main

February 25, 2018

1 Appendix B: GBP/USD Exchange Rate during 2017 UK Election Night

In [186]:
```
import numpy as np
import csv
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets
import statsmodels.api as sm
from statsmodels.regression.linear_model import RegressionResults
import random

import pandas as pd
from collections import OrderedDict
from datetime import date

from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pylab

import tweepy

import json

import re
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
```

In [187]:
```
CONSUMER_KEY = '4BByuBKYk19fpSl5iMIkju3c0'
CONSUMER_SECRET = '2EK91aTOs7uMJ1oWECBRUwkXrxGykigrsmtqtOIAvFBPXiucQq'
ACCESS_TOKEN = '892729320736739328-E30nIY5dacqxeugxPoe3TXB2fIjITZB'
ACCESS_TOKEN_SECRET = 'WMViNA7y1d1trkb5nt7L5dOAHdScmYBMHm33sLeUVZrWT'


auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth)
```

```
        tweets_raw = api.user_timeline(screen_name = 'bbcelection', count = 200, include_rts
        for i in range(0,5):
            oldest=tweets_raw[-1].id
            new_tweets = api.user_timeline(screen_name = 'bbcelection',count=200,max_id=oldes
            tweets_raw.extend(new_tweets)

        data = [[tw.created_at.year, tw.created_at.month, tw.created_at.day,"%s.%s"%(tw.creat
        tweets=pd.DataFrame(data, columns=['year','month','date','time','tweet_id','tweet'])
        tweets = tweets[tweets.year==2017]
```

```
In [188]: # Wikipedia data
        UKpoll = pd.read_csv('data/UK2017Poll.txt', sep='\t', header=0)
        UKpoll.columns=['ID','Con_poll', 'Lab_poll', 'Lib_poll','SNP_poll','Pla_poll','Greens
                        'UKIP_poll', 'Other_poll', 'Seat','Region','2015']
        results = pd.read_csv("data/result.csv")
```

```
In [189]: tweets.tweet=tweets.tweet.astype(str)
        tweets_cleaned = tweets[tweets.tweet.str.contains('#GE2017')]
        tweets_cleaned['time_full'] = tweets_cleaned["year"].map(str)+ "/"+ tweets_cleaned["m
            "/"+ tweets_cleaned["date"].map(str)  + "/"+ tweets_cleaned["time"].map(str)
```

1.1 Merging result data with time

```
In [190]: tweets_cleaned['Constituency']=np.nan
        for i in range(len(tweets)):
            tweets_cleaned.Constituency[i] = tweets_cleaned.tweet[i][tweets_cleaned.tweet[i]
        tweets_cleaned.to_csv('data/tweets.csv',sep=',')
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\AlexH\Anaconda3\lib\site-packages\pandas\core\indexing.py:179: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self._setitem_with_indexer(indexer, value)
```

```
In [191]: #manual matching
        tw_matched = pd.read_csv('data/tweets_matched.csv')
        tw_matched = tw_matched.dropna(axis=0, how='any')
        tw_matched = tw_matched.drop_duplicates(subset='ID', keep='last')
        tw_matched.index = np.arange(len(tw_matched))
        tw_matched.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 642 entries, 0 to 641
```

```
Data columns (total 9 columns):
year            642 non-null int64
month           642 non-null int64
date            642 non-null int64
time            642 non-null float64
tweet_id        642 non-null float64
tweet           642 non-null object
time_full       642 non-null object
Constituency    642 non-null object
ID              642 non-null object
dtypes: float64(2), int64(3), object(4)
memory usage: 50.2+ KB
```

```
In [192]: results = results.merge( tw_matched[['ID','time_full']], how='left', left_on = 'ID',
          results
          results[['Con[b]','Lab[c]','LD','SNP','UKIP','Grn[d]','DUP']]=(results[['Con[b]','Lab
          results['time']=np.nan
          for i in range(0,len(results)):
              try:
                  results['time'].loc[i]= datetime.strptime(results.time_full[i],'%Y/%m/%d/%H.%
              except:
                  results['time'].loc[i] = np.nan
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\pandas\core\indexing.py:179: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  self._setitem_with_indexer(indexer, value)
```

1.2 Merging poll with result

```
In [193]: full_table = pd.merge(results, UKpoll,  how='left', left_on='ID', right_on = 'ID')
          full_table = full_table.sort_values(by='time').reset_index(drop=True)
```

1.3 Other factors

taken from UK sensus

```
In [194]:     #    {nan,
              #'West Midlands',
              # 'Scotland',
              # 'South East',
              # 'Yorkshire and The Humber',
              # 'Wales',
              # 'South West',
              # 'East of England',
              # 'London',
```

```
            # 'North West',
            # 'North East',
            # 'East Midlands'}

        xl = pd.ExcelFile("data/Wages.xlsx")
        wages=xl.parse("Data")
        wages.head()
        full_table = pd.merge(full_table, wages[['ONSConstID','WageMedianConst']],  how='left

        xl = pd.ExcelFile("data/Business-numbers.xlsx")
        business=xl.parse("Data")
        #full_table = pd.merge(full_table, business,  how='left', left_on='Constituency', ri

        xl = pd.ExcelFile("data/Population-by-age.xlsx")
        population=xl.parse("Data")
        full_table = pd.merge(full_table, population[['ONSConstID','Pop65ConstRate']],  how=

        full_table['islab'] = (full_table['Last Election']=='Lab').astype(int)
        full_table['iscon'] = (full_table['Last Election']=='Con').astype(int)
        full_table['islib'] = (full_table['Last Election']=='LD').astype(int)
        full_table['issnp'] = (full_table['Last Election']=='SNP').astype(int)
        full_table['london'] = (full_table['Region']=='London').astype(int)
        full_table['Turnout'] = [float(v.rstrip("%")) for v in full_table.turnout]
        full_table['iswales'] = (full_table['Region']=='Wales').astype(int)
        full_table['isscot'] = (full_table['Region']=='Scotland').astype(int)
```

In [ ]:

2 Analysis

2.1 Single Factor

```
In [195]: columns = ['time', 'Constituency','low', 'ave', 'high']
          index= full_table.index
          pred=pd.DataFrame(index=index, columns=columns)

          parameters_con = np.empty((650,2))
          stderr_con = np.empty((650,2))
          parameters_lab = np.empty((650,2))
          stderr_lab = np.empty((650,2))
          parameters_snp = np.empty((650,2))
          stderr_snp = np.empty((650,2))

          sim_num = 50
          for i in range(18,len(full_table)-14):

              #labour regression
             # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
```

```
data_lab= pd.concat([full_table['Lab_poll'],\
                     full_table['Lab[c]']], axis=1)
#,full_table['Con_poll']*full_table['iscon'],
X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
Y_lab=data_lab['Lab[c]'][0:i]
X_lab=sm.add_constant(X_lab, has_constant='add')
model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

parameters_lab[i,:]=model_lab.params
stderr_lab[i,:]=model_lab.bse

#Conservatives regression
#data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
data_con= pd.concat([full_table['Con_poll'],\
                     full_table['Con[b]']], axis=1)
X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
Y_con=data_con['Con[b]'][0:i]
X_con=sm.add_constant(X_con, has_constant='add')
model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

parameters_con[i,:]=model_con.params
stderr_con[i,:]=model_con.bse


data_snp=pd.concat([full_table['SNP_poll'], full_table['SNP']], axis=1)
X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0




predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant='ad
predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
result = np.zeros([sim_num, len(predict_lab)])

#Sampling the regression parameters to generate predicted outcome
for j in range(0, sim_num):

    #Labour
    param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), Regr
    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=
```

```
#fill NaN with mean
#predict_lab = predict_lab.fillna(predict_lab.mean())
predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
#result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resid)-
result_lab = np.random.normal(result_temp,err_dev)


#Conservative
param_con = np.random.multivariate_normal(np.asarray(model_con.params), Regr
predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_cons


#predict_con = predict_con.fillna(predict_con.mean())
predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

result_temp = np.matmul(param_con,np.array(predict_con).T)
#result_con = np.random.normal(result_temp,np.std(model_con.resid))
err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resid)-
result_con = np.random.normal(result_temp,err_dev)




#SNP
predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_constant
predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
if not model_snp==0 and len(model_snp.resid)>8:
    param_snp = np.random.multivariate_normal(np.asarray(model_snp.params), M

    #predict_snp = predict_snp.fillna(predict_snp.mean())

    result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
    #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
    err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp.resi
    result_snp = np.random.normal(result_temp,err_dev)
    #if the poll is 0, the resulting simulated data should also be 0
    ind=np.where(predict_snp.SNP_poll==0)
    result_snp[:,ind]=0

else:
    #no data
    result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100
```

```
            #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,1)))
            #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_num,1)
            other = 1-result_con-result_lab-result_snp

            result_temp = (result_con>result_lab) & (result_con>result_snp) & (result_co
            result = np.append(result, result_temp,axis=0)

        #remove the zeros during initialisation
        result = np.delete(result,range(0,sim_num),0)
        result = result.astype(int)

        #filling the result of intermediates states with current count
#       select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.isnul

        EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(result
#       print(jointb['time'][i],jointb['State'][i], EEV)

        #make table for plot
        pred.time[i]=full_table.time[i]
        pred.Constituency[i]=full_table.ID[i]
        pred.low[i]=np.percentile(EEV,10)
        pred.ave[i]=np.mean(EEV)
        pred.high[i]=np.percentile(EEV,90)

    # keep only the last value at a certain time. And remove NAs.
    pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
    pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

2.1.1 Reading Financial Data

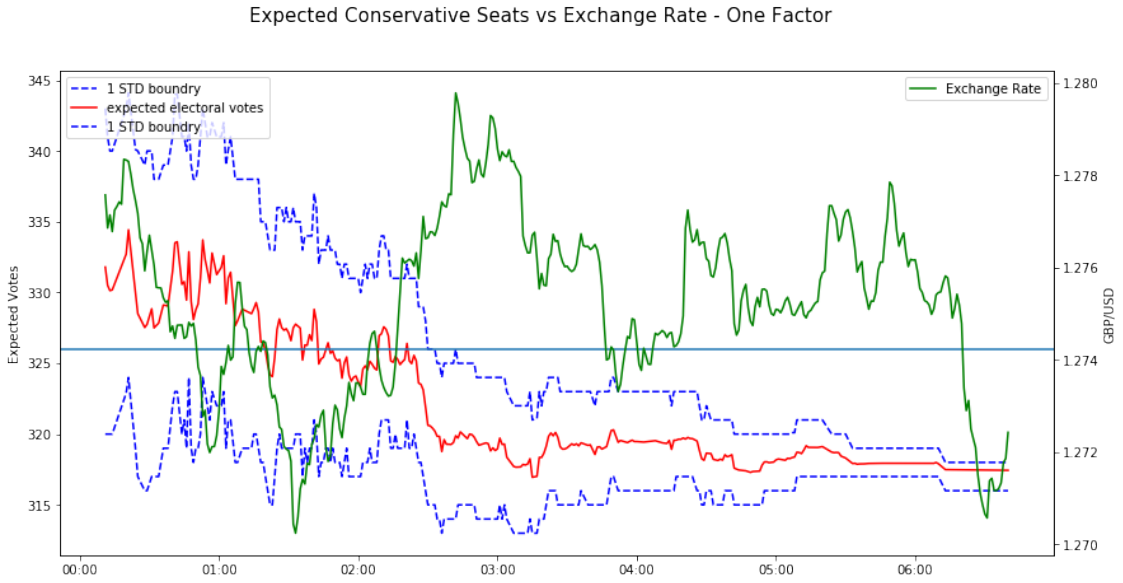
```
In [196]: fx = pd.read_csv('data/FX data.csv',header=None,sep='\;')

    #the sixth column is 0, drop it
    fx = fx.drop(fx.columns[5], 1)

    # average the minute values.
    fx['mean'] = fx.ix[:,2:4].astype(float).mean(axis=1)

    # extract time value to datetime format
    fx['time'] = fx.ix[:,0]
    fx['time'] = [datetime.strptime(v, '%Y%m%d %H%M%S') for v in fx['time']]

    #change time to UTC to match twitter
```

```
            fx['time'] = [v + timedelta(hours=5) for v in fx['time']]

            fx = fx.set_index(['time'])
            fx = fx.loc[pred.time[0]:pred.time[len(pred)-1]]
            fx = fx.drop(fx.columns[0:5],1)
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: ParserWarning: Falling back
```

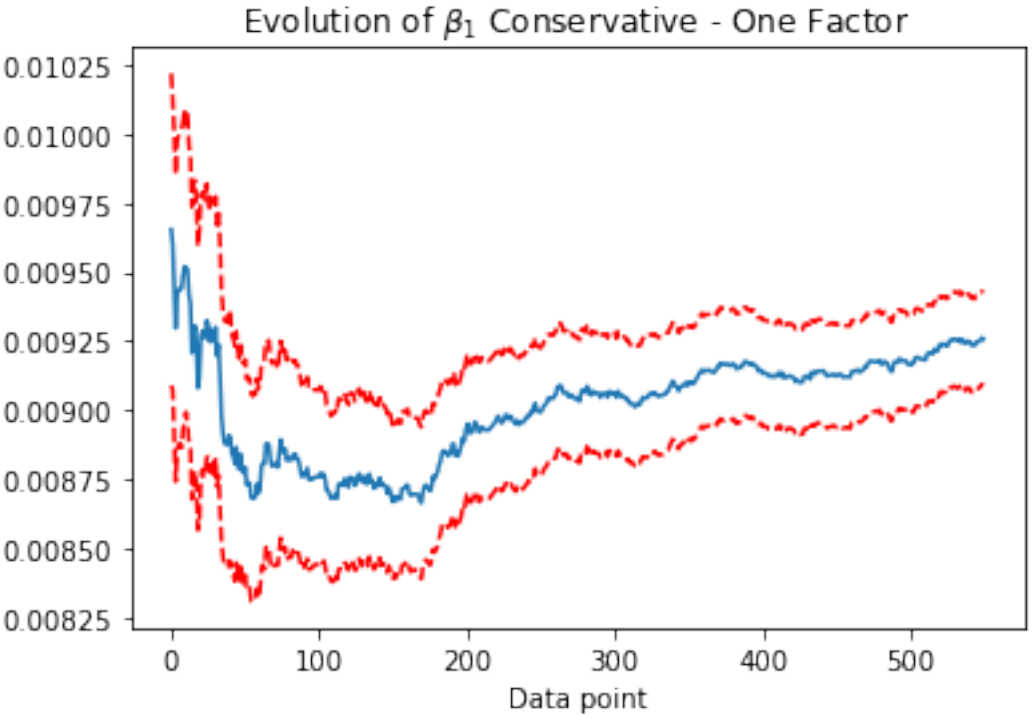
```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate_ix
```

```
In [197]: fig, ax1 = plt.subplots()
          fig.set_size_inches(14, 7)
          ax1.plot_date(pred.time, pred.low,'b--', label='1 STD boundry')
          ax1.plot_date(pred.time, pred.ave,'r-', label='expected electoral votes')
          ax1.plot_date(pred.time, pred.high,'b--', label='1 STD boundry')
          plt.ylabel('Expected Votes')
          pylab.legend(loc='upper left')
          plt.axhline(y=326)
          #ax1.set_ylim([300,350])

          ax2 = ax1.twinx()
          ax2.plot(fx['mean'],'g', label='Exchange Rate')

          HMFmt = mdates.DateFormatter('%H:%M')
          ax1.xaxis.set_major_formatter(HMFmt)
          _ = plt.xticks(rotation=90)

          plt.ylabel('GBP/USD')
          plt.xlabel('UTC Time')
          pylab.legend(loc='upper right')

          fig.suptitle('Expected Conservative Seats vs Exchange Rate - One Factor', fontsize=15
          fig.savefig('results-onefac.png')
          plt.show()
```




```
In [198]: plt.plot(parameters_lab[50:600,1])
          plt.plot(parameters_lab[50:600,1]-stderr_lab[50:600,1],'r--')
          plt.plot(parameters_lab[50:600,1]+stderr_lab[50:600,1],'r--')
          plt.title(r'Evolution of $\beta_1 $ Labour - One Factor')
          plt.xlabel('Data point')
          plt.savefig('lab one factor b1.png')
          plt.show()

          plt.plot(parameters_con[50:600,1])
          plt.plot(parameters_con[50:600,1]-stderr_con[50:600,1],'r--')
          plt.plot(parameters_con[50:600,1]+stderr_con[50:600,1],'r--')
          plt.title(r'Evolution of $\beta_1 $ Conservative - One Factor')
          plt.xlabel('Data point')
          plt.savefig('con one factor b1.png')
          plt.show()
```


Evolution of β_1 Labour - One Factor
0.0122
0.0120
0.0118
0.0116
0.0114
0.0112
0.0110
0.0108
0.0106
0
100
200
300
400
500
Data point


Evolution of β_1 Conservative - One Factor
0.01025
0.01000
0.00975
0.00950
0.00925
0.00900
0.00875
0.00850
0.00825
0
100
200
300
400
500
Data point

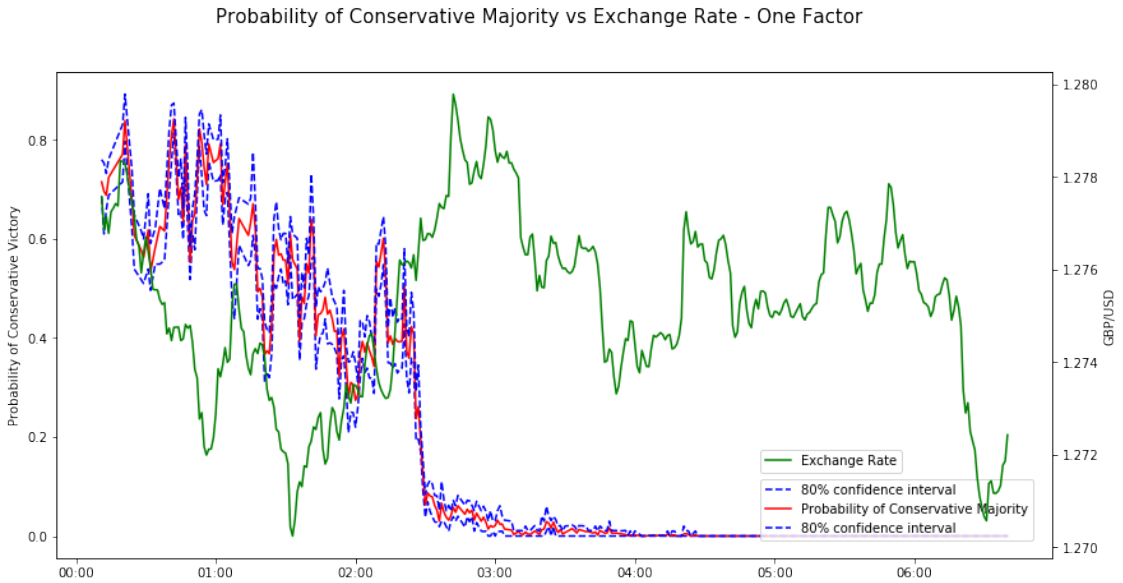
```
In [199]: columns = ['time', 'Constituency','low', 'ave', 'high']
          index= full_table.index
          pred=pd.DataFrame(index=index, columns=columns)

          parameters_con = np.empty((650,2))
          stderr_con = np.empty((650,2))
          parameters_lab = np.empty((650,2))
          stderr_lab = np.empty((650,2))
          parameters_snp = np.empty((650,2))
          stderr_snp = np.empty((650,2))

          sim_num = 10
          for i in range(18,len(full_table)-14):
              #labour regression
             # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
              data_lab= pd.concat([full_table['Lab_poll'],\
                                   full_table['Lab[c]']], axis=1)
              #,full_table['Con_poll']*full_table['iscon'],
              X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
              Y_lab=data_lab['Lab[c]'][0:i]
              X_lab=sm.add_constant(X_lab, has_constant='add')
              model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

              parameters_lab[i,:]=model_lab.params
              stderr_lab[i,:]=model_lab.bse

              #Conservatives regression
              #data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
              data_con= pd.concat([full_table['Con_poll'],\
                                   full_table['Con[b]']], axis=1)
              X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
              Y_con=data_con['Con[b]'][0:i]
              X_con=sm.add_constant(X_con, has_constant='add')
              model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

              parameters_con[i,:]=model_con.params
              stderr_con[i,:]=model_con.bse

              data_snp=pd.concat([full_table['SNP_poll'], full_table['SNP']], axis=1)
              X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
              Y_snp=data_snp['SNP'][0:i]
              X_snp=sm.add_constant(X_snp, has_constant='add')
              try:
                  model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
                  parameters_snp[i,:]
                  stderr_snp[i,:]=model_snp.bse
              except:
                  model_snp = 0
```

```
prob = np.zeros(10)
for k in range(0,10):

    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
    result = np.zeros([sim_num, len(predict_lab)])

    #Sampling the regression parameters to generate predicted outcome
    for j in range(0, sim_num):

        #Labour
        param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), M
        predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_const


        #fill NaN with mean
        #predict_lab = predict_lab.fillna(predict_lab.mean())
        predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

        result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
        #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
        err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resi
        result_lab = np.random.normal(result_temp,err_dev)


        #Conservative
        param_con = np.random.multivariate_normal(np.asarray(model_con.params), M
        predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_


        #predict_con = predict_con.fillna(predict_con.mean())
        predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

        result_temp = np.matmul(param_con,np.array(predict_con).T)
        #result_con = np.random.normal(result_temp,np.std(model_con.resid))
        err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resi
        result_con = np.random.normal(result_temp,err_dev)



        #SNP
        predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_cons
        predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
        if not model_snp==0 and len(model_snp.resid)>8:
            param_snp = np.random.multivariate_normal(np.asarray(model_snp.params
```

```
                    #predict_snp = predict_snp.fillna(predict_snp.mean())

                    result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
                    #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
                    err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp
                    result_snp = np.random.normal(result_temp,err_dev)
                    #if the poll is 0, the resulting simulated data should also be 0
                    ind=np.where(predict_snp.SNP_poll==0)
                    result_snp[:,ind]=0

                else:
                    #no data
                    result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

                #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,
                #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_nu
                other = 1-result_con-result_lab-result_snp

                result_temp = (result_con>result_lab) & (result_con>result_snp) & (result
                result = np.append(result, result_temp,axis=0)

            #remove the zeros during initialisation
            result = np.delete(result,range(0,sim_num),0)
            result = result.astype(int)

            #filling the result of intermediates states with current count
        #    select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.i

            EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(res
        #    print(jointb['time'][i],jointb['State'][i], EEV)

            prob[k] = sum(EEV>326)/len(EEV)


        #make table for plot
        pred.time[i]=full_table.time[i]
        pred.Constituency[i]=full_table.ID[i]
        pred.low[i]=np.percentile(prob,10)
        pred.ave[i]=np.mean(prob)
        pred.high[i]=np.percentile(prob,90)

    # keep only the last value at a certain time. And remove NAs.
    pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
    pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

In [200]:
```
#%%
fig, ax1 = plt.subplots()
fig.set_size_inches(14, 7)
ax1.plot_date(pred.time, pred.low,'b--', label='80% confidence interval')
ax1.plot_date(pred.time, pred.ave,'r-', label='Probability of Conservative Majority')
ax1.plot_date(pred.time, pred.high,'b--', label='80% confidence interval')
plt.ylabel('Probability of Conservative Victory')
#pylab.legend(loc='lower right')
#plt.axhline(y=270)
ax1.legend(loc='center left', bbox_to_anchor=(0.7, 0.1))
#ax1.set_ylim([0.5,0.9])

ax2 = ax1.twinx()
ax2.plot(fx['mean'],'g', label='Exchange Rate')
#ax2.set_ylim([18.5,21.5])

HMFmt = mdates.DateFormatter('%H:%M')
ax1.xaxis.set_major_formatter(HMFmt)
_ = plt.xticks(rotation=90)

plt.ylabel('GBP/USD')

#pylab.legend(loc='upper right')
ax2.legend(loc='center left', bbox_to_anchor=(0.7, 0.2))

fig.suptitle('Probability of Conservative Majority vs Exchange Rate - One Factor', fo

#plt.figure(figsize=(20,10))
plt.show()
fig.savefig('prob-onefac.png')
```



2.2 3 factor

```
In [201]: columns = ['time', 'Constituency','low', 'ave', 'high']
          index= full_table.index
          pred=pd.DataFrame(index=index, columns=columns)

          parameters_con = np.empty((650,4))
          stderr_con = np.empty((650,4))
          parameters_lab = np.empty((650,4))
          stderr_lab = np.empty((650,4))
          parameters_snp = np.empty((650,4))
          stderr_snp = np.empty((650,4))

          sim_num = 50
          for i in range(18,len(full_table)-14):

              #labour regression
             # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
              data_lab= pd.concat([full_table['Lab_poll'],\
                                   full_table['WageMedianConst'] , \
                                   full_table['Pop65ConstRate'] ,\
                                    full_table['Lab[c]']], axis=1)
              #,full_table['Con_poll']*full_table['iscon'],
              X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
              Y_lab=data_lab['Lab[c]'][0:i]
              X_lab=sm.add_constant(X_lab, has_constant='add')
              model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

              parameters_lab[i,:]=model_lab.params
              stderr_lab[i,:]=model_lab.bse

              #Conservatives regression
              #data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
              data_con= pd.concat([full_table['Con_poll'],\
                                   full_table['WageMedianConst'],  \
                                   full_table['Pop65ConstRate'] ,\
                                   full_table['Con[b]']], axis=1)
              X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
              Y_con=data_con['Con[b]'][0:i]
              X_con=sm.add_constant(X_con, has_constant='add')
              model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

              parameters_con[i,:]=model_con.params
              stderr_con[i,:]=model_con.bse
```

```
data_snp=pd.concat([full_table['SNP_poll'],  full_table['WageMedianConst'],
                    full_table['Pop65ConstRate'], full_table['SNP']], axis=1)
X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0
```

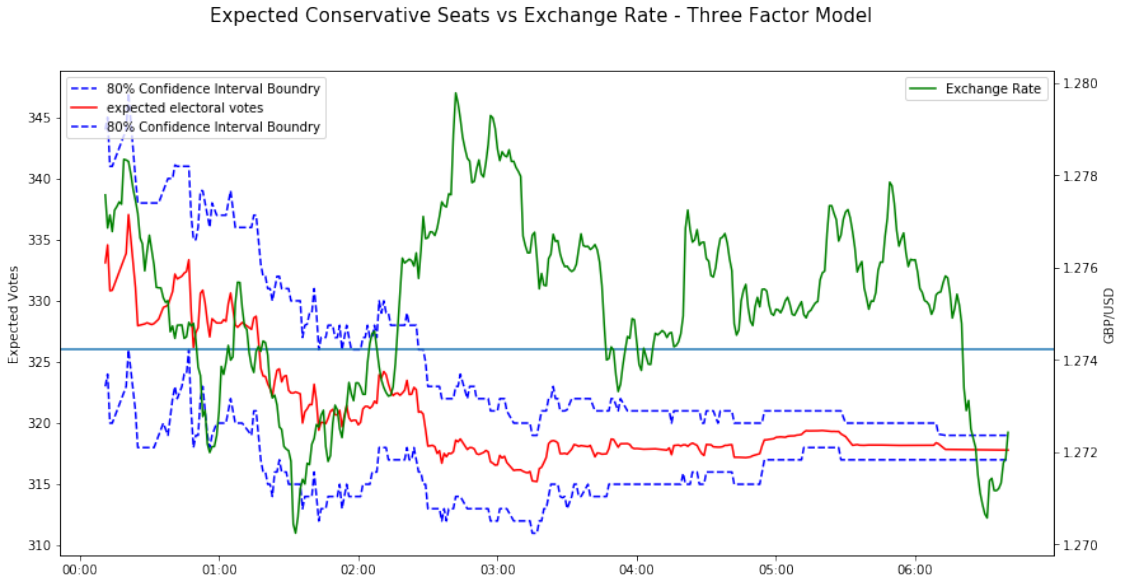
```
predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant='ad
predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
result = np.zeros([sim_num, len(predict_lab)])

#Sampling the regression parameters to generate predicted outcome
for j in range(0, sim_num):

    #Labour
    param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), Regr
    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant


    #fill NaN with mean
    #predict_lab = predict_lab.fillna(predict_lab.mean())
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

    result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
    #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
    err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resid)-
    result_lab = np.random.normal(result_temp,err_dev)


    #Conservative
    param_con = np.random.multivariate_normal(np.asarray(model_con.params), Regr
    predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_cons


    #predict_con = predict_con.fillna(predict_con.mean())
    predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

    result_temp = np.matmul(param_con,np.array(predict_con).T)
    #result_con = np.random.normal(result_temp,np.std(model_con.resid))
```

```
        err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resid)-4
        result_con = np.random.normal(result_temp,err_dev)




        #SNP
        predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_constant
        predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
        if not model_snp==0 and len(model_snp.resid)>8:
            param_snp = np.random.multivariate_normal(np.asarray(model_snp.params), M

            #predict_snp = predict_snp.fillna(predict_snp.mean())

            result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
            #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
            err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp.resi
            result_snp = np.random.normal(result_temp,err_dev)
            #if the poll is 0, the resulting simulated data should also be 0
            ind=np.where(predict_snp.SNP_poll==0)
            result_snp[:,ind]=0

        else:
            #no data
            result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

        other = 1-result_con-result_lab-result_snp

        result_temp = (result_con>result_lab) & (result_con>result_snp) & (result_con
        result = np.append(result, result_temp,axis=0)

    #remove the zeros during initialisation
    result = np.delete(result,range(0,sim_num),0)
    result = result.astype(int)

    #filling the result of intermediates states with current count
#    select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.isnul

    EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(result
#    print(jointb['time'][i],jointb['State'][i], EEV)

    #make table for plot
    pred.time[i]=full_table.time[i]
    pred.Constituency[i]=full_table.ID[i]
    pred.low[i]=np.percentile(EEV,10)
    pred.ave[i]=np.mean(EEV)
    pred.high[i]=np.percentile(EEV,90)
```

```
        # keep only the last value at a certain time. And remove NAs.
        pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
        pred.index = np.arange(0,len(pred))
```

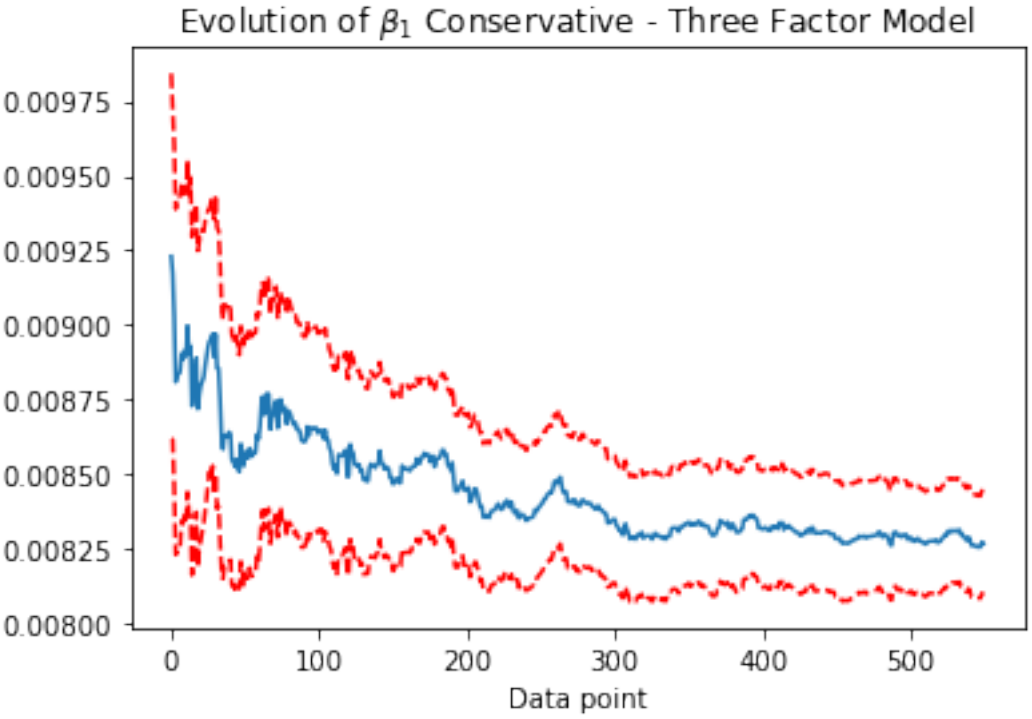
```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:140: RuntimeWarning: invalid v
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

In [202]:
```
fig, ax1 = plt.subplots()
fig.set_size_inches(14, 7)
ax1.plot_date(pred.time, pred.low,'b--', label='80% Confidence Interval Boundry')
ax1.plot_date(pred.time, pred.ave,'r-', label='expected electoral votes')
ax1.plot_date(pred.time, pred.high,'b--', label='80% Confidence Interval Boundry')
plt.ylabel('Expected Votes')
pylab.legend(loc='upper left')
plt.axhline(y=326)
#ax1.set_ylim([280,380])

ax2 = ax1.twinx()
ax2.plot(fx['mean'],'g', label='Exchange Rate')

HMFmt = mdates.DateFormatter('%H:%M')
ax1.xaxis.set_major_formatter(HMFmt)
_ = plt.xticks(rotation=90)

plt.ylabel('GBP/USD')
plt.xlabel('UTC Time')
pylab.legend(loc='upper right')

fig.suptitle('Expected Conservative Seats vs Exchange Rate - Three Factor Model', fo
fig.savefig('results-threefac.png')
plt.show()
```




```
In [203]: plt.plot(parameters_lab[50:600,1])
          plt.plot(parameters_lab[50:600,1]-stderr_lab[50:600,1],'r--')
          plt.plot(parameters_lab[50:600,1]+stderr_lab[50:600,1],'r--')
          plt.title(r'Evolution of $\beta_1 $ Labour - Three Factor Model')
          plt.xlabel('Data point')
          plt.savefig('lab threefac b1.png')
          plt.show()

          plt.plot(parameters_con[50:600,1])
          plt.plot(parameters_con[50:600,1]-stderr_con[50:600,1],'r--')
          plt.plot(parameters_con[50:600,1]+stderr_con[50:600,1],'r--')
          plt.title(r'Evolution of $\beta_1 $ Conservative - Three Factor Model')
          plt.xlabel('Data point')
          plt.savefig('con treefac b1.png')
          plt.show()
```


Evolution of β_1 Labour - Three Factor Model
0.0116
0.0114
0.0112
0.0110
0.0108
0.0106
0.0104
0.0102
0
100
200
300
400
500
Data point


Evolution of β_1 Conservative - Three Factor Model
0.00975
0.00950
0.00925
0.00900
0.00875
0.00850
0.00825
0.00800
0
100
200
300
400
500
Data point

In [204]:
```
columns = ['time', 'Constituency','low', 'ave', 'high']
index= full_table.index
pred=pd.DataFrame(index=index, columns=columns)

parameters_con = np.empty((650,4))
stderr_con = np.empty((650,4))
parameters_lab = np.empty((650,4))
stderr_lab = np.empty((650,4))
parameters_snp = np.empty((650,4))
stderr_snp = np.empty((650,4))

sim_num = 10
for i in range(18,len(full_table)-14):


    #labour regression
   # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
    data_lab= pd.concat([full_table['Lab_poll'],\
                         full_table['WageMedianConst'] , \
                         full_table['Pop65ConstRate'] ,\
                          full_table['Lab[c]']], axis=1)
    #,full_table['Con_poll']*full_table['iscon'],
    X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
    Y_lab=data_lab['Lab[c]'][0:i]
    X_lab=sm.add_constant(X_lab, has_constant='add')
    model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

    parameters_lab[i,:]=model_lab.params
    stderr_lab[i,:]=model_lab.bse

    #Conservatives regression
    #data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
    data_con= pd.concat([full_table['Con_poll'],\
                         full_table['WageMedianConst'],  \
                         full_table['Pop65ConstRate'] ,\
                         full_table['Con[b]']], axis=1)
    X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
    Y_con=data_con['Con[b]'][0:i]
    X_con=sm.add_constant(X_con, has_constant='add')
    model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

    parameters_con[i,:]=model_con.params
    stderr_con[i,:]=model_con.bse


    data_snp=pd.concat([full_table['SNP_poll'],  full_table['WageMedianConst'],
                        full_table['Pop65ConstRate'], full_table['SNP']], axis=1)
    X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
```

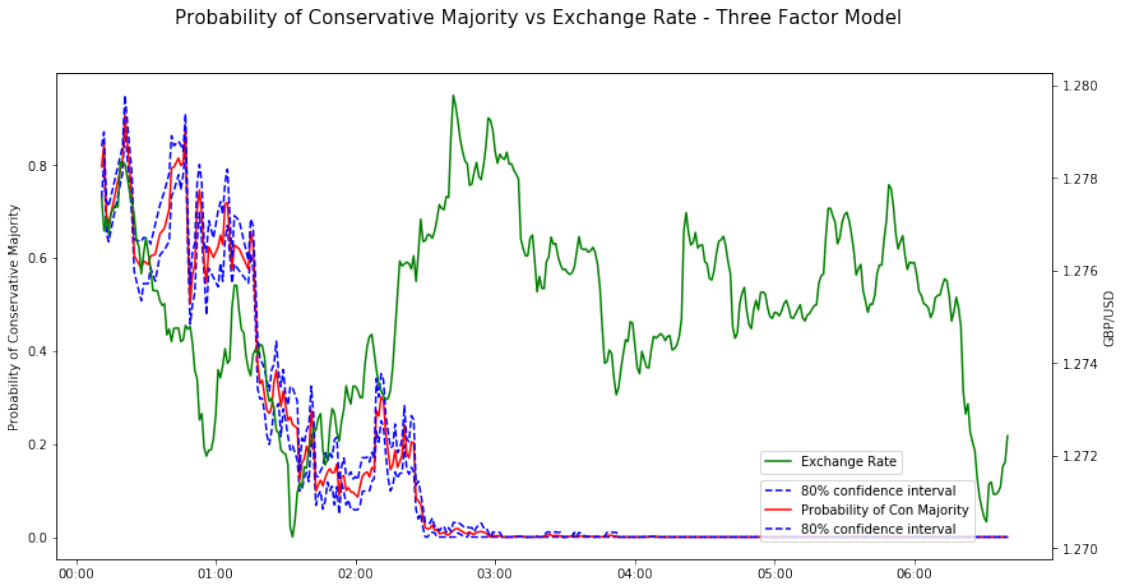
```
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0



prob = np.zeros(10)
for k in range(0,10):

    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
    result = np.zeros([sim_num, len(predict_lab)])

    #Sampling the regression parameters to generate predicted outcome
    for j in range(0, sim_num):

        #Labour
        param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), M
        predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_const


        #fill NaN with mean
        #predict_lab = predict_lab.fillna(predict_lab.mean())
        predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

        result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
        #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
        err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resi
        result_lab = np.random.normal(result_temp,err_dev)


        #Conservative
        param_con = np.random.multivariate_normal(np.asarray(model_con.params), M
        predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_


        #predict_con = predict_con.fillna(predict_con.mean())
        predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

        result_temp = np.matmul(param_con,np.array(predict_con).T)
        #result_con = np.random.normal(result_temp,np.std(model_con.resid))
        err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resi
        result_con = np.random.normal(result_temp,err_dev)
```

```
        #SNP
        predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_cons
        predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
        if not model_snp==0 and len(model_snp.resid)>8:
            param_snp = np.random.multivariate_normal(np.asarray(model_snp.params

            #predict_snp = predict_snp.fillna(predict_snp.mean())

            result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
            #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
            err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp
            result_snp = np.random.normal(result_temp,err_dev)
            #if the poll is 0, the resulting simulated data should also be 0
            ind=np.where(predict_snp.SNP_poll==0)
            result_snp[:,ind]=0

        else:
            #no data
            result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

        #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,
        #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_nu
        other = 1-result_con-result_lab-result_snp

        result_temp = (result_con>result_lab) & (result_con>result_snp) & (result
        result = np.append(result, result_temp,axis=0)

    #remove the zeros during initialisation
    result = np.delete(result,range(0,sim_num),0)
    result = result.astype(int)

    #filling the result of intermediates states with current count
#    select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.i

    EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(res
#    print(jointb['time'][i],jointb['State'][i], EEV)

    prob[k] = sum(EEV>326)/len(EEV)


#make table for plot
pred.time[i]=full_table.time[i]
pred.Constituency[i]=full_table.ID[i]
pred.low[i]=np.percentile(prob,10)
pred.ave[i]=np.mean(prob)
```

```
            pred.high[i]=np.percentile(prob,90)

        # keep only the last value at a certain time. And remove NAs.
        pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
        pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:143: RuntimeWarning: invalid v
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

In [205]:
```
#%%
fig, ax1 = plt.subplots()
fig.set_size_inches(14, 7)
ax1.plot_date(pred.time, pred.low,'b--', label='80% confidence interval')
ax1.plot_date(pred.time, pred.ave,'r-', label='Probability of Con Majority')
ax1.plot_date(pred.time, pred.high,'b--', label='80% confidence interval')
plt.ylabel('Probability of Conservative Majority')
#pylab.legend(loc='lower right')
#plt.axhline(y=270)
ax1.legend(loc='center left', bbox_to_anchor=(0.7, 0.1))
#ax1.set_ylim([0.5,0.9])

ax2 = ax1.twinx()
ax2.plot(fx['mean'],'g', label='Exchange Rate')
#ax2.set_ylim([18.5,21.5])

HMFmt = mdates.DateFormatter('%H:%M')
ax1.xaxis.set_major_formatter(HMFmt)
_ = plt.xticks(rotation=90)

plt.ylabel('GBP/USD')

#pylab.legend(loc='upper right')
ax2.legend(loc='center left', bbox_to_anchor=(0.7, 0.2))

fig.suptitle('Probability of Conservative Majority vs Exchange Rate - Three Factor M

#plt.figure(figsize=(20,10))
plt.show()
fig.savefig('prob-threefac.png')
```



2.3 Regional Dummies Model

```
In [259]: columns = ['time', 'Constituency','low', 'ave', 'high']
          index= full_table.index
          pred=pd.DataFrame(index=index, columns=columns)

          parameters_con = np.empty((650,13))
          stderr_con = np.empty((650,13))
          parameters_lab = np.empty((650,13))
          stderr_lab = np.empty((650,13))
          parameters_snp = np.empty((650,4))
          stderr_snp = np.empty((650,4))

          sim_num = 50
          for i in range(18,len(full_table)-14):

              #labour regression
             # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
              data_lab= pd.concat([full_table['Lab_poll'],\
                                   full_table['Lab_poll']*full_table['london'],full_table['Lab
                                   full_table['Lab_poll']*full_table['isscot'],
                                   full_table['WageMedianConst'] , \
                                   full_table['WageMedianConst']*full_table['london'], full_tab
                                   full_table['WageMedianConst']*full_table['isscot'],\
                                   full_table['Pop65ConstRate'] ,\
                                   full_table['Pop65ConstRate']*full_table['london'], full_tabl
                                   full_table['Pop65ConstRate']*full_table['isscot'],\
                                    full_table['Lab[c]']], axis=1)
```

```
#,full_table['Con_poll']*full_table['iscon'],
X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
Y_lab=data_lab['Lab[c]'][0:i]
X_lab=sm.add_constant(X_lab, has_constant='add')
model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

parameters_lab[i,:]=model_lab.params
stderr_lab[i,:]=model_lab.bse

#Conservatives regression
#data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
data_con= pd.concat([full_table['Con_poll'],\
                     full_table['Con_poll']*full_table['london'],full_table['Con
                     full_table['Con_poll']*full_table['isscot'],
                     full_table['WageMedianConst'],  \
                     full_table['WageMedianConst']*full_table['london'], full_tab
                     full_table['WageMedianConst']*full_table['isscot'],\
                     full_table['Pop65ConstRate'] ,\
                     full_table['Pop65ConstRate']*full_table['london'], full_tabl
                     full_table['Pop65ConstRate']*full_table['isscot'],\
                     full_table['Con[b]']], axis=1)
X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
Y_con=data_con['Con[b]'][0:i]
X_con=sm.add_constant(X_con, has_constant='add')
model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

parameters_con[i,:]=model_con.params
stderr_con[i,:]=model_con.bse

data_snp=pd.concat([full_table['SNP_poll'], full_table['WageMedianConst'],
                    full_table['Pop65ConstRate'], full_table['SNP']], axis=1)
X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0



predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant='ad
predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
result = np.zeros([sim_num, len(predict_lab)])
```

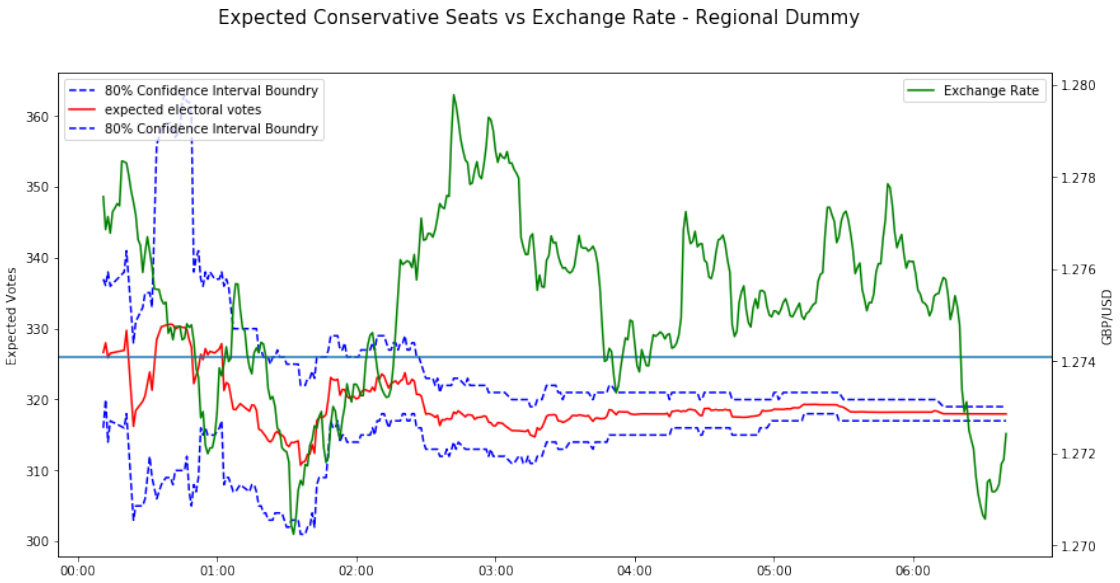
```
#Sampling the regression parameters to generate predicted outcome
for j in range(0, sim_num):

    #Labour
    param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), Regre
    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=


    #fill NaN with mean
    #predict_lab = predict_lab.fillna(predict_lab.mean())
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

    result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
    #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
    err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resid)-1
    result_lab = np.random.normal(result_temp,err_dev)


    #Conservative
    param_con = np.random.multivariate_normal(np.asarray(model_con.params), Regre
    predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_cons


    #predict_con = predict_con.fillna(predict_con.mean())
    predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

    result_temp = np.matmul(param_con,np.array(predict_con).T)
    #result_con = np.random.normal(result_temp,np.std(model_con.resid))
    err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resid)-1
    result_con = np.random.normal(result_temp,err_dev)


    #SNP
    predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_constant
    predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
    if not model_snp==0 and len(model_snp.resid)>8:
        param_snp = np.random.multivariate_normal(np.asarray(model_snp.params), N

        #predict_snp = predict_snp.fillna(predict_snp.mean())

        result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
        #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
        err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp.resi
        result_snp = np.random.normal(result_temp,err_dev)
        #if the poll is 0, the resulting simulated data should also be 0
        ind=np.where(predict_snp.SNP_poll==0)
        result_snp[:,ind]=0
```

```
                else:
                    #no data
                    result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

                    #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,1)))
                    #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_num,1)
                    other = 1-result_con-result_lab-result_snp

                    result_temp = (result_con>result_lab) & (result_con>result_snp) & (result_co
                    result = np.append(result, result_temp,axis=0)

            #remove the zeros during initialisation
            result = np.delete(result,range(0,sim_num),0)
            result = result.astype(int)

            #filling the result of intermediates states with current count
        #    select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.isnul

            EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(result
        #    print(jointb['time'][i],jointb['State'][i], EEV)


            #make table for plot
            pred.time[i]=full_table.time[i]
            pred.Constituency[i]=full_table.ID[i]
            pred.low[i]=np.percentile(EEV,10)
            pred.ave[i]=np.mean(EEV)
            pred.high[i]=np.percentile(EEV,90)

        # keep only the last value at a certain time. And remove NAs.
        pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
        pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:75: RuntimeWarning: covariance
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:90: RuntimeWarning: covariance
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:127: RuntimeWarning: invalid v
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

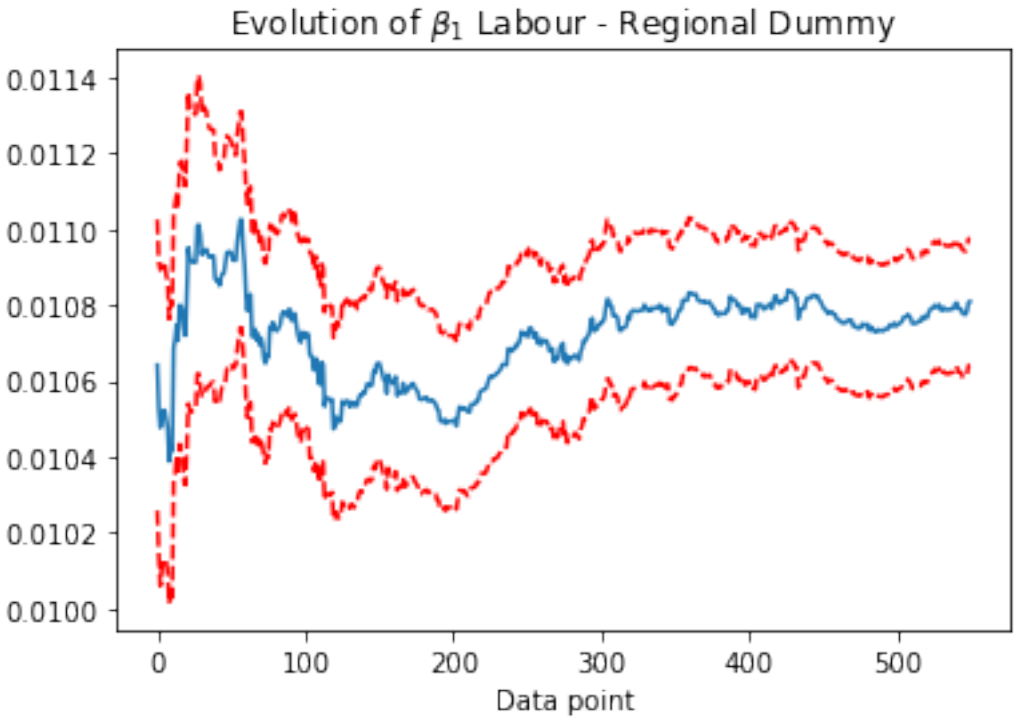
In [252]:
```
fig, ax1 = plt.subplots()
fig.set_size_inches(14, 7)
ax1.plot_date(pred.time, pred.low,'b--', label='80% Confidence Interval Boundry')
ax1.plot_date(pred.time, pred.ave,'r-', label='expected electoral votes')
ax1.plot_date(pred.time, pred.high,'b--', label='80% Confidence Interval Boundry')
plt.ylabel('Expected Votes')
pylab.legend(loc='upper left')
plt.axhline(y=326)
```

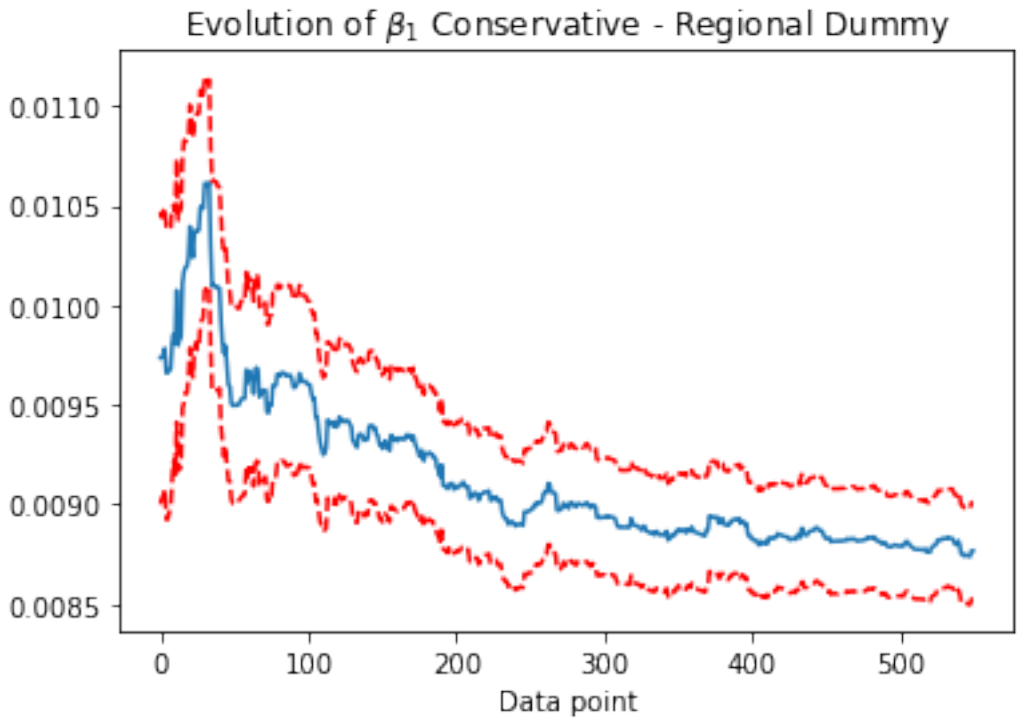
```
#ax1.set_ylim([280,380])

ax2 = ax1.twinx()
ax2.plot(fx['mean'],'g', label='Exchange Rate')

HMFmt = mdates.DateFormatter('%H:%M')
ax1.xaxis.set_major_formatter(HMFmt)
_ = plt.xticks(rotation=90)

plt.ylabel('GBP/USD')
plt.xlabel('UTC Time')
pylab.legend(loc='upper right')

fig.suptitle('Expected Conservative Seats vs Exchange Rate - Regional Dummy', fontsiz
fig.savefig('results-dummy.png')
plt.show()
```



In [208]:
```
plt.plot(parameters_lab[50:600,1])
plt.plot(parameters_lab[50:600,1]-stderr_lab[50:600,1],'r--')
plt.plot(parameters_lab[50:600,1]+stderr_lab[50:600,1],'r--')
plt.title(r'Evolution of $\beta_1 $ Labour - Regional Dummy')
plt.xlabel('Data point')
plt.savefig('lab dummy b1.png')
plt.show()

plt.plot(parameters_con[50:600,1])
plt.plot(parameters_con[50:600,1]-stderr_con[50:600,1],'r--')
plt.plot(parameters_con[50:600,1]+stderr_con[50:600,1],'r--')
```

```
plt.title(r'Evolution of $\beta_1 $ Conservative - Regional Dummy')
plt.xlabel('Data point')
plt.savefig('con dummy b1.png')
plt.show()
```






```
In [209]: columns = ['time', 'Constituency','low', 'ave', 'high']
          index= full_table.index
          pred=pd.DataFrame(index=index, columns=columns)

          parameters_con = np.empty((650,13))
          stderr_con = np.empty((650,13))
          parameters_lab = np.empty((650,13))
          stderr_lab = np.empty((650,13))
          parameters_snp = np.empty((650,4))
          stderr_snp = np.empty((650,4))

          sim_num = 10
          for i in range(18,len(full_table)-14):

              #labour regression
             # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
              data_lab= pd.concat([full_table['Lab_poll'],\
                                   full_table['Lab_poll']*full_table['london'],full_table['Lab
                                   full_table['Lab_poll']*full_table['isscot'],
                                   full_table['WageMedianConst'] , \
                                   full_table['WageMedianConst']*full_table['london'], full_tab
                                   full_table['WageMedianConst']*full_table['isscot'],\
                                   full_table['Pop65ConstRate'] ,\
```

```
                    full_table['Pop65ConstRate']*full_table['london'], full_tabl
                    full_table['Pop65ConstRate']*full_table['isscot'],\
                     full_table['Lab[c]']], axis=1)
#,full_table['Con_poll']*full_table['iscon'],
X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
Y_lab=data_lab['Lab[c]'][0:i]
X_lab=sm.add_constant(X_lab, has_constant='add')
model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

parameters_lab[i,:]=model_lab.params
stderr_lab[i,:]=model_lab.bse

#Conservatives regression
#data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
data_con= pd.concat([full_table['Con_poll'],\
                    full_table['Con_poll']*full_table['london'],full_table['Con
                    full_table['Con_poll']*full_table['isscot'],
                    full_table['WageMedianConst'],  \
                    full_table['WageMedianConst']*full_table['london'], full_tab
                    full_table['WageMedianConst']*full_table['isscot'],\
                    full_table['Pop65ConstRate'] ,\
                    full_table['Pop65ConstRate']*full_table['london'], full_tabl
                    full_table['Pop65ConstRate']*full_table['isscot'],\
                    full_table['Con[b]']], axis=1)
X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
Y_con=data_con['Con[b]'][0:i]
X_con=sm.add_constant(X_con, has_constant='add')
model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

parameters_con[i,:]=model_con.params
stderr_con[i,:]=model_con.bse


data_snp=pd.concat([full_table['SNP_poll'], full_table['WageMedianConst'],
                    full_table['Pop65ConstRate'], full_table['SNP']], axis=1)
X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0


prob = np.zeros(10)
for k in range(0,10):
```

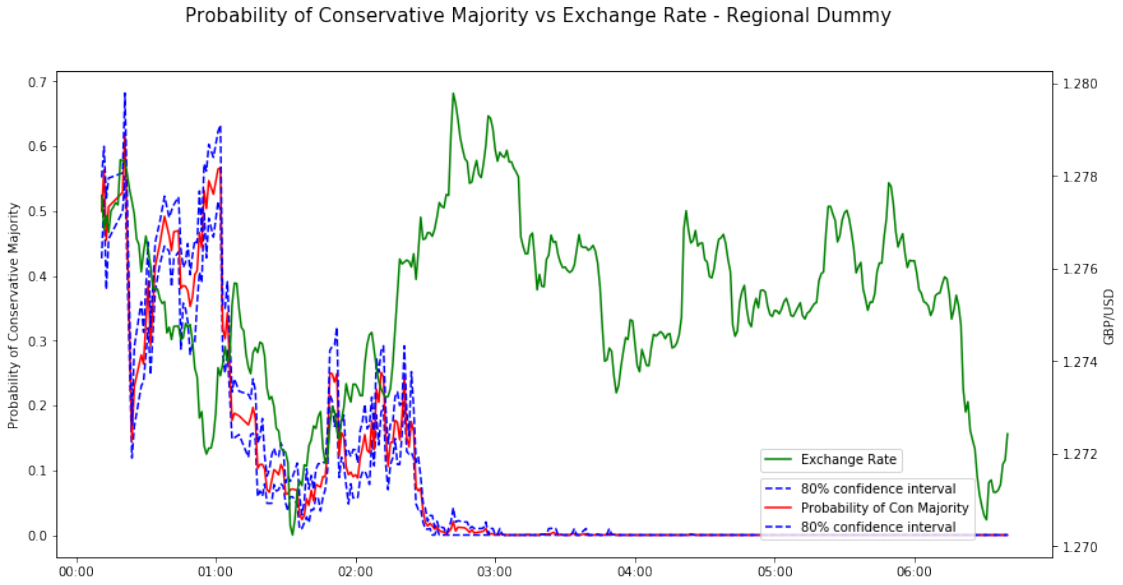
```
predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=
predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
result = np.zeros([sim_num, len(predict_lab)])


#Sampling the regression parameters to generate predicted outcome
for j in range(0, sim_num):

    #Labour
    param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), I
    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_const


    #fill NaN with mean
    #predict_lab = predict_lab.fillna(predict_lab.mean())
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

    result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
    #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
    err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resi
    result_lab = np.random.normal(result_temp,err_dev)


    #Conservative
    param_con = np.random.multivariate_normal(np.asarray(model_con.params), I
    predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_


    #predict_con = predict_con.fillna(predict_con.mean())
    predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

    result_temp = np.matmul(param_con,np.array(predict_con).T)
    #result_con = np.random.normal(result_temp,np.std(model_con.resid))
    err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resi
    result_con = np.random.normal(result_temp,err_dev)




    #SNP
    predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_cons
    predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
    if not model_snp==0 and len(model_snp.resid)>8:
        param_snp = np.random.multivariate_normal(np.asarray(model_snp.params

        #predict_snp = predict_snp.fillna(predict_snp.mean())
```

```
                    result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
                    #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
                    err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp
                    result_snp = np.random.normal(result_temp,err_dev)
                    #if the poll is 0, the resulting simulated data should also be 0
                    ind=np.where(predict_snp.SNP_poll==0)
                    result_snp[:,ind]=0

                else:
                    #no data
                    result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

                #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,
                #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_nu
                other = 1-result_con-result_lab-result_snp

                result_temp = (result_con>result_lab) & (result_con>result_snp) & (result
                result = np.append(result, result_temp,axis=0)

            #remove the zeros during initialisation
            result = np.delete(result,range(0,sim_num),0)
            result = result.astype(int)

            #filling the result of intermediates states with current count
        #    select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.i

            EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(res
        #    print(jointb['time'][i],jointb['State'][i], EEV)



            prob[k] = sum(EEV>326)/len(EEV)


        #make table for plot
        pred.time[i]=full_table.time[i]
        pred.Constituency[i]=full_table.ID[i]
        pred.low[i]=np.percentile(prob,10)
        pred.ave[i]=np.mean(prob)
        pred.high[i]=np.percentile(prob,90)

    # keep only the last value at a certain time. And remove NAs.
    pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
    pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:92: RuntimeWarning: covariance
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:107: RuntimeWarning: covarianc
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:156: RuntimeWarning: invalid v
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

```
In [210]: #%%
          fig, ax1 = plt.subplots()
          fig.set_size_inches(14, 7)
          ax1.plot_date(pred.time, pred.low,'b--', label='80% confidence interval')
          ax1.plot_date(pred.time, pred.ave,'r-', label='Probability of Con Majority')
          ax1.plot_date(pred.time, pred.high,'b--', label='80% confidence interval')
          plt.ylabel('Probability of Conservative Majority')
          #pylab.legend(loc='lower right')
          #plt.axhline(y=270)
          ax1.legend(loc='center left', bbox_to_anchor=(0.7, 0.1))
          #ax1.set_ylim([0.5,0.9])

          ax2 = ax1.twinx()
          ax2.plot(fx['mean'],'g', label='Exchange Rate')
          #ax2.set_ylim([18.5,21.5])

          HMFmt = mdates.DateFormatter('%H:%M')
          ax1.xaxis.set_major_formatter(HMFmt)
          _ = plt.xticks(rotation=90)

          plt.ylabel('GBP/USD')

          #pylab.legend(loc='upper right')
          ax2.legend(loc='center left', bbox_to_anchor=(0.7, 0.2))

          fig.suptitle('Probability of Conservative Majority vs Exchange Rate - Regional Dummy

          #plt.figure(figsize=(20,10))
          plt.show()
          fig.savefig('prob-dummy.png')
```



2.4 Best Subset Regression

```
In [253]: X_con = data_con.drop('Con[b]',axis=1)
          Y_con=data_con['Con[b]']
          predictorcols=X_con
          target=Y_con
          train=X_con
```

```
In [254]: import itertools

          AICs = {}
          for k in range(1,len(predictorcols)+1):
          #    print(k)
              for variables in itertools.combinations(predictorcols, k):
                  predictors = train[list(variables)]
                  predictors['Intercept'] = 1
                  res = sm.OLS(target, predictors, missing='drop').fit()
                  AICs[variables] = 2*(k+1) - 2*res.llf
          pd.Series(AICs).idxmin()
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
Out[254]: ('Con_poll', 0, 2, 3, 4, 'Pop65ConstRate', 6, 8)
```

```
In [255]: X_lab = data_lab.drop('Lab[c]',axis=1)
          Y_lab=data_lab['Lab[c]']
          predictorcols=X_lab
          target=Y_lab
          train=X_lab
```

```
In [256]: AICs = {}
          for k in range(1,len(predictorcols)+1):
          #    print(k)
              for variables in itertools.combinations(predictorcols, k):
                  predictors = train[list(variables)]
                  predictors['Intercept'] = 1
                  res = sm.OLS(target, predictors, missing='drop').fit()
                  AICs[variables] = 2*(k+1) - 2*res.llf
          pd.Series(AICs).idxmin()
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
Out[256]: ('Lab_poll', 0, 1, 'WageMedianConst', 3, 4, 5, 'Pop65ConstRate', 7)
```

2.5 Final

```
In [263]: columns = ['time', 'Constituency','low', 'ave', 'high']
          index= full_table.index
          pred=pd.DataFrame(index=index, columns=columns)

          parameters_con = np.empty((650,9))
          stderr_con = np.empty((650,9))
          parameters_lab = np.empty((650,10))
          stderr_lab = np.empty((650,10))
          parameters_snp = np.empty((650,4))
          stderr_snp = np.empty((650,4))

          sim_num = 50
          for i in range(18,len(full_table)-14):

              #labour regression
             # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
              data_lab= pd.concat([full_table['Lab_poll'],\
                                   full_table['Lab_poll']*full_table['london'],full_table['Lab
                                   full_table['WageMedianConst'] , \
                                   full_table['WageMedianConst']*full_table['london'], full_tab
                                   full_table['WageMedianConst']*full_table['isscot'],\
```

```
                    full_table['Pop65ConstRate'] ,\
                    full_table['Pop65ConstRate']*full_table['iswales'],\
                     full_table['Lab[c]']], axis=1)
#,full_table['Con_poll']*full_table['iscon'],
X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
Y_lab=data_lab['Lab[c]'][0:i]
X_lab=sm.add_constant(X_lab, has_constant='add')
model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

parameters_lab[i,:]=model_lab.params
stderr_lab[i,:]=model_lab.bse

#Conservatives regression
#data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
data_con= pd.concat([full_table['Con_poll'],\
                    full_table['Con_poll']*full_table['london'],\
                    full_table['Con_poll']*full_table['isscot'],
                    full_table['WageMedianConst']*full_table['london'], full_tab
                    full_table['Pop65ConstRate'] ,\
                    full_table['Pop65ConstRate']*full_table['london'],\
                    full_table['Pop65ConstRate']*full_table['isscot'],\
                    full_table['Con[b]']], axis=1)
X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
Y_con=data_con['Con[b]'][0:i]
X_con=sm.add_constant(X_con, has_constant='add')
model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

parameters_con[i,:]=model_con.params
stderr_con[i,:]=model_con.bse

data_snp=pd.concat([full_table['SNP_poll'], full_table['WageMedianConst'],
                  full_table['Pop65ConstRate'], full_table['SNP']], axis=1)
X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0



predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant='add
predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
result = np.zeros([sim_num, len(predict_lab)])
```

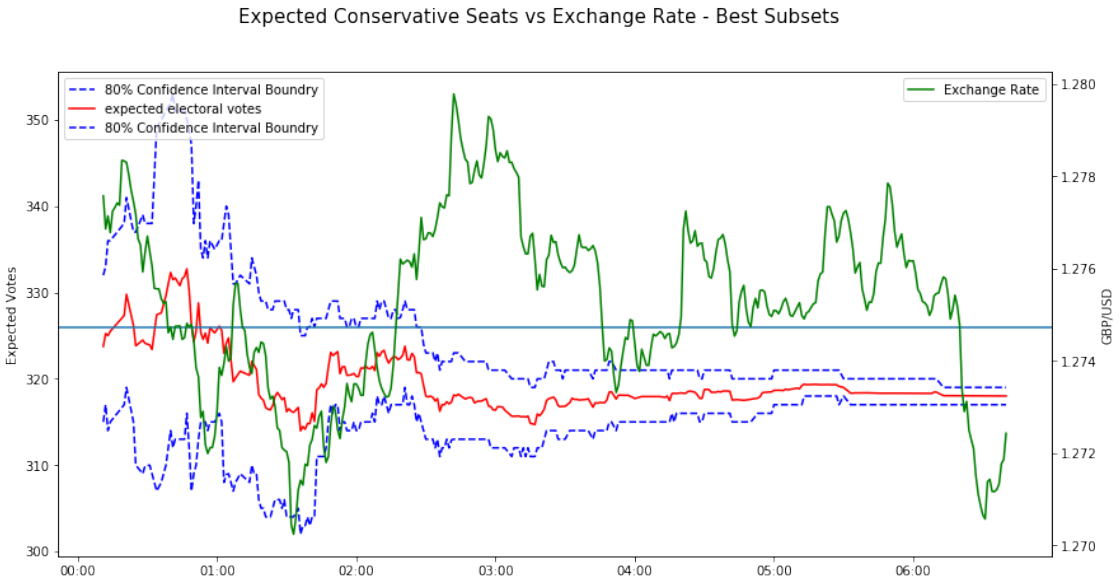
```
#Sampling the regression parameters to generate predicted outcome
for j in range(0, sim_num):

    #Labour
    param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), Regre
    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=


    #fill NaN with mean
    #predict_lab = predict_lab.fillna(predict_lab.mean())
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

    result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
    #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
    err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resid)-1
    result_lab = np.random.normal(result_temp,err_dev)


    #Conservative
    param_con = np.random.multivariate_normal(np.asarray(model_con.params), Regre
    predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has_cons


    #predict_con = predict_con.fillna(predict_con.mean())
    predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

    result_temp = np.matmul(param_con,np.array(predict_con).T)
    #result_con = np.random.normal(result_temp,np.std(model_con.resid))
    err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resid)-1
    result_con = np.random.normal(result_temp,err_dev)




    #SNP
    predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_constant
    predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
    if not model_snp==0 and len(model_snp.resid)>8:
        param_snp = np.random.multivariate_normal(np.asarray(model_snp.params), M

        #predict_snp = predict_snp.fillna(predict_snp.mean())

        result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
        #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
        err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp.resi
        result_snp = np.random.normal(result_temp,err_dev)
        #if the poll is 0, the resulting simulated data should also be 0
```

```
                    ind=np.where(predict_snp.SNP_poll==0)
                    result_snp[:,ind]=0

                else:
                    #no data
                    result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

                #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,1)))
                #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_num,1)
                other = 1-result_con-result_lab-result_snp

                result_temp = (result_con>result_lab) & (result_con>result_snp) & (result_co
                result = np.append(result, result_temp,axis=0)

            #remove the zeros during initialisation
            result = np.delete(result,range(0,sim_num),0)
            result = result.astype(int)

            EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(result


            #make table for plot
            pred.time[i]=full_table.time[i]
            pred.Constituency[i]=full_table.ID[i]
            pred.low[i]=np.percentile(EEV,10)
            pred.ave[i]=np.mean(EEV)
            pred.high[i]=np.percentile(EEV,90)

        # keep only the last value at a certain time. And remove NAs.
        pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
        pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:72: RuntimeWarning: covariance
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:87: RuntimeWarning: covariance
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:126: RuntimeWarning: invalid v
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

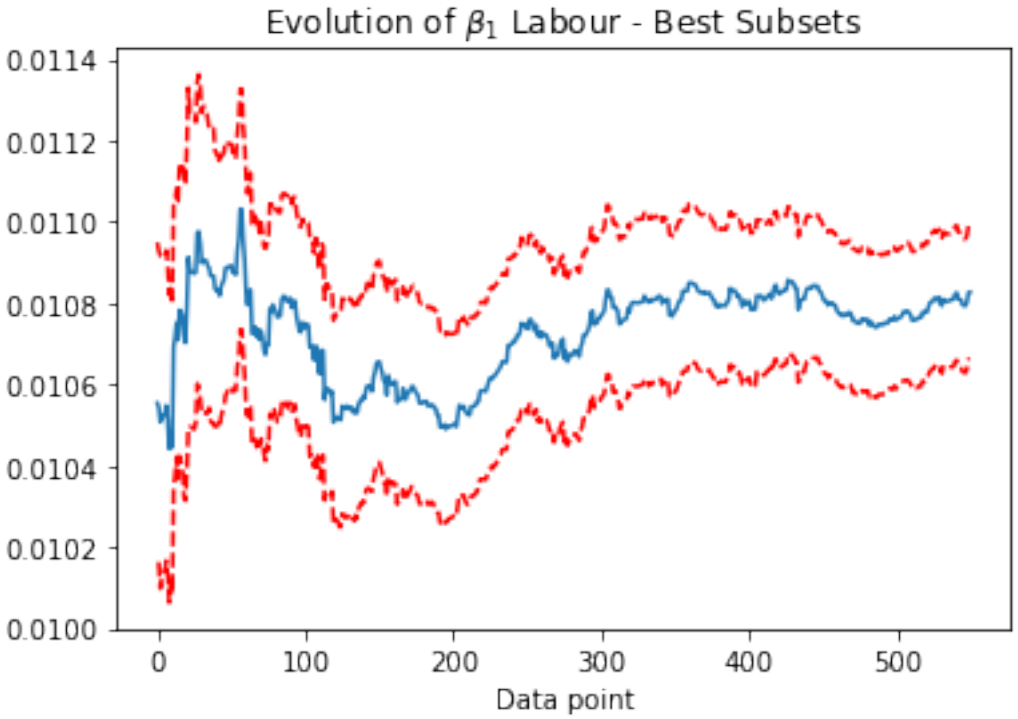
In [212]:
```
fig, ax1 = plt.subplots()
fig.set_size_inches(14, 7)
ax1.plot_date(pred.time, pred.low,'b--', label='80% Confidence Interval Boundry')
ax1.plot_date(pred.time, pred.ave,'r-', label='expected electoral votes')
ax1.plot_date(pred.time, pred.high,'b--', label='80% Confidence Interval Boundry')
plt.ylabel('Expected Votes')
pylab.legend(loc='upper left')
plt.axhline(y=326)
#ax1.set_ylim([280,380])
```

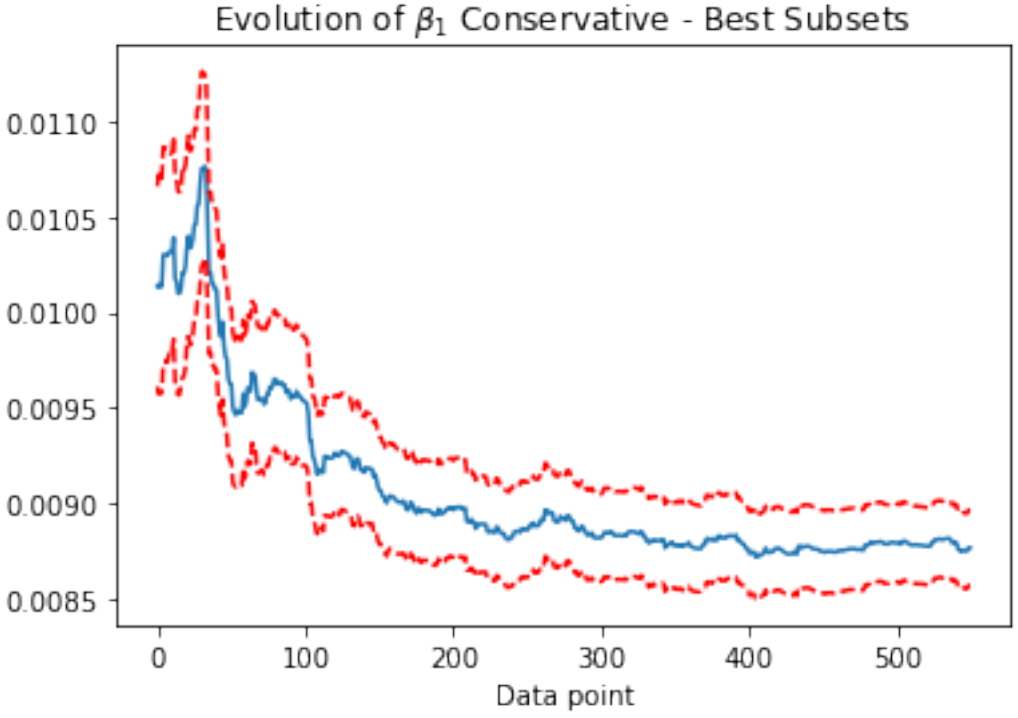
```
ax2 = ax1.twinx()
ax2.plot(fx['mean'],'g', label='Exchange Rate')

HMFmt = mdates.DateFormatter('%H:%M')
ax1.xaxis.set_major_formatter(HMFmt)
_ = plt.xticks(rotation=90)

plt.ylabel('GBP/USD')
plt.xlabel('UTC Time')
pylab.legend(loc='upper right')

fig.suptitle('Expected Conservative Seats vs Exchange Rate - Best Subsets', fontsize=
fig.savefig('results-final.png')
plt.show()
```



In [213]:
```
plt.plot(parameters_lab[50:600,1])
plt.plot(parameters_lab[50:600,1]-stderr_lab[50:600,1],'r--')
plt.plot(parameters_lab[50:600,1]+stderr_lab[50:600,1],'r--')
plt.title(r'Evolution of $\beta_1 $ Labour - Best Subsets')
plt.xlabel('Data point')
plt.savefig('lab final b1.png')
plt.show()

plt.plot(parameters_con[50:600,1])
plt.plot(parameters_con[50:600,1]-stderr_con[50:600,1],'r--')
plt.plot(parameters_con[50:600,1]+stderr_con[50:600,1],'r--')
plt.title(r'Evolution of $\beta_1 $ Conservative - Best Subsets')
```

```
plt.xlabel('Data point')
plt.savefig('con final b1.png')
plt.show()
```





In [214]:
```
columns = ['time', 'Constituency','low', 'ave', 'high']
index= full_table.index
pred=pd.DataFrame(index=index, columns=columns)

parameters_con = np.empty((650,9))
stderr_con = np.empty((650,9))
parameters_lab = np.empty((650,10))
stderr_lab = np.empty((650,10))
parameters_snp = np.empty((650,4))
stderr_snp = np.empty((650,4))

sim_num = 10
for i in range(18,len(full_table)-14):

    #labour regression
   # data_lab=full_table[['Lab_poll','Lab[c]','Total','Last Election','WageMedianCon
    data_lab= pd.concat([full_table['Lab_poll'],\
                         full_table['Lab_poll']*full_table['london'],full_table['Lab
                         full_table['WageMedianConst'] , \
                         full_table['WageMedianConst']*full_table['london'], full_tab
                         full_table['WageMedianConst']*full_table['isscot'],\
                         full_table['Pop65ConstRate'] ,\
                         full_table['Pop65ConstRate']*full_table['iswales'],\
                          full_table['Lab[c]']], axis=1)
    #,full_table['Con_poll']*full_table['iscon'],
    X_lab = data_lab.drop('Lab[c]',axis=1)[0:i]
    Y_lab=data_lab['Lab[c]'][0:i]
    X_lab=sm.add_constant(X_lab, has_constant='add')
    model_lab = sm.RLM(Y_lab, X_lab,missing = 'drop').fit()

    parameters_lab[i,:]=model_lab.params
    stderr_lab[i,:]=model_lab.bse

    #Conservatives regression
    #data_con=full_table[['Con_poll','Con[b]','Total','Last Election','WageMedianCon
    data_con= pd.concat([full_table['Con_poll'],\
                         full_table['Con_poll']*full_table['london'],\
                         full_table['Con_poll']*full_table['isscot'],
                         full_table['WageMedianConst']*full_table['london'], full_tab
                         full_table['Pop65ConstRate'] ,\
                         full_table['Pop65ConstRate']*full_table['london'],\
                         full_table['Pop65ConstRate']*full_table['isscot'],\
                         full_table['Con[b]']], axis=1)
    X_con = data_con.drop(['Con[b]'], axis = 1)[0:i]
    Y_con=data_con['Con[b]'][0:i]
```

```
X_con=sm.add_constant(X_con, has_constant='add')
model_con = sm.RLM(Y_con, X_con,missing = 'drop').fit()

parameters_con[i,:]=model_con.params
stderr_con[i,:]=model_con.bse



data_snp=pd.concat([full_table['SNP_poll'], full_table['WageMedianConst'],
                    full_table['Pop65ConstRate'], full_table['SNP']], axis=1)
X_snp = data_snp.drop(['SNP'],axis = 1)[0:i]
Y_snp=data_snp['SNP'][0:i]
X_snp=sm.add_constant(X_snp, has_constant='add')
try:
    model_snp = sm.RLM(Y_snp, X_snp,missing = 'drop').fit()
    parameters_snp[i,:]
    stderr_snp[i,:]=model_snp.bse
except:
    model_snp = 0


prob = np.zeros(10)
for k in range(0,10):

    predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_constant=
    predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]
    result = np.zeros([sim_num, len(predict_lab)])


    #Sampling the regression parameters to generate predicted outcome
    for j in range(0, sim_num):

        #Labour
        param_lab = np.random.multivariate_normal(np.asarray(model_lab.params), I
        predict_lab=sm.add_constant(data_lab.drop('Lab[c]',axis=1)[i:], has_const


        #fill NaN with mean
        #predict_lab = predict_lab.fillna(predict_lab.mean())
        predict_lab=predict_lab[np.isfinite(predict_lab['Lab_poll'])]

        result_temp =  np.matmul(param_lab,np.array(predict_lab).T)
        #result_lab = np.random.normal(result_temp,np.std(model_lab.resid))
        err_dev = np.std(model_lab.resid)*(np.random.chisquare(len(model_lab.resi
        result_lab = np.random.normal(result_temp,err_dev)
```

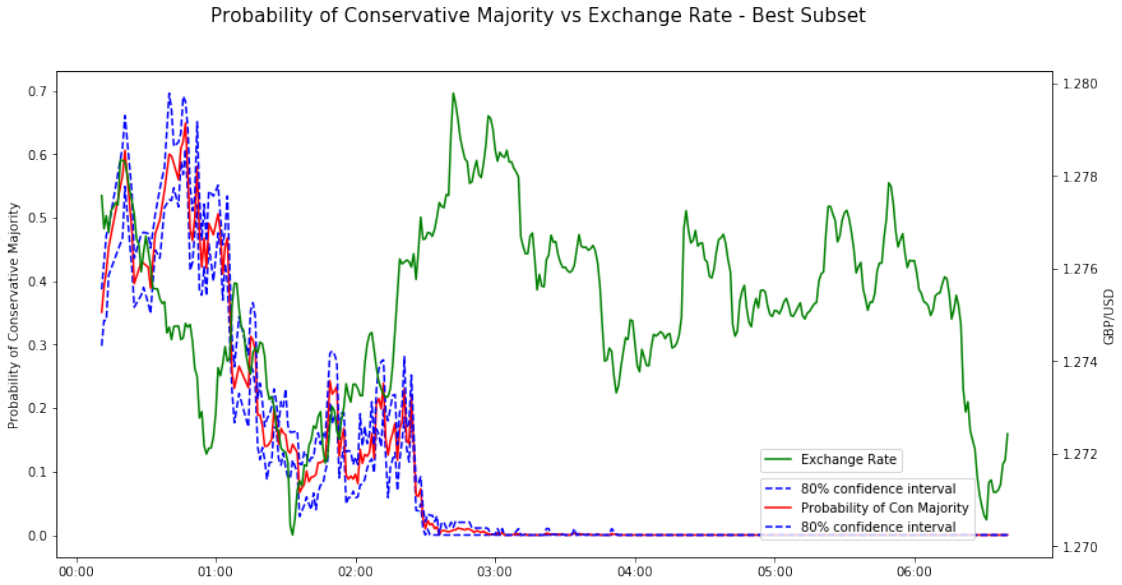
```
        #Conservative
        param_con = np.random.multivariate_normal(np.asarray(model_con.params), 
        predict_con=sm.add_constant(data_con.drop(['Con[b]'], axis = 1)[i:], has


        #predict_con = predict_con.fillna(predict_con.mean())
        predict_con = predict_con[np.isfinite(predict_con['Con_poll'])]

        result_temp = np.matmul(param_con,np.array(predict_con).T)
        #result_con = np.random.normal(result_temp,np.std(model_con.resid))
        err_dev = np.std(model_con.resid)*(np.random.chisquare(len(model_con.resi
        result_con = np.random.normal(result_temp,err_dev)



        #SNP
        predict_snp=sm.add_constant(data_snp.drop(['SNP'],axis = 1)[i:], has_cons
        predict_snp = predict_snp[np.isfinite(predict_snp['SNP_poll'])]
        if not model_snp==0 and len(model_snp.resid)>8:
            param_snp = np.random.multivariate_normal(np.asarray(model_snp.params

            #predict_snp = predict_snp.fillna(predict_snp.mean())

            result_temp =  np.matmul(param_snp,np.array(predict_snp).T)
            #result_snp = np.random.normal(result_temp,np.std(model_snp.resid))
            err_dev = np.std(model_snp.resid)*(np.random.chisquare(len(model_snp
            result_snp = np.random.normal(result_temp,err_dev)
            #if the poll is 0, the resulting simulated data should also be 0
            ind=np.where(predict_snp.SNP_poll==0)
            result_snp[:,ind]=0

        else:
            #no data
            result_snp = np.tile(predict_snp['SNP_poll'],(sim_num,1))/100

        #result_pla = np.nan_to_num(np.tile(full_table['Pla_poll'][i:],(sim_num,
        #result_ukip = np.nan_to_num(np.tile(full_table['UKIP_poll'][i:],(sim_nu
        other = 1-result_con-result_lab-result_snp

        result_temp = (result_con>result_lab) & (result_con>result_snp) & (result
        result = np.append(result, result_temp,axis=0)

    #remove the zeros during initialisation
    result = np.delete(result,range(0,sim_num),0)
    result = result.astype(int)

    #filling the result of intermediates states with current count
#    select.result[select.result.isnull()[0:i]]=(select['trump'][select.result.i
```

```
                EEV=len(full_table.Party[0:i][full_table.Party=='Con'])+ np.count_nonzero(res
            #    print(jointb['time'][i],jointb['State'][i], EEV)



                prob[k] = sum(EEV>326)/len(EEV)


            #make table for plot
            pred.time[i]=full_table.time[i]
            pred.Constituency[i]=full_table.ID[i]
            pred.low[i]=np.percentile(prob,10)
            pred.ave[i]=np.mean(prob)
            pred.high[i]=np.percentile(prob,90)

        # keep only the last value at a certain time. And remove NAs.
        pred=pred.drop_duplicates(subset='time', keep='last').dropna(axis=0, how='all')
        pred.index = np.arange(0,len(pred))
```

```
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:89: RuntimeWarning: covariance
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:104: RuntimeWarning: covarian
C:\Users\AlexH\Anaconda3\lib\site-packages\ipykernel_launcher.py:153: RuntimeWarning: invalid
C:\Users\AlexH\Anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:1353: Runtime
  return np.dot(wresid, wresid) / self.df_resid
```

In [215]:
```
        #%%
        fig, ax1 = plt.subplots()
        fig.set_size_inches(14, 7)
        ax1.plot_date(pred.time, pred.low,'b--', label='80% confidence interval')
        ax1.plot_date(pred.time, pred.ave,'r-', label='Probability of Con Majority')
        ax1.plot_date(pred.time, pred.high,'b--', label='80% confidence interval')
        plt.ylabel('Probability of Conservative Majority')
        #pylab.legend(loc='lower right')
        #plt.axhline(y=270)
        ax1.legend(loc='center left', bbox_to_anchor=(0.7, 0.1))
        #ax1.set_ylim([0.5,0.9])

        ax2 = ax1.twinx()
        ax2.plot(fx['mean'],'g', label='Exchange Rate')
        #ax2.set_ylim([18.5,21.5])

        HMFmt = mdates.DateFormatter('%H:%M')
        ax1.xaxis.set_major_formatter(HMFmt)
        _ = plt.xticks(rotation=90)

        plt.ylabel('GBP/USD')
```

```
#pylab.legend(loc='upper right')
ax2.legend(loc='center left', bbox_to_anchor=(0.7, 0.2))

fig.suptitle('Probability of Conservative Majority vs Exchange Rate - Best Subset', i

#plt.figure(figsize=(20,10))
plt.show()
fig.savefig('prob-final.png')
```



In [264]:
```
model_con.summary()
```

Out[264]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                    Robust linear Model Regression Results
==============================================================================
Dep. Variable:                 Con[b]   No. Observations:                  615
Model:                            RLM   Df Residuals:                      606
Method:                          IRLS   Df Model:                            8
Norm:                          HuberT
Scale Est.:                       mad
Cov Type:                          H1
Date:                Sun, 25 Feb 2018
Time:                        14:31:38
No. Iterations:                    20
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const              -0.0573      0.010     -5.830      0.000      -0.077      -0.038
Con_poll            0.0088      0.000     45.892      0.000       0.008       0.009
0                  -0.0017      0.001     -2.338      0.019      -0.003      -0.000
1                   0.0033      0.001      3.800      0.000       0.002       0.005
2               -8.188e-05    3.4e-05     -2.407      0.016      -0.000   -1.52e-05
3                   0.0001   1.85e-05      6.798      0.000    8.95e-05       0.000
Pop65ConstRate      0.5456      0.057      9.560      0.000       0.434       0.658
4                   0.8284      0.252      3.294      0.001       0.335       1.321
5                  -0.3547      0.128     -2.769      0.006      -0.606      -0.104
==============================================================================

If the model instance has been used for another fit with different fit
parameters, then the fit options might not be the correct ones anymore .
"""
```

In [265]:
```
model_lab.summary()
```

Out[265]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                    Robust linear Model Regression Results
==============================================================================
Dep. Variable:                 Lab[c]   No. Observations:                  615
Model:                            RLM   Df Residuals:                      605
Method:                          IRLS   Df Model:                            9
Norm:                          HuberT
Scale Est.:                       mad
Cov Type:                          H1
Date:                Sun, 25 Feb 2018
Time:                        14:31:38
No. Iterations:                    22
==============================================================================
                      coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const                0.1696      0.021      8.065      0.000       0.128       0.211
Lab_poll             0.0108      0.000     67.779      0.000       0.010       0.011
0                    0.0007      0.000      2.516      0.012       0.000       0.001
1                   -0.0014      0.001     -2.578      0.010      -0.003      -0.000
WageMedianConst -5.149e-05   2.53e-05     -2.033      0.042      -0.000   -1.85e-06
2                -3.16e-05   1.92e-05     -1.647      0.100   -6.92e-05       6e-06
3                    0.0003    8.6e-05      2.953      0.003    8.54e-05       0.000
4                   -0.0001   9.51e-06    -10.728      0.000      -0.000   -8.34e-05
Pop65ConstRate      -0.4526      0.044    -10.184      0.000      -0.540      -0.366
5                   -0.4594      0.146     -3.155      0.002      -0.745      -0.174
==============================================================================

If the model instance has been used for another fit with different fit
parameters, then the fit options might not be the correct ones anymore .
"""
```