RNN_EURUSD_update_2020_03_09

March 12, 2020

[4]:
```
# Import libraries
# LSTM for EURUSD prices from
# https://finance.yahoo.com/quote/EURUSD%3DX/history?
 ↪period1=1070236800&period2=1583366400&interval=1mo&filter=history&frequency=1mo
# Data is on my GitHub and will be downloaded in the next step
import numpy
import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import numpy as np
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import mplfinance as mpf
%matplotlib notebook
```

```
Using TensorFlow backend.
```

[5]:
```
url = 'https://raw.githubusercontent.com/DataScientist2807/RNN/master/EURUSD.
 ↪csv'
df = pd.read_csv(url, error_bad_lines=False)
```

[6]:
```
# Length of dataset is 148. We have 148 prices for Open, High, Low and Close
len(df)
```

[6]: 148

[7]:
```
# We first have a look at the data with head and tail commands
df.head()
```

[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2008-01-01	1.460110	1.559284	1.437298	1.486503	1.486503	0
1	2008-02-01	1.486591	1.557099	1.445191	1.519203	1.519203	0
2	2008-03-01	1.518395	1.590306	0.072902	1.575796	1.575796	0

```
3  2008-04-01  1.561695  1.601307  1.551711  1.562207   1.562207       0
4  2008-05-01  1.547796  1.581803  1.537090  1.555791   1.555791       0
```

[8]:
```
# The last date look different then the others. It is the date when I
 ↪programmed this.
df.tail()
```

[8]:
```
           Date      Open      High       Low     Close  Adj Close  Volume
143  2019-12-01  1.101910  1.124101  1.100376  1.120230   1.120230       0
144  2020-01-01  1.122083  1.122838  1.099324  1.102913   1.102913       0
145  2020-02-01  1.109609  1.109609  1.077958  1.103000   1.103000       0
146  2020-03-01  1.102809  1.120750  1.102809  1.114405   1.114405       0
147  2020-03-05  1.113586  1.122083  1.112471  1.120448   1.120448       0
```

[9]:
```
# We will drop it to be consistent. That every first day of each month is the
 ↪baseline.
df = df[:-1]
df.tail()
```

[9]:
```
           Date      Open      High       Low     Close  Adj Close  Volume
142  2019-11-01  1.115611  1.119445  1.098286  1.102000   1.102000       0
143  2019-12-01  1.101910  1.124101  1.100376  1.120230   1.120230       0
144  2020-01-01  1.122083  1.122838  1.099324  1.102913   1.102913       0
145  2020-02-01  1.109609  1.109609  1.077958  1.103000   1.103000       0
146  2020-03-01  1.102809  1.120750  1.102809  1.114405   1.114405       0
```

Visualization with pyplot

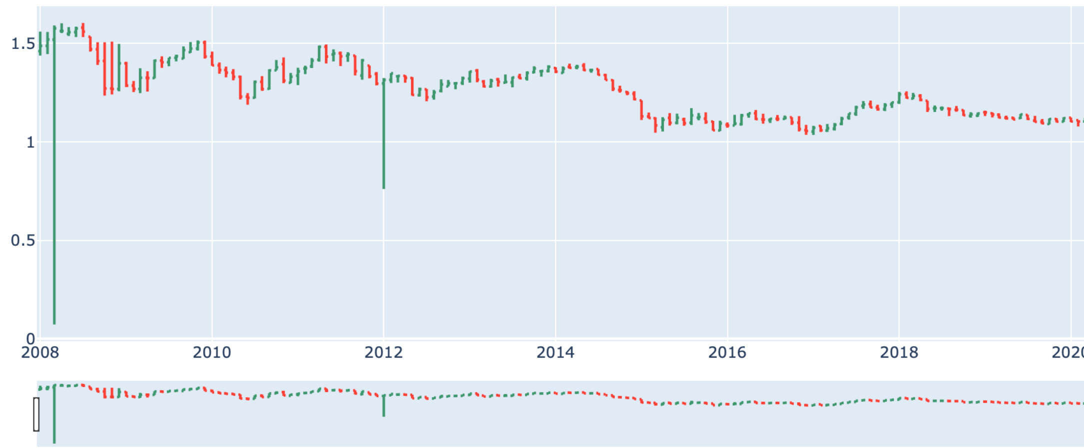
[10]:
```
# Types for dataset looks fine although Date shouldn't be an object and more
 ↪datetime type
df.dtypes
```

[10]:
```
Date           object
Open          float64
High          float64
Low           float64
Close         float64
Adj Close     float64
Volume          int64
dtype: object
```

[11]:
```
fig = go.Figure(data=go.Ohlc(x=df['Date'],
                    open=df['Open'],
                    high=df['High'],
                    low=df['Low'],
                    close=df['Close']))
fig.show()
```

[12]:
```
from IPython.display import Image
PATH = "/Users/marcelbruckmann/"
Image(filename = PATH + "Plot_EURUSD1.png", width=1000, height=300)
```

[12]:



[13]:
```
# We see some prices here who are not fitting in the whole picture.
# Luckily the plot is quite innovative and when we do a mouse-over we can see␣
↪the date and prices
# We probably could automize to change prices of outlieres but this we won't do␣
↪here (only 2 prices to change)
# We will change the prices manually. More specifically we will take the new␣
↪price as the price one month before
```

Remove price outliers

[14]:
```
# Outliers in March 2018 (Low) and Jan 2012 (Low)
print(df.loc[df['Date'] == "2008-03-01"])
print(df.loc[df['Date'] == "2012-01-01"])
```

```
         Date      Open      High       Low     Close  Adj Close  Volume
2  2008-03-01  1.518395  1.590306  0.072902  1.575796   1.575796       0
          Date      Open      High       Low     Close  Adj Close  Volume
48  2012-01-01  1.296092  1.323399  0.760572  1.313957   1.313957       0
```

[15]:
```
# Clearly we see both outliers in Low price with 0.072902 and 0.760572
```

[16]:
```
# Let's get the prices for previous month for both outliers
price_outlier1 = df[df.Date == "2008-02-01"]["Low"].values[0]
price_outlier2 = df[df.Date == "2011-12-01"]["Low"].values[0]
print("Previous price for outlier 1 is " + str(price_outlier1) + " and for␣
↪outlier 2 is " + str(price_outlier2))
```

```
Previous price for outlier 1 is 1.445191 and for outlier 2 is 1.286124
```

[17]:
```
# We will change now both prices to previous prices
df.loc[df['Date'] == "2008-03-01", 'Low'] = price_outlier1
```

```
df.loc[df['Date'] == "2012-01-01", 'Low'] = price_outlier2
```

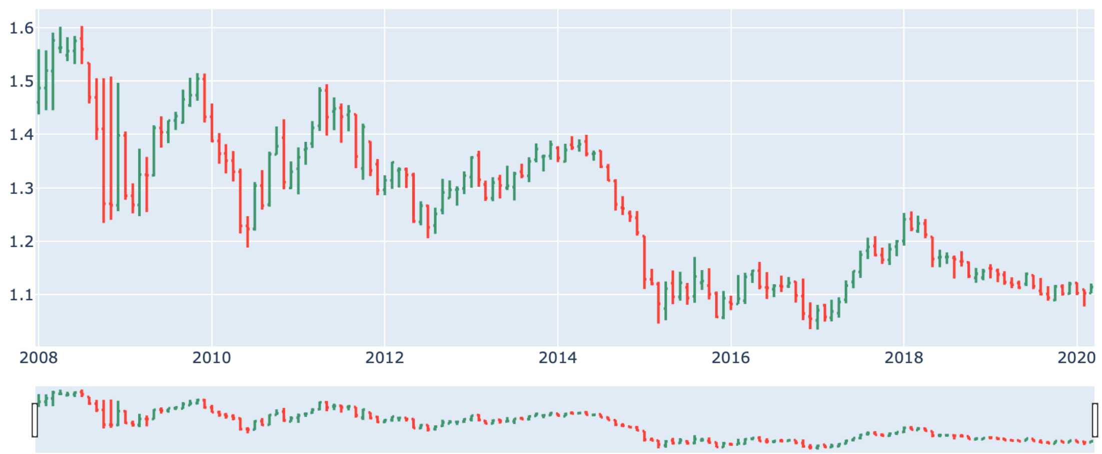
[18]:
```
# Outliers in March 2018 (Low) and Jan 2012 (Low)
print(df.loc[df['Date'] == "2008-03-01"])
print(df.loc[df['Date'] == "2012-01-01"])
```

```
         Date      Open      High       Low     Close  Adj Close  Volume
2  2008-03-01  1.518395  1.590306  1.445191  1.575796   1.575796       0
          Date      Open      High       Low     Close  Adj Close  Volume
48  2012-01-01  1.296092  1.323399  1.286124  1.313957   1.313957       0
```

[19]:
```
# Looks good, let's check the candlestick chart
fig = go.Figure(data=go.Ohlc(x=df['Date'],
                    open=df['Open'],
                    high=df['High'],
                    low=df['Low'],
                    close=df['Close']))
fig.show()
```

[20]:
```
from IPython.display import Image
PATH = "/Users/marcelbruckmann/"
Image(filename = PATH + "Plot_EURUSD2.png", width=1000, height=300)
```

[20]:



[21]:
```
# Looks much better right? As we can see both outliers are eliminated and we␣
↪can continue.
```

[22]:
```
# One important thing before we start is to set the seed for reproducibility
numpy.random.seed(1234)
```

[23]:
```
# We will use the four columns of price state and put them into different␣
↪dataframes
df_open = df[["Open"]]
```

```
df_high = df[["High"]]
df_low = df[["Low"]]
df_close = df[["Close"]]
# Example
df_high.head()
```

[23]:

```
       High
0  1.559284
1  1.557099
2  1.590306
3  1.601307
4  1.581803
```

[24]:

```
# Our algorithm needs to understand all these values hence we transform them␣
 ↪into values or floats to be specifically.
# Although we can see from above that it is already a type float jupyter␣
 ↪notebook is not showing '' by default.
```

[25]:

```
df_open, df_high, df_low, df_close  = df_open.values, df_high.values, df_low.
 ↪values, df_close.values
df_open, df_high, df_low, df_close = df_open.astype('float32'), df_high.
 ↪astype('float32'), df_low.astype('float32'), df_close.astype('float32')
```

[26]:

```
# Next we will normalize the data
sc = MinMaxScaler(feature_range=(0, 1))
df_open, df_high, df_low, df_close = sc.fit_transform(df_open), sc.
 ↪fit_transform(df_high), sc.fit_transform(df_low), sc.fit_transform(df_close)
```

[27]:

```
# Split data into trainset and testset
# Hence all prices have the same lenth we only have to write the size for␣
 ↪trainset and testset once
train_size = int(len(df_open) * 2/3)
test_size = len(df_open) - train_size
# Now the split:
train_open, test_open = df_open[0:train_size,:], df_open[train_size:
 ↪len(df_open),:]
train_high, test_high = df_high[0:train_size,:], df_high[train_size:
 ↪len(df_high),:]
train_low, test_low = df_low[0:train_size,:], df_low[train_size:len(df_low),:]
train_close, test_close = df_close[0:train_size,:], df_close[train_size:
 ↪len(df_close),:]
```

[28]:

```
# Convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
```

```
    return numpy.array(dataX), numpy.array(dataY)
```

[29]:
```
# Reshape into X=t and Y=t+1
look_back = 1
trainX_open, trainY_open = create_dataset(train_open, look_back)
testX_open, testY_open = create_dataset(test_open, look_back)
trainX_high, trainY_high = create_dataset(train_high, look_back)
testX_high, testY_high = create_dataset(test_high, look_back)
trainX_low, trainY_low = create_dataset(train_low, look_back)
testX_low, testY_low = create_dataset(test_low, look_back)
trainX_close, trainY_close = create_dataset(train_close, look_back)
testX_close, testY_close = create_dataset(test_close, look_back)
```

[30]:
```
# reshape input to be [samples, time steps, features]
trainX_open = numpy.reshape(trainX_open, (trainX_open.shape[0], 1, trainX_open.
 ↪shape[1]))
testX_open = numpy.reshape(testX_open, (testX_open.shape[0], 1, testX_open.
 ↪shape[1]))
trainX_high = numpy.reshape(trainX_high, (trainX_high.shape[0], 1, trainX_high.
 ↪shape[1]))
testX_high = numpy.reshape(testX_high, (testX_high.shape[0], 1, testX_high.
 ↪shape[1]))
trainX_low = numpy.reshape(trainX_low, (trainX_low.shape[0], 1, trainX_low.
 ↪shape[1]))
testX_low = numpy.reshape(testX_low, (testX_low.shape[0], 1, testX_low.
 ↪shape[1]))
trainX_close = numpy.reshape(trainX_close, (trainX_close.shape[0], 1,␣
 ↪trainX_close.shape[1]))
testX_close = numpy.reshape(testX_close, (testX_close.shape[0], 1, testX_close.
 ↪shape[1]))
```

Define LSTM

[31]:
```
def get_LSTM(trainX, trainY, epochsval, batchsize, verboseval):
    # Create and fit the LSTM
    model = Sequential()
    model.add(LSTM(4, input_shape=(1, look_back)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(trainX, trainY, epochs=epochsval, batch_size=batchsize,␣
 ↪verbose=verboseval)
    return model
```

LSTM Prediction for Open Price

[32]:
```
scoreOpen = get_LSTM(trainX_open, trainY_open, 10, 1, 2)
```

```
WARNING:tensorflow:From /Users/marcelbruckmann/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
```

```
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /Users/marcelbruckmann/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/10
 - 0s - loss: 0.1926
Epoch 2/10
 - 0s - loss: 0.0851
Epoch 3/10
 - 0s - loss: 0.0381
Epoch 4/10
 - 0s - loss: 0.0256
Epoch 5/10
 - 0s - loss: 0.0227
Epoch 6/10
 - 0s - loss: 0.0213
Epoch 7/10
 - 0s - loss: 0.0201
Epoch 8/10
 - 0s - loss: 0.0187
Epoch 9/10
 - 0s - loss: 0.0175
Epoch 10/10
 - 0s - loss: 0.0165
```

LSTM Prediction for High Price

[33]:
```
scoreHigh = get_LSTM(trainX_high, trainY_high, 10, 1, 2)
```

```
Epoch 1/10
 - 0s - loss: 0.2892
Epoch 2/10
 - 0s - loss: 0.1552
Epoch 3/10
 - 0s - loss: 0.0739
Epoch 4/10
 - 0s - loss: 0.0419
Epoch 5/10
 - 0s - loss: 0.0339
Epoch 6/10
 - 0s - loss: 0.0315
Epoch 7/10
 - 0s - loss: 0.0295
Epoch 8/10
```

```
 - 0s - loss: 0.0273
Epoch 9/10
 - 0s - loss: 0.0252
Epoch 10/10
 - 0s - loss: 0.0233
```

LSTM Prediction for Low Price

[34]:
```python
scoreLow = get_LSTM(trainX_low, trainY_low, 10, 1, 2)
```

```
Epoch 1/10
 - 0s - loss: 0.3098
Epoch 2/10
 - 0s - loss: 0.1835
Epoch 3/10
 - 0s - loss: 0.1032
Epoch 4/10
 - 0s - loss: 0.0590
Epoch 5/10
 - 0s - loss: 0.0432
Epoch 6/10
 - 0s - loss: 0.0392
Epoch 7/10
 - 0s - loss: 0.0371
Epoch 8/10
 - 0s - loss: 0.0351
Epoch 9/10
 - 0s - loss: 0.0335
Epoch 10/10
 - 0s - loss: 0.0319
```

LSTM Prediction for Close Price

[35]:
```python
scoreClose = get_LSTM(trainX_close, trainY_close, 10, 1, 2)
```

```
Epoch 1/10
 - 0s - loss: 0.2919
Epoch 2/10
 - 0s - loss: 0.1704
Epoch 3/10
 - 0s - loss: 0.0921
Epoch 4/10
 - 0s - loss: 0.0522
Epoch 5/10
 - 0s - loss: 0.0403
Epoch 6/10
 - 0s - loss: 0.0369
Epoch 7/10
 - 0s - loss: 0.0350
```

```
Epoch 8/10
 - 0s - loss: 0.0331
Epoch 9/10
 - 0s - loss: 0.0314
Epoch 10/10
 - 0s - loss: 0.0296
```

Predictions Open, High, Low, Close

[36]:
```
trainPredictOpen = scoreOpen.predict(trainX_open)
testPredictOpen = scoreOpen.predict(testX_open)
trainPredictHigh = scoreHigh.predict(trainX_high)
testPredictHigh = scoreHigh.predict(testX_high)
trainPredictLow = scoreLow.predict(trainX_low)
testPredictLow = scoreLow.predict(testX_low)
trainPredictClose = scoreClose.predict(trainX_close)
testPredictClose = scoreClose.predict(testX_close)
```

[37]:
```
def invpred(price):
    val = sc.inverse_transform(price)
    return val
```

Invert prediction values Open Price

[38]:
```
trainPredictOpen = sc.inverse_transform(trainPredictOpen)
trainYOpen = sc.inverse_transform([trainY_open])
testPredictOpen = sc.inverse_transform(testPredictOpen)
testYOpen = sc.inverse_transform([testY_open])
```

Invert prediction values High Price

[39]:
```
trainPredictHigh = sc.inverse_transform(trainPredictHigh)
trainYHigh = sc.inverse_transform([trainY_high])
testPredictHigh = sc.inverse_transform(testPredictHigh)
testYHigh = sc.inverse_transform([testY_high])
```

Invert prediction values Low Price

[40]:
```
trainPredictLow = sc.inverse_transform(trainPredictLow)
trainYLow = sc.inverse_transform([trainY_low])
testPredictLow = sc.inverse_transform(testPredictLow)
testYLow = sc.inverse_transform([testY_low])
```

Invert prediction values Close Price

[41]:
```
trainPredictClose = sc.inverse_transform(trainPredictClose)
trainYClose = sc.inverse_transform([trainY_close])
testPredictClose = sc.inverse_transform(testPredictClose)
testYClose = sc.inverse_transform([testY_close])
```

Invert prediction values Close Price

[42]:
```
def score_RMSE(trainY, trainPredict):
    return math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
```

Show RMSE results

[43]:
```
report = pd.DataFrame(
{"Train": [format(score_RMSE(trainYOpen, trainPredictOpen), '.2f') ,␣
 ↪format(score_RMSE(trainYHigh, trainPredictHigh), '.2f'),␣
 ↪format(score_RMSE(trainYLow, trainPredictLow), '.2f'),␣
 ↪format(score_RMSE(trainYClose, trainPredictClose), '.2f')],
"Test": [format(score_RMSE(testYOpen, testPredictOpen), '.2f'),␣
 ↪format(score_RMSE(testYHigh, testPredictHigh), '.2f'),␣
 ↪format(score_RMSE(testYLow, testPredictLow), '.2f'),␣
 ↪format(score_RMSE(testYClose, testPredictClose), '.2f') ]},
index = ["Open", "High", "Low", "Close"])
print(report)
```

```
      Train  Test
Open   0.07  0.10
High   0.08  0.14
Low    0.09  0.15
Close  0.09  0.15
```

[44]:
```
# Shift train predictions for plotting
trainPredictPlotOpen = numpy.empty_like(df_open)
trainPredictPlotOpen[:, :] = numpy.nan
trainPredictPlotOpen[look_back:len(trainPredictOpen)+look_back, :] =␣
 ↪trainPredictOpen

trainPredictPlotHigh = numpy.empty_like(df_high)
trainPredictPlotHigh[:, :] = numpy.nan
trainPredictPlotHigh[look_back:len(trainPredictHigh)+look_back, :] =␣
 ↪trainPredictHigh

trainPredictPlotLow = numpy.empty_like(df_low)
trainPredictPlotLow[:, :] = numpy.nan
trainPredictPlotLow[look_back:len(trainPredictLow)+look_back, :] =␣
 ↪trainPredictLow

trainPredictPlotClose = numpy.empty_like(df_close)
trainPredictPlotClose[:, :] = numpy.nan
trainPredictPlotClose[look_back:len(trainPredictClose)+look_back, :] =␣
 ↪trainPredictClose

# Shift test predictions for plotting
testPredictPlotOpen = numpy.empty_like(df_open)
testPredictPlotOpen[:, :] = numpy.nan
testPredictPlotOpen[len(trainPredictOpen)+(look_back*2)+1:len(df_open)-1, :] =␣
 ↪testPredictOpen

testPredictPlotHigh = numpy.empty_like(df_high)
testPredictPlotHigh[:, :] = numpy.nan
```

```
testPredictPlotHigh[len(trainPredictHigh)+(look_back*2)+1:len(df_high)-1, :] =␣
 ↪testPredictHigh

testPredictPlotLow = numpy.empty_like(df_low)
testPredictPlotLow[:, :] = numpy.nan
testPredictPlotLow[len(trainPredictLow)+(look_back*2)+1:len(df_low)-1, :] =␣
 ↪testPredictLow

testPredictPlotClose = numpy.empty_like(df_close)
testPredictPlotClose[:, :] = numpy.nan
testPredictPlotClose[len(trainPredictClose)+(look_back*2)+1:len(df_close)-1, :]␣
 ↪= testPredictClose
```

[45]:
```
# plot baseline and predictions
import datetime
import matplotlib.dates as mdates
fig, ax = plt.subplots(figsize=(18,10))
ax.plot(sc.inverse_transform(df_open)) # Real Open Price
ax.plot(trainPredictPlotOpen) # Train Open Price
ax.plot(testPredictPlotHigh, 'r-')
ax.plot(testPredictPlotClose, 'b-')
ax.plot(testPredictPlotOpen, 'k-')
ax.plot(testPredictPlotLow, 'g-')
x1 = np.linspace(2,140,50)
y1 = -0.0030*x1+1.6
ax.plot(x1, y1)

x2 = np.linspace(2,140,50)
y2 = -0.0030*x2+1.3
ax.plot(x2, y2)

x3 = np.linspace(2,140,50)
y3 = -0.0010*x3+1.25
ax.plot(x3, y3)
fig.show()
```

```
/Users/marcelbruckmann/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:22: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is
a non-GUI backend, so cannot show the figure.
```

Conclusion

We first found two outliers who are really far from the rices surrounded by them. This can be an error by yahoo.com or some big news announced like non-farm payrolls by Federal Reserve. I leave it up to people who read this to check the influences of the outliers. We've set up the LSTM Model to make our predictions on OHLC (Open-High-Low-Close) price. Further we've calculated RMSE (Root Mean Suared Error). Finally we've drawn the nice plot. A bit strange is that the arrangement of the colors. E.g. The black color represents the predicted open price and green color the low price prediction. This cannot be. To have some rough predictions under risk probably we can use it. We've also drawn some lines for trend. To get familiar with the topic I would recommend the reader to look up this if it is not understandable. But basically the lines showing the channel where the price could move into the future. We do not have a channel only more also a triangle shown. This triangle usually becomes closer, means the momentum is getting slower. A trader would wait for an outbreak of one of these lines. After breaking a line, the line becomes a resistance line and it is then unlikely that the price moves back. Actually in our plot the price moved back. This is called noise. The price is currently not yet broken and there is a chance that price moves further downwards.

THANK YOU FOR READING THIS! I HOPE YOU FOUND IT VALUABLE FOR YOU OR YOUR BUSINESS

[46]:
```
# For every half year we will take the highest and lowest points
# We have data from 2008 to 2020.
df.head()
```

[46]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2008-01-01	1.460110	1.559284	1.437298	1.486503	1.486503	0
1	2008-02-01	1.486591	1.557099	1.445191	1.519203	1.519203	0
2	2008-03-01	1.518395	1.590306	1.445191	1.575796	1.575796	0
3	2008-04-01	1.561695	1.601307	1.551711	1.562207	1.562207	0
4	2008-05-01	1.547796	1.581803	1.537090	1.555791	1.555791	0

[47]:
```
dfCopy = df.copy()
```

[48]:
```
# We will add feature Month to the dataset. We need to import datetime first␣
↪and convert Date to Datetime
from datetime import datetime
dfCopy['Date'] = pd.to_datetime(dfCopy['Date'])
dfCopy.head()
```

[48]:
```
        Date      Open      High       Low     Close  Adj Close  Volume
0 2008-01-01  1.460110  1.559284  1.437298  1.486503   1.486503       0
1 2008-02-01  1.486591  1.557099  1.445191  1.519203   1.519203       0
2 2008-03-01  1.518395  1.590306  1.445191  1.575796   1.575796       0
3 2008-04-01  1.561695  1.601307  1.551711  1.562207   1.562207       0
4 2008-05-01  1.547796  1.581803  1.537090  1.555791   1.555791       0
```

[49]:
```
dfCopy.dtypes
```

[49]:
```
Date         datetime64[ns]
Open                float64
High                float64
Low                 float64
Close               float64
Adj Close           float64
Volume                int64
dtype: object
```

[50]:
```
dfCopy["Month"] = np.nan
dfCopy["Month"] = dfCopy["Date"].dt.month
dfCopy["Year"] = np.nan
dfCopy["Year"] = dfCopy["Date"].dt.year
dfCopy.head()
```

[50]:
```
        Date      Open      High       Low     Close  Adj Close  Volume  \
0 2008-01-01  1.460110  1.559284  1.437298  1.486503   1.486503       0
1 2008-02-01  1.486591  1.557099  1.445191  1.519203   1.519203       0
2 2008-03-01  1.518395  1.590306  1.445191  1.575796   1.575796       0
3 2008-04-01  1.561695  1.601307  1.551711  1.562207   1.562207       0
4 2008-05-01  1.547796  1.581803  1.537090  1.555791   1.555791       0

   Month  Year
0      1  2008
1      2  2008
2      3  2008
3      4  2008
4      5  2008
```

[51]:
```
dfCopy["Year"] = dfCopy["Year"].astype(str)
dfCopy["Month"] = dfCopy["Month"].astype(str)
dfCopy["Y/M"] = dfCopy[['Year', 'Month']].apply(lambda x: '-'.join(x), axis=1)
```

[52]:
```
dfCopy.head()
```

[52]:
```
         Date      Open      High       Low     Close  Adj Close  Volume Month  \
0 2008-01-01  1.460110  1.559284  1.437298  1.486503   1.486503       0      1
1 2008-02-01  1.486591  1.557099  1.445191  1.519203   1.519203       0      2
2 2008-03-01  1.518395  1.590306  1.445191  1.575796   1.575796       0      3
3 2008-04-01  1.561695  1.601307  1.551711  1.562207   1.562207       0      4
4 2008-05-01  1.547796  1.581803  1.537090  1.555791   1.555791       0      5

   Year     Y/M
0  2008  2008-1
1  2008  2008-2
2  2008  2008-3
3  2008  2008-4
4  2008  2008-5
```

[53]:
```python
# We take the first and the sencond highest prices of the year and group our␣
↪dataframe:
dfH = pd.DataFrame(data=dfCopy.groupby('Year')['Close'].apply(lambda grp: grp.
↪nlargest(2)))
dfH = dfH.reset_index()
dfH.head()
```

[53]:
```
   Year  level_1     Close
0  2008        2  1.575796
1  2008        5  1.575002
2  2009       22  1.503895
3  2009       21  1.473297
4  2010       24  1.387694
```

[54]:
```python
# We take the first and the sencond lowest prices of the year and group our␣
↪dataframe:
dfL = pd.DataFrame(data=dfCopy.groupby('Year')['Close'].apply(lambda grp: grp.
↪nsmallest(2)))
dfL = dfL.reset_index()
dfL.head()
```

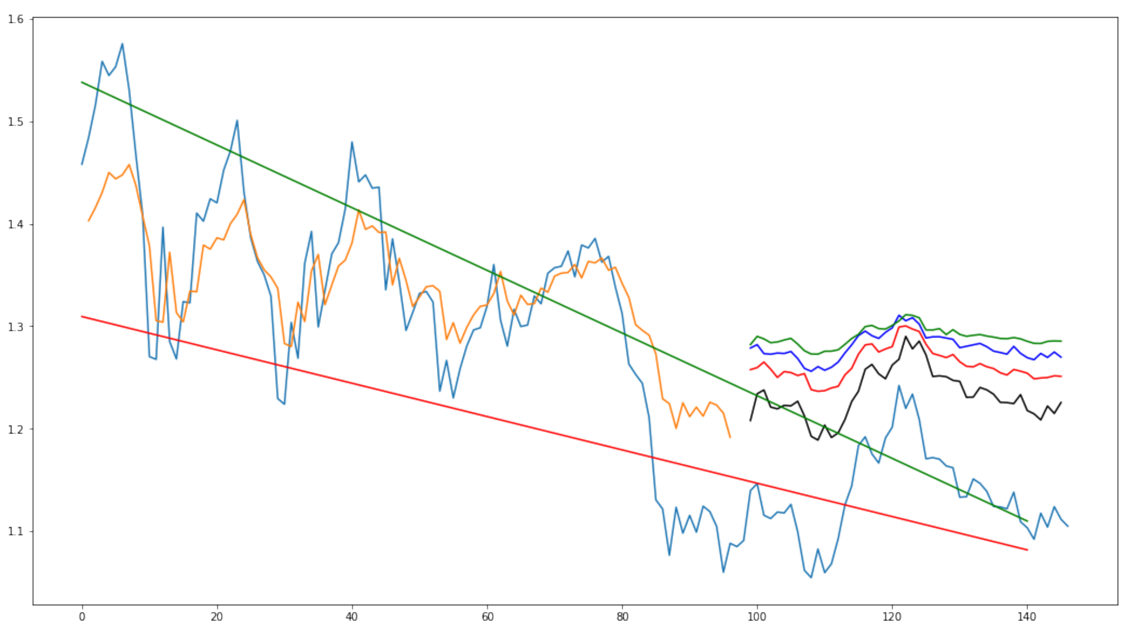
[54]:
```
   Year  level_1     Close
0  2008       10  1.267507
1  2008        9  1.270196
2  2009       13  1.267893
3  2009       12  1.285099
4  2010       29  1.223002
```

[55]:
```python
dfL_ind = dfL["level_1"].values
dfL_price = dfL["Close"].values
dfH_ind = dfH["level_1"].values
dfH_price = dfH["Close"].values
```

[56]:
```python
# plot baseline and predictions
import datetime
import matplotlib.dates as mdates
```

```
fig, ax = plt.subplots(figsize=(18,10))
ax.plot(sc.inverse_transform(df_open)) # Real Open Price
ax.plot(trainPredictPlotOpen) # Train Open Price
ax.plot(testPredictPlotHigh, 'r-')
ax.plot(testPredictPlotClose, 'b-')
ax.plot(testPredictPlotOpen, 'k-')
ax.plot(testPredictPlotLow, 'g-')

coef = np.polyfit(dfL_ind, dfL_price, 1)
equ = np.poly1d(coef)
x_plot = np.linspace(0,140,148)
y_plot = equ(x_plot)
plt.plot(x_plot, y_plot, color='r')

coef = np.polyfit(dfH_ind, dfH_price, 1)
equ = np.poly1d(coef)
x_plot = np.linspace(0,140,148)
y_plot = equ(x_plot)
plt.plot(x_plot, y_plot, color='g')

fig.show()
```

```
/Users/marcelbruckmann/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:24: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is
a non-GUI backend, so cannot show the figure.
```



Conclusion

In the first part we manually tried to draw our trend lines. For the second part we added some features and grouped our data to get highest and lowest prices for each year. Actually we get 2 prices for the highest and 2 for the lowest per year. We used polyfit and poly1d from numpy to translate our points into a plot. The plot looks different then the previous one. We see that our green line is not reaching the peeks of the highest prices. For the red line it looks a bit better. Good news is, that we've created a triangle and both lines are focussing more to the direction of the current PRICE! Although we didn't catch all highs and lows (above and under can be noise) we are pretty close to the current price for EURUSD.

THANK YOU!

[57]:
```
# FIBONACCI RETRACEMENT
```

[58]:
```
# Fom previous exercise we use the highest close and the lowest close
```

[89]:
```
highestClose = np.max(dfCopy[["Close"]].values)
highestClose
```

[89]:
```
1.575796
```

[90]:
```
lowestClose = np.min(dfCopy[["Close"]].values)
lowestClose
```

[90]:
```
1.054741
```

[101]:
```
fib1 = .236
fib2 = .382
fib3 = .50
fib4 = .618
```

[105]:
```
# Calculate prices for each retracement
fib1_line = lowestClose + ((highestClose - lowestClose) * fib1)
fib2_line = lowestClose + ((highestClose - lowestClose) * fib2)
fib3_line = lowestClose + ((highestClose - lowestClose) * fib3)
fib4_line = lowestClose + ((highestClose - lowestClose) * fib4)
fib4_line
```

[105]:
```
1.37675299
```

[157]:
```
# plot baseline and predictions
import datetime
import matplotlib.dates as mdates
from matplotlib import colors as mcolors
fig, ax = plt.subplots(figsize=(18,10))
ax.plot(sc.inverse_transform(df_open)) # Real Open Price
ax.plot(trainPredictPlotOpen) # Train Open Price
ax.plot(testPredictPlotHigh, 'r-')
ax.plot(testPredictPlotClose, 'b-')
ax.plot(testPredictPlotOpen, 'k-')
ax.plot(testPredictPlotLow, 'g-')
```

```
coef = np.polyfit(dfL_ind, dfL_price, 1)
equ = np.poly1d(coef)
x_plot = np.linspace(0,140,148)
y_plot = equ(x_plot)
plt.plot(x_plot, y_plot, color='r')

coef = np.polyfit(dfH_ind, dfH_price, 1)
equ = np.poly1d(coef)
x_plot = np.linspace(0,140,148)
y_plot = equ(x_plot)
plt.plot(x_plot, y_plot, color='g')

# FIBONACCI
alpha_value = 0.4

fib0x = np.linspace(100,140,40)
fib0y = 0.00*fib0x+lowestClose
ax.plot(fib0x, fib0y, 'b--', alpha=alpha_value)

##############################################################################
fib1x = np.linspace(100,140,40)
fib1y = 0.00*fib1x+fib1_line
ax.plot(fib1x, fib1y, 'b--', alpha=alpha_value)
ax.fill_between(fib0x, lowestClose, fib1y, alpha=alpha_value, color='#fed0fc')
ax.text(100, lowestClose + 0.01, "0%")

##############################################################################
fib2x = np.linspace(100,140,40)
fib2y = 0.00*fib2x+fib2_line
ax.plot(fib2x, fib2y, 'b--', alpha=alpha_value)
ax.fill_between(fib1x, fib1_line, fib2y, alpha=alpha_value, color='#c5c9c7')
ax.text(100, fib1_line + 0.01, "26.6%")

##############################################################################
fib3x = np.linspace(100,140,40)
fib3y = 0.00*fib3x+fib3_line
ax.plot(fib3x, fib3y, 'b--', alpha=alpha_value)
ax.fill_between(fib2x, fib2_line, fib3y, alpha=alpha_value, color='#bdf6f3')
ax.text(100, fib2_line + 0.01, "38.2%")

##############################################################################
fib4x = np.linspace(100,140,40)
fib4y = 0.00*fib4x+fib4_line
ax.plot(fib4x, fib4y, 'b--', alpha=alpha_value)

ax.fill_between(fib3x, fib3_line, fib4y, alpha=alpha_value, color='#b2fba5')
```

```python
ax.text(100, fib3_line + 0.01, "50%")
################################################################################

fib5x = np.linspace(0,140,40)
fib5y = 0.00*fib5x+highestClose
ax.plot(fib5x, fib5y, 'b--', alpha=alpha_value)

ax.fill_between(fib4x, fib4_line, highestClose, alpha=alpha_value,␣
 ↪color='#fd798f')
ax.text(100, highestClose - 0.01, "100%")
ax.text(100, fib4_line + 0.01, "61.8%")



fig.show()
```

```
/Users/marcelbruckmann/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:72: UserWarning:

Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is
a non-GUI backend, so cannot show the figure.
```

Conclusion

We have drawn the FIBONACCI lines into the graph. When we have a look at the green, blue and grey area then we have a high proability that price will move into those areas (up to 61.8 % chance). Our prediction is actually going into the blue area. Unlikely is that price moves to green area but if the trend is turning in the opposite direction and is building new trends then this can become feasible. One point at last. The current price is in the pink area and if the price only touches the price of 1.054741 (our lowest close price) then the chance is higher for going in opposite direction.