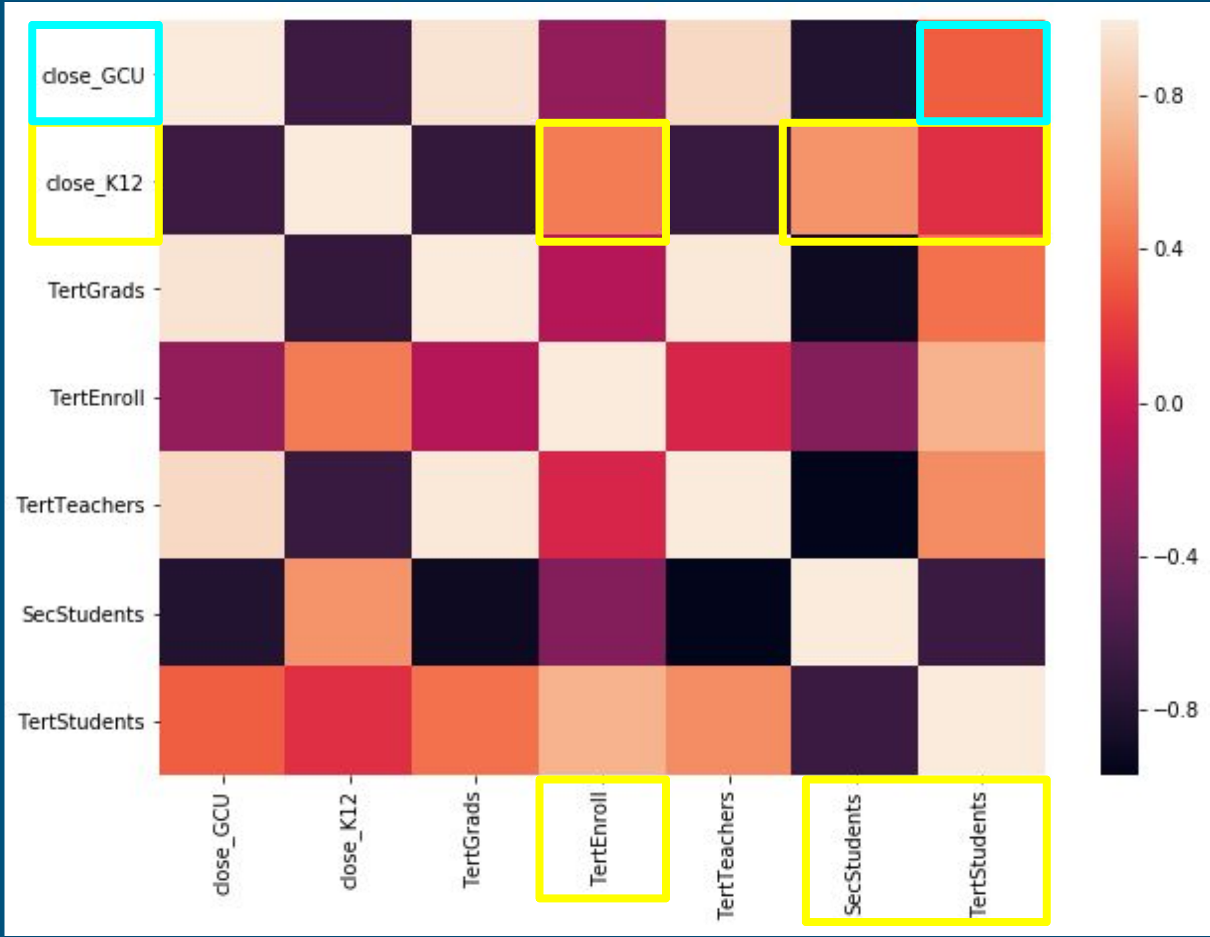


1) What does the market think about enrollment data? Was the stock price ever at all affected by enrollment data?

- WBES asserts that enrollment being quite static over the last 7 years

- 10-K annual reporting from K12 & GCU shows institutional enrollment increasing over last 10 years

- Stock price is quite unaffected by national enrollment figures
- Improving institutional enrollment sustains the stability of stock price



- Using a heat map, there are slightly positive correlations between GCU/K12 and student enrollment

- Rising enrollment at K12 compared to flat growth of national high school enrollment highlights trending popularity of cyber classrooms - stock price is affected by enrollment in some way

2) Can we predict the stock price of these two academic institutions <u>without</u> the consideration of enrollment data and purely on historical data?

- A/B testing lens - a model with just historical stock data and one with additional variables
- ARIMA modeling (Unit 6) and time-series analysis to help see this

```
modelGCU = pm.auto_arima(GCUdf.close_GCU, start_p=1, start_q=1,
                         test='adf',       # use adftest to find optimal 'd'
                         max_p=3, max_q=3, # maximum p and q
                         m=1,              # frequency of series
                         d=None,           # let model determine 'd'
                         seasonal=False,   # No Seasonality
                         start_P=0,
                         D=0,
                         trace=True,
                         error_action='ignore',
                         suppress_warnings=True,
                         stepwise=True)
print(modelGCU.summary())
```

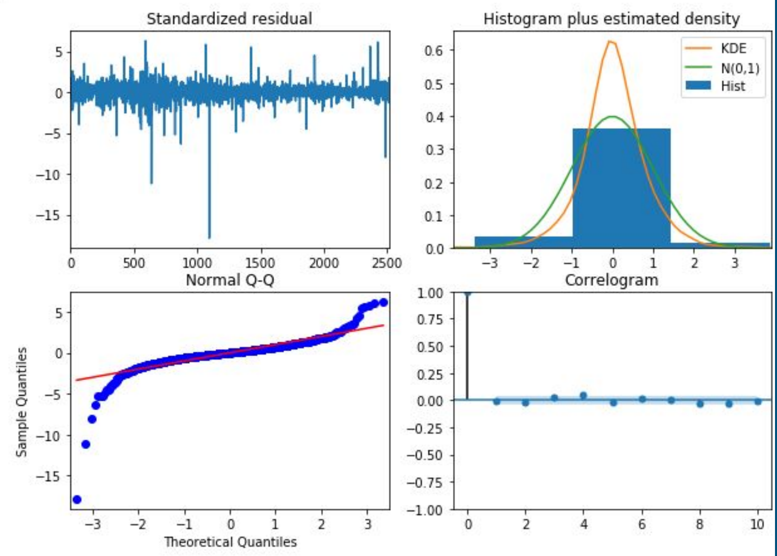
```
Fit ARIMA: order=(1, 1, 1); AIC=6880.269, BIC=6903.592, Fit time=0.221 seconds
Fit ARIMA: order=(0, 1, 0); AIC=6877.895, BIC=6889.557, Fit time=0.002 seconds
Fit ARIMA: order=(1, 1, 0); AIC=6879.231, BIC=6896.724, Fit time=0.023 seconds
Fit ARIMA: order=(0, 1, 1); AIC=6879.217, BIC=6896.709, Fit time=0.019 seconds
Total fit time: 0.272 seconds
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                 2517
Model:                 ARIMA(0, 1, 0)   Log Likelihood               -3436.948
Method:                           css   S.D. of innovations              0.948
Date:                Mon, 24 Jun 2019   AIC                           6877.895
Time:                        15:53:01   BIC                           6889.557
Sample:                             1   HQIC                          6882.127

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0442      0.019      2.339      0.019       0.007       0.081
==============================================================================
```

```
modelK12 = pm.auto_arima(K12df.close_K12, start_p=1, start_q=1,
                         test='adf',       # use adftest to find optimal 'd'
                         max_p=3, max_q=3, # maximum p and q
                         m=1,              # frequency of series
                         d=None,           # let model determine 'd'
                         seasonal=False,   # No Seasonality
                         start_P=0,
                         D=0,
                         trace=True,
                         error_action='ignore',
                         suppress_warnings=True,
                         stepwise=True)
print(modelK12.summary())
```

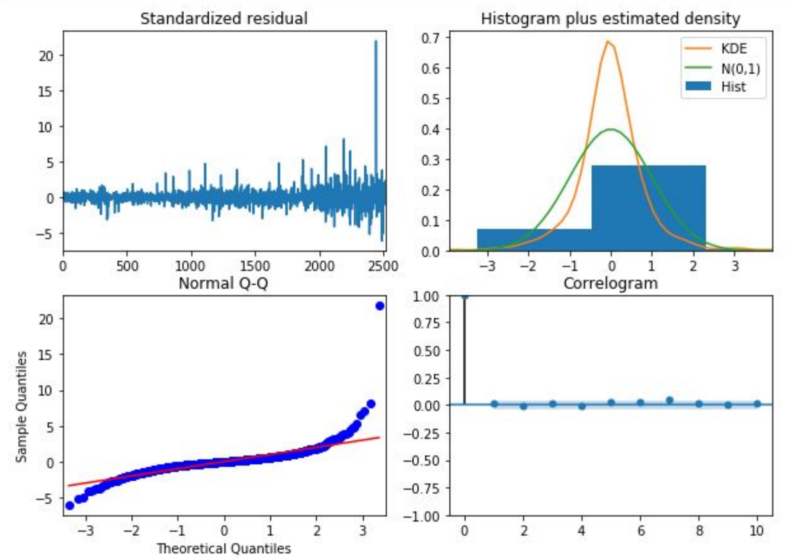
```
Fit ARIMA: order=(1, 1, 1); AIC=4668.898, BIC=4692.221, Fit time=0.318 seconds
Fit ARIMA: order=(0, 1, 0); AIC=4673.536, BIC=4685.198, Fit time=0.003 seconds
Fit ARIMA: order=(1, 1, 0); AIC=4672.198, BIC=4689.691, Fit time=0.037 seconds
Fit ARIMA: order=(0, 1, 1); AIC=4672.247, BIC=4689.740, Fit time=0.031 seconds
Fit ARIMA: order=(2, 1, 1); AIC=4670.652, BIC=4699.806, Fit time=0.360 seconds
Fit ARIMA: order=(1, 1, 2); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(2, 1, 2); AIC=nan, BIC=nan, Fit time=nan seconds
Total fit time: 1.007 seconds
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                 2517
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -2330.449
Method:                       css-mle   S.D. of innovations              0.611
Date:                Mon, 24 Jun 2019   AIC                           4668.898
Time:                        15:53:02   BIC                           4692.221
Sample:                             1   HQIC                          4677.363

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0059      0.014      0.424      0.672      -0.021       0.033
```

- Auto_Arima Python package - helps find lowest AIC score that uses fewer features to achieve same goodness of fit
- Used training/test splits taught in course to test ARIMA model

```
modelK12.plot_diagnostics(figsize=(10,7))
plt.show()
```



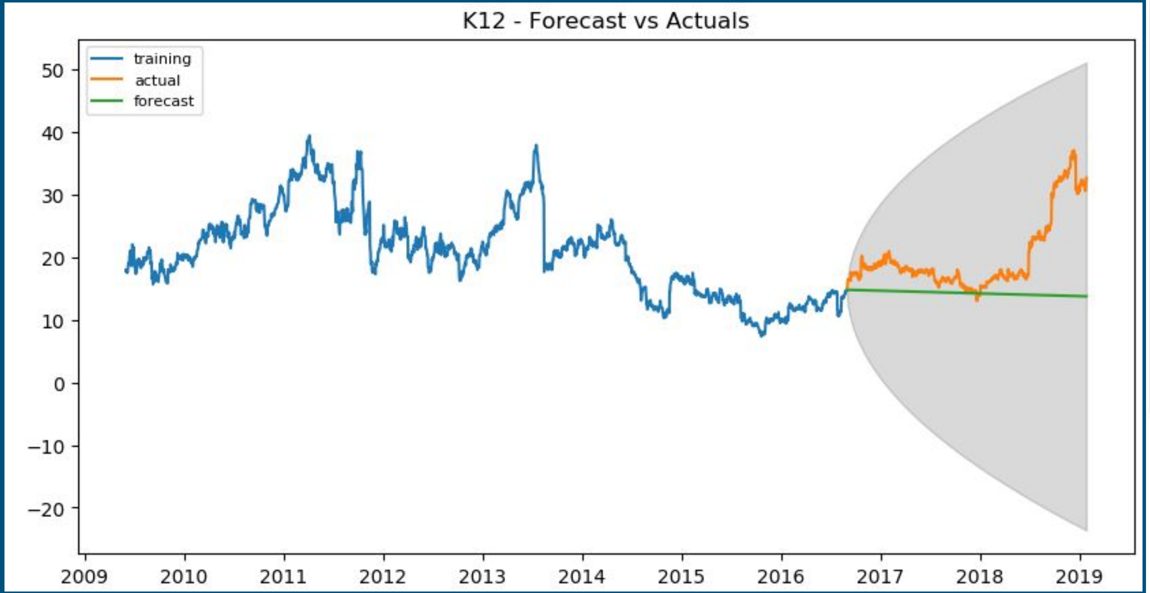

```
modelGCU.plot_diagnostics(figsize=(10,7))
plt.show()
```



- Data is not stationary for robust ARIMA modeling

- Standardized Residuals & ADF Stationarity Tests

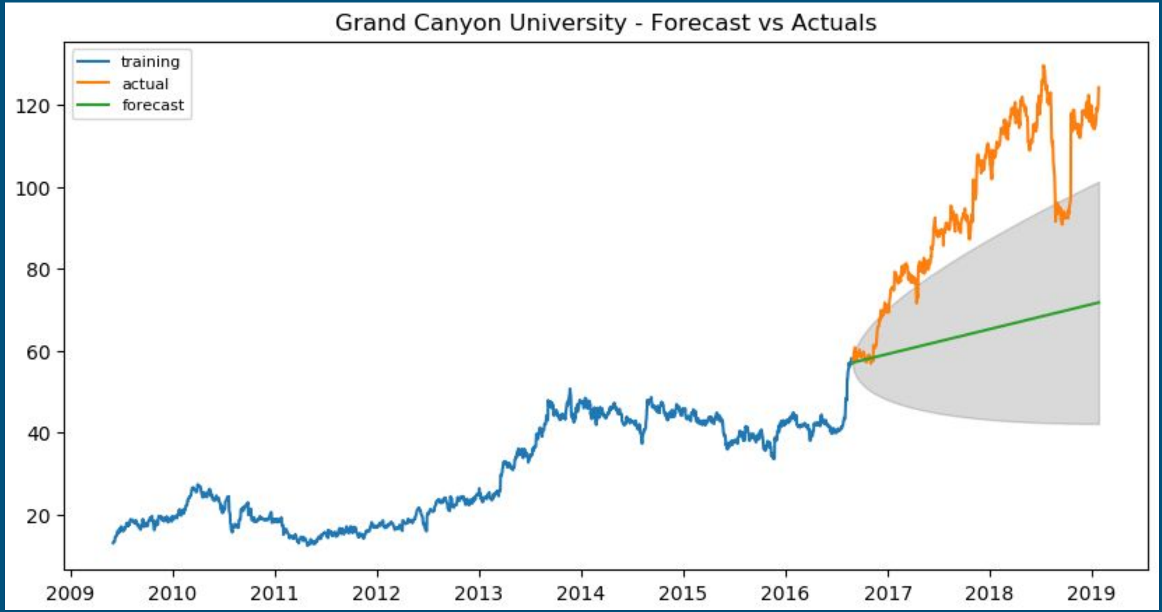
```
In [116]: resultGCU = adfuller(GCUdf.close_GCU)
          print('ADF Statistic for GCU Closing Stock Price: %f' % resultGCU[0])
          print('p-value: %f' % resultGCU[1])

ADF Statistic for GCU Closing Stock Price: 0.899404
p-value: 0.993083

In [117]: resultK12 = adfuller(K12df.close_K12)
          print('ADF Statistic for K12 Closing Stock Price: %f' % resultK12[0])
          print('p-value: %f' % resultK12[1])

ADF Statistic for K12 Closing Stock Price: -2.410103
p-value: 0.138905
```




```
### K12 MODEL
def forecast_accuracy1(forecast1, actual1):
    mape1 = np.mean(np.abs(forecast1 - actual1)/np.abs(actual1))  # MAPE
    me1= np.mean(forecast1 - actual1)             # ME
    mae1 = np.mean(np.abs(forecast1 - actual1))    # MAE
    mpe1 = np.mean((forecast1 - actual1)/actual1)   # MPE
    rmse1 = np.mean((forecast1 - actual1)**2)**.5  # RMSE
    corr1 = np.corrcoef(forecast1, actual1)[0,1]   # corr
    mins1 = np.amin(np.hstack([forecast1[:,None],
                              actual1[:,None]]), axis=1)
    maxs1 = np.amax(np.hstack([forecast1[:,None],
                              actual1[:,None]]), axis=1)
    minmax1 = 1 - np.mean(mins1/maxs1)             # minmax
    acfK12 = acf(fcK12-testK12)[1]                      # ACF1
    return({'mape':mape1, 'me':me1, 'mae': mae1,
            'mpe': mpe1, 'rmse':rmse1, 'acfK12':acfK12,
            'corr':corr1, 'minmax':minmax1})

forecast_accuracy1(fcK12, testK12.values)
```

```
{'mape': 0.24143460969328995,
 'me': -5.729557042034238,
 'mae': 5.742021293828014,
 'mpe': -0.2405118630505522,
 'rmse': 8.223037079993466,
 'acfK12': 0.9922247398257569,
 'corr': -0.6248833611677602,
 'minmax': 0.24141277529052863}
```

- K12 prediction is wide-ranging - stock price can’t go negative either
- Mean absolute percentage error: ~76% (mediocre)
- Root of mean squared error: 8.22/100 (errors are squared before averaged so RMSE gives relatively high weight to large errors)



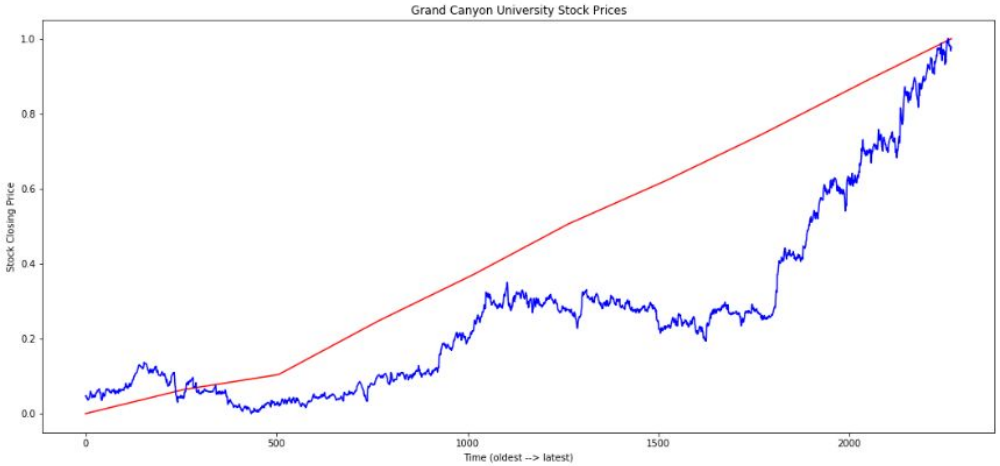

```
## GCU MODEL
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me = np.mean(forecast - actual)             # ME
    mae = np.mean(np.abs(forecast - actual))    # MAE
    mpe = np.mean((forecast - actual)/actual)   # MPE
    rmse = np.mean((forecast - actual)**2)**.5  # RMSE
    corr = np.corrcoef(forecast, actual)[0,1]   # corr
    mins = np.amin(np.hstack([forecast[:,None],
                              actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None],
                              actual[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)             # minmax
    acfGCU = acf(fcGCU-testGCU)[1]                      # ACF1
    return({'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse, 'acfGCU':acfGCU,
            'corr':corr, 'minmax':minmax})

forecast_accuracy(fcGCU, testGCU.values)
```

```
{'mape': 0.2977565582824608,
 'me': -30.707790732531542,
 'mae': 30.730416503866188,
 'mpe': -0.2973627506394804,
 'rmse': 34.79483543107784,
 'acfGCU': 0.991196509716407,
 'corr': 0.8892695971648542,
 'minmax': 0.29775425058557503}
```

- GCU prediction <> what happened in reality
- Mean absolute percentage error: ~71% (mediocre)
- Root of mean squared error: 34.79/100 (large differences within the errors to predicted values)

- *Conclusion: Stock price prediction is difficult because it might not pan out to reality - data is non-stationary by nature and actuals are affected by outside variables other than historical data*

```
In [55]: input_feature = GCUniStock.iloc[:,0:2].values
         input_data = input_feature

In [56]: sc= MinMaxScaler(feature_range=(0,1))
         input_data[:,0:2] = sc.fit_transform(input_feature[:,:])

In [57]: plt.figure(figsize=(15,10))
         plt.subplots(1,sharex=True)
         plt.plot(input_feature[:,0], color='red')
         plt.title("Grand Canyon University Enrollment")
         plt.xlabel("Time (oldest --> latest)")
         plt.ylabel("Yearly Enrollment")
         plt.plot(input_feature[:,1], color='blue')
         plt.title("Grand Canyon University Stock Prices")
         plt.xlabel("Time (oldest --> latest)")
         plt.ylabel("Stock Closing Price")
         plt.show()

<Figure size 1080x720 with 0 Axes>
```



3) Can we take a multivariate approach towards time-series modeling to see if we can predict stock prices based on reported enrollment data?

- Knowing that enrollment data has a positive bias towards stock price, can we incorporate this into the prediction?
- Recurrent Neural Networks?
- Normalize the data; notice the trends of the graph moving positively together

```python
lookback= 50

test_size=int(.3 * len(GCUniStock))
xGC = []
yGC = []
for i in range(len(GCUniStock)-lookback-1):
    t1 = []
    for j in range(0,lookback):

        t1.append(input_data[[(i+j)], :])
    xGC.append(t1)
    yGC.append(input_data[i+ lookback,1])
```

```python
xGC, yGC= np.array(xGC), np.array(yGC)
xGC_test = xGC[:test_size+lookback]

xGC = xGC.reshape(xGC.shape[0],lookback, 2)
xGC_test = xGC_test.reshape(xGC_test.shape[0],lookback, 2)
print(xGC.shape)
print(xGC_test.shape)
```

```
(2217, 50, 2)
(730, 50, 2)
```

```python
modelGC = Sequential()
modelGC.add(LSTM(units=30, return_sequences= True, input_shape=(xGC.shape[1],2)))
modelGC.add(LSTM(units=30, return_sequences=True))
modelGC.add(LSTM(units=30))
modelGC.add(Dense(units=1))
modelGC.summary()
```

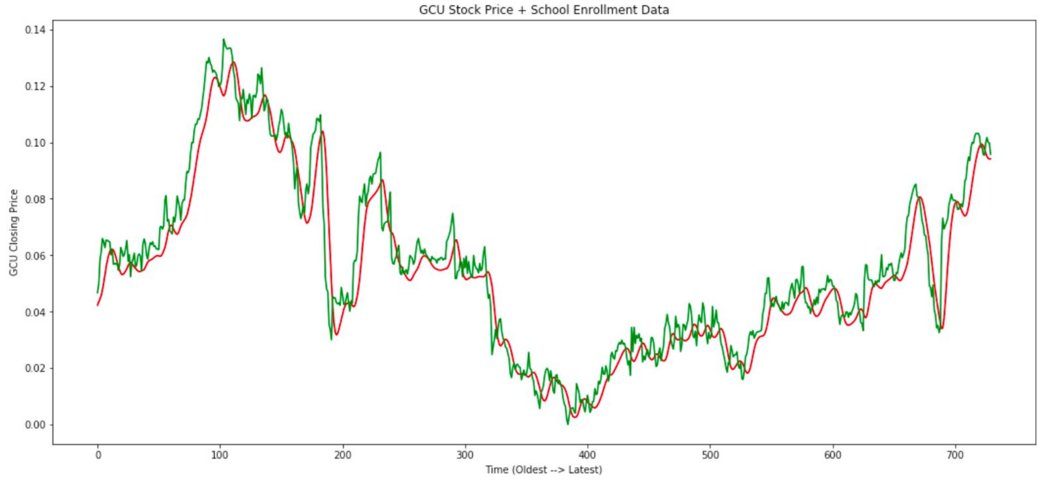
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_4 (LSTM)                (None, 50, 30)            3960      
_________________________________________________________________
lstm_5 (LSTM)                (None, 50, 30)            7320      
_________________________________________________________________
lstm_6 (LSTM)                (None, 30)                7320      
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 31        
=================================================================
Total params: 18,631
Trainable params: 18,631
Non-trainable params: 0
_________________________________________________________________
```

- *Long Short Term Models*: sequential data preserved in hidden cells (memory), absorbs data through new inputs, and makes predictions based on what the model knows to project forward
- Total parameters tell us that for # rows x 2 different data-types (test/training data), the model has created 18,000+ parameters/factors to consider when predicting the y_output (stock price)

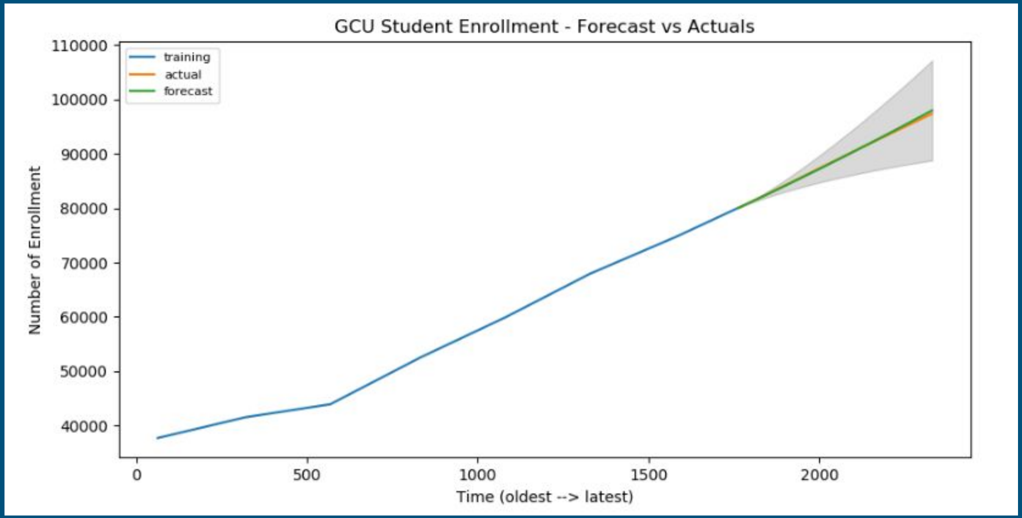
```
In [61]: modelGC.compile(optimizer='adam', loss='mean_squared_error')
         modelGC.fit(xGC, yGC, epochs=50, batch_size=32)

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from ter
Instructions for updating:
Use tf.cast instead.
Epoch 1/50
2217/2217 [==============================] - 6s 3ms/step - loss: 0.0134
Epoch 2/50
2217/2217 [==============================] - 3s 1ms/step - loss: 5.6194e-04
Epoch 3/50
2217/2217 [==============================] - 3s 1ms/step - loss: 4.4287e-04
Epoch 4/50
2217/2217 [==============================] - 3s 1ms/step - loss: 4.3730e-04
Epoch 5/50
2217/2217 [==============================] - 3s 1ms/step - loss: 4.3407e-04
```

```
GCpredicted_value= modelGC.predict(xGC_test)
```

```
plt.plot(GCpredicted_value, color= 'red')
plt.plot(input_data[lookback:test_size+(2*lookback),1], color='green')
plt.title("GCU Stock Price + School Enrollment Data")
plt.xlabel("Time (Oldest --> Latest)")
plt.ylabel("GCU Closing Price")
plt.show()
```



- Adam = adaptive learning rate method similar to principles of gradient descent - ability to learn/interpret data for different parameters
- Predicted value and the test data (700 days) seems to work within accuracy



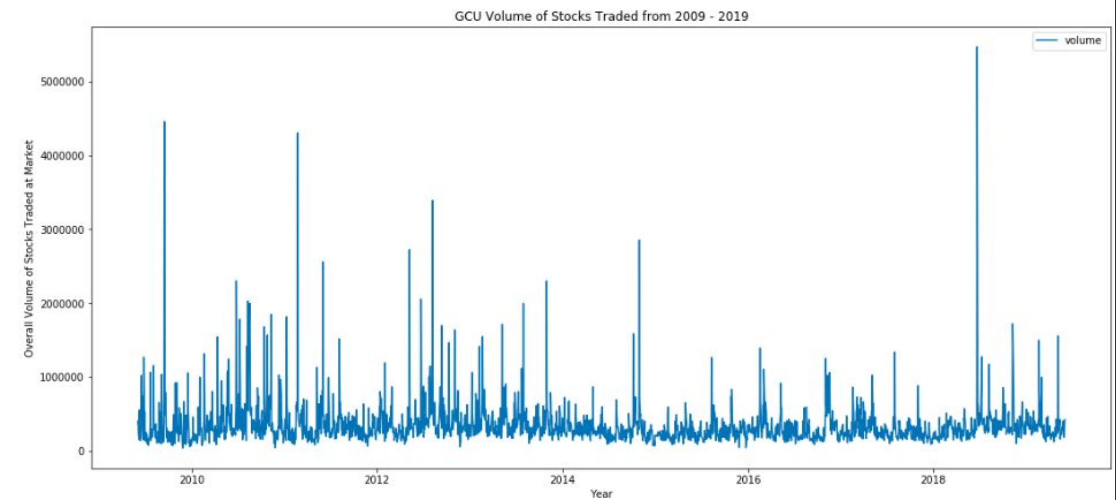

```
EnrollARIMA = pm.auto_arima(GCUEnrollARIMA.Enrollment, start_p=1, start_q=1,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)
print(EnrollARIMA.summary())
```

```
Fit ARIMA: order=(1, 2, 1); AIC=nan, BIC=nan, Fit time=nan seconds
Fit ARIMA: order=(0, 2, 0); AIC=3501.620, BIC=3513.071, Fit time=0.002 seconds
Fit ARIMA: order=(1, 2, 0); AIC=3503.620, BIC=3520.797, Fit time=0.025 seconds
Fit ARIMA: order=(0, 2, 1); AIC=3503.620, BIC=3520.797, Fit time=0.016 seconds
Total fit time: 0.139 seconds
                             ARIMA Model Results
==============================================================================
Dep. Variable:                   D2.y   No. Observations:                 2266
Model:                 ARIMA(0, 2, 0)   Log Likelihood               -1748.810
Method:                           css   S.D. of innovations              0.524
Date:                Tue, 25 Jun 2019   AIC                           3501.620
Time:                        10:53:29   BIC                           3513.071
Sample:                             2   HQIC                          3505.798

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0067      0.011      0.607      0.544      -0.015       0.028
==============================================================================
```

4) So what?

- What if you could use ARIMA modeling to predict student enrollment and use LSTM modeling to predict stock prices?

- What if you could predict any outside feature that influences stock prices and create reasonable predictions using recurrent neural networks?



5) Conclusion..

- Stock price prediction is extremely difficult because of investor sentiments are subjective and event-driven

- But multivariable approaches towards time-series prediction is worthwhile product financial or strategy analysts can use for both investment and institutional companies