PossionRegression

April 7, 2020

```
In [ ]: import pandas as pd
        from patsy import dmatrices
        import numpy as np
        import statsmodels.api as sm
        import matplotlib.pyplot as plt
```

```
In [111]: #reading the data frame
          df = pd.read_csv('pest_traps.csv', header=0, infer_datetime_format=True, parse_dates=
```

```
In [112]: df.shape
```

```
Out[112]: (12434, 7)
```

1 Building dataframe for the pest CEW

```
In [113]: #dividing the dataframe based on each pests - pest CEW
          df_CEW = df.loc[['CEW'], ['farm', 'trap_count','year','date']]
```

```
In [114]: df_CEW.head()
```

```
Out[114]:              farm  trap_count  year      date
          pest
          CEW       Pelham-G           0  2006  20060619
          CEW   Litchfield-W           0  2006  20060619
          CEW   Litchfield-M           0  2006  20060619
          CEW    Merrimack-T           0  2006  20060619
          CEW       Hollis-L           1  2006  20060619
```

```
In [115]: df_CEW.tail()
```

```
Out[115]:            farm  trap_count  year      date
          pest
          CEW    Hollis-K           3  2018  20181008
          CEW   Hollis-B2           0  2018  20181008
          CEW    Hollis-L           8  2018  20181008
          CEW    Hollis-K           3  2018  20181015
          CEW    Hollis-L           2  2018  20181015
```

```
In [116]: df_CEW.shape

Out[116]: (3626, 4)

In [117]: from datetime import date

          def compute_weeks(startDate, endDate):
              s_yyyy = str(startDate)[0:4]
              s_mm = str(startDate)[5:7]
              s_dd = str(startDate)[8:10]

              d1 = date(int(s_yyyy),int(s_mm),int(s_dd))

              e_yyyy = str(endDate)[0:4]
              e_mm = str(endDate)[5:7]
              e_dd = str(endDate)[8:10]

              d2 = date(int(e_yyyy),int(e_mm),int(e_dd))

              return (int((d2-d1).days / 7))

          def convert_pandasDate(change_date):
              change_date = str(change_date)[:4] + "/" + str(change_date)[4:6] + "/" + str(cha
              change_date = pd.to_datetime(change_date)
              return (change_date)


In [118]: start_cew_date = df_CEW['date'].values[0]
          search_cew_STdate = convert_pandasDate(start_cew_date)

          end_cew_date = df_CEW['date'].values[3626-1]
          search_cew_ENDdate = convert_pandasDate(end_cew_date)

          df_CEW['date'] = df_CEW['date'].apply(convert_pandasDate)
          #df_CEW['date'] = pd.to_datetime(df_CEW['date'])

          print(search_cew_STdate)
          print(search_cew_ENDdate)

          print("total weeeks " , compute_weeks(search_cew_STdate, search_cew_ENDdate))

2006-06-19 00:00:00
2018-10-15 00:00:00
total weeeks  643


In [185]: df_CEW['date'] = pd.to_datetime(df_CEW['date'])
```

2 Building dataframe for the pest ECB

```
In [120]: #dividing the dataframe based on each pests - pest ECB
          df_ECB = df.loc[['ECB'], ['farm', 'trap_count','year','date']]

In [121]: df_ECB.head()

Out[121]:              farm  trap_count  year      date
          pest
          ECB   Litchfield-W          16  2006  20060619
          ECB       Hollis-B           7  2006  20060619
          ECB        Mason-B          11  2006  20060619
          ECB   Litchfield-W          25  2006  20060626
          ECB       Hollis-B          31  2006  20060626

In [122]: df_ECB.tail()

Out[122]:                farm  trap_count  year      date
          pest
          ECB     Hollis-JL-Pl           0  2018  20180924
          ECB   Peterborough-R           0  2018  20180924
          ECB          Mason-B           0  2018  20180924
          ECB     NewIpswich-B           0  2018  20180924
          ECB        Milford-M           0  2018  20180924

In [123]: df_ECB.shape

Out[123]: (4979, 4)

In [124]: start_ecb_date = df_ECB['date'].values[0]
          search_ecb_STdate = convert_pandasDate(start_cew_date)

          end_ecb_date = df_ECB['date'].values[4979-1]
          search_ecb_ENDdate = convert_pandasDate(end_cew_date)

          df_ECB['date'] = df_ECB['date'].apply(convert_pandasDate)

          print(search_ecb_STdate)
          print(search_ecb_ENDdate)

          print("total weeeks " , compute_weeks(search_ecb_STdate, search_ecb_ENDdate))
```

```
2006-06-19 00:00:00
2018-10-15 00:00:00
total weeeks  643
```

```
In [188]: df_ECB['date'] = pd.to_datetime(df_ECB['date'])
```

3 Building dataframe for the pest FAW

```
In [215]: #dividing the dataframe based on each pests - pest FAW
          df_FAW = df.loc[['FAW'], ['farm', 'trap_count','year','date']]

In [216]: df_FAW.head()

Out[216]:              farm  trap_count  year      date
          pest
          FAW       Pelham-G           0  2006  20060619
          FAW   Litchfield-W           0  2006  20060619
          FAW   Litchfield-M           0  2006  20060619
          FAW    Merrimack-T           0  2006  20060619
          FAW       Hollis-L           0  2006  20060619

In [217]: df_FAW.tail()

Out[217]:            farm  trap_count  year      date
          pest
          FAW    Antrim-T           0  2018  20181001
          FAW   Milford-M           0  2018  20181001
          FAW   Hollis-B2           0  2018  20181008
          FAW    Hollis-L           2  2018  20181008
          FAW    Hollis-L           3  2018  20181015

In [218]: df_FAW.shape

Out[218]: (3829, 4)

In [219]: start_faw_date = df_FAW['date'].values[0]
          search_faw_STdate = convert_pandasDate(start_cew_date)

          end_faw_date = df_FAW['date'].values[3829-1]
          search_faw_ENDdate = convert_pandasDate(end_cew_date)

          df_FAW['date'] = df_FAW['date'].apply(convert_pandasDate)

          print(search_faw_STdate)
          print(search_faw_ENDdate)

          print("total weeeks " , compute_weeks(search_faw_STdate, search_faw_ENDdate))

2006-06-19 00:00:00
2018-10-15 00:00:00
total weeeks  643


In [220]: df_FAW['date'] = pd.to_datetime(df_FAW['date'])
```

4 NOAA DATA

```
In [130]: df_NOOA = pd.read_csv('DAW.csv', header=0)
```

```
/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785: DtypeWarning: Col
  interactivity=interactivity, compiler=compiler, result=result)
```

```
In [131]: df_NOOA.shape
```

```
Out[131]: (473979, 29)
```

```
In [132]: df_NOOA.columns
```

```
Out[132]: Index(['station', 'valid', 'tmpf', 'dwpf', 'relh', 'drct', 'sknt', 'p01i',
                 'alti', 'mslp', 'vsby', 'gust', 'skyc1', 'skyc2', 'skyc3', 'skyc4',
                 'skyl1', 'skyl2', 'skyl3', 'skyl4', 'wxcodes', 'ice_accretion_1hr',
                 'ice_accretion_3hr', 'ice_accretion_6hr', 'peak_wind_gust',
                 'peak_wind_drct', 'peak_wind_time', 'feel', 'metar'],
                dtype='object')
```

```
In [133]: df_equation  = df_NOOA[['tmpf','dwpf', 'drct','feel', 'valid']]
          df_equation.head()
```

```
Out[133]:     tmpf   dwpf  drct   feel             valid
          0  24.80  10.40  60.0  20.46  2006-01-01 00:38
          1  24.08  10.04  80.0  19.63  2006-01-01 00:51
          2  21.92  15.98  20.0  13.39  2006-01-01 01:51
          3  21.20  17.60  30.0  11.61  2006-01-01 02:15
          4  21.92  17.06  30.0  13.39  2006-01-01 02:51
```

```
In [134]: df_equation['valid'] = pd.to_datetime(df_equation['valid'])
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: 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
  """Entry point for launching an IPython kernel.
```

```
In [135]: print(search_cew_STdate)
          print(search_cew_ENDdate)
```

```
2006-06-19 00:00:00
2018-10-15 00:00:00
```

5 selecting weather dataframe within the date range - CEW Pest

In [136]:
```
#selecting the dataframe within the range - CEW Pest
#df_equation = df_equation.loc[(df_equation['valid'] >= search_st_date) & (df_equati
df_equation = df_equation.loc[(df_equation['valid'] >= search_cew_STdate) & (df_equat
```

In [137]:
```
#need to get all the columns from here and sum up fro every week
#select date and each column, and find the average for each column
df_tempf = df_equation[['tmpf','dwpf', 'drct','feel','valid']]
df_tempf["tmpf"] = df_tempf["tmpf"].fillna(0)
df_tempf["dwpf"] = df_tempf["dwpf"].fillna(0)
df_tempf["drct"] = df_tempf["drct"].fillna(0)
df_tempf["feel"] = df_tempf["feel"].fillna(0)
```

In [138]:
```
df_tempf.tail()
```

Out[138]:

	tmpf	dwpf	drct	feel	valid
341757	0.0	0.0	0.0	0.0	2018-10-14 23:35:00
341758	0.0	0.0	190.0	0.0	2018-10-14 23:40:00
341759	0.0	0.0	0.0	0.0	2018-10-14 23:50:00
341760	46.0	39.9	0.0	46.0	2018-10-14 23:51:00
341761	0.0	0.0	0.0	0.0	2018-10-14 23:55:00

In [139]:
```
#df_tempf
```

In [140]:
```
from datetime import date

def getweekly_temperature(df_tempf,CEW_weather_data_dic):

    week_counts = 0

    count = 0
    temp_index = 0

    count_day0 = 0
    count_day1 = 0
    count_day2 = 0
    count_day3 = 0
    count_day4 = 0
    count_day5 = 0
    count_day6= 0
    totaldays = 0

    temperature = 0
    dewTemp = 0
    winDir = 0
    feelTemp = 0

    #empt sets for all the values
```

```
lst_tempf = []
lst_dwpf = []
lst_drct = []
lst_feel = []
lst_date = []

for i , j in df_tempf.iterrows():

    if count ==0:
        row = (i,j)
        #print(row[1][1])
        #initial_date = row[1][1]
        initial_date = row[1][4]
        f_yyyy = str(initial_date)[0:4]
        f_mm = str(initial_date)[5:7]
        f_dd = str(initial_date)[8:10]
        f_date = date(int(f_yyyy),int(f_mm),int(f_dd))

    row_data = (i,j)
    #print(row_data)
    #summing up the temperature
    #temperature = row_data[1][0]
    temperature = temperature + row_data[1][0]
    dewTemp = dewTemp + row_data[1][1]
    winDir = winDir + row_data[1][2]
    feelTemp = feelTemp + row_data[1][3]

    #last_date = row_data[1][1]
    last_date = row_data[1][4]
    l_yyyy = str(last_date)[0:4]
    l_mm = str(last_date)[5:7]
    l_dd = str(last_date)[8:10]
    l_date = date(int(l_yyyy),int(l_mm),int(l_dd))

    #change in date
    delta = l_date - f_date
    #print (delta.days)
    if delta.days%7 == 0:
        week_counts = week_counts + 1

    if delta.days == 0:
        count_day0 = count_day0 + 1
    elif delta.days == 1:
        count_day1 = count_day1 + 1
    elif delta.days == 2:
        count_day2 = count_day2 + 1
    elif delta.days == 3:
        count_day3 = count_day3 + 1
```

```
elif delta.days == 4:
    count_day4 = count_day4 + 1
elif delta.days == 5:
    count_day5 = count_day5 + 1
elif delta.days == 6:
    count_day6 = count_day6 + 1
#elif delta.days == 7:
else:
    #compute average temperature
    totaldays = count_day0 + count_day1 + count_day2 + count_day3 + count_da

    avg_Temp = temperature/totaldays
    avg_dewTemp = dewTemp/totaldays
    avg_winDir = winDir/totaldays
    avg_feelTemp = feelTemp/totaldays

    #appending values to the list
    lst_tempf.append(avg_Temp)
    lst_dwpf.append(avg_dewTemp)
    lst_drct.append(avg_winDir)
    lst_feel.append(avg_feelTemp)

    lst_date.append(f_date)
    #print(f_date)

    #get the start date
    #put all into the date frame

    temp_index = temp_index + 1

    #flush the date
    f_date = l_date

    #flush days count
    count_day0 = 0
    count_day1 = 0
    count_day2 = 0
    count_day3 = 0
    count_day4 = 0
    count_day5 = 0
    count_day6 = 0

    #flushing all the records ---- temperature
    temperature = 0
    dewTemp = 0
    winDir = 0
    feelTemp = 0
```

```
                #initialize the repetation
                count = count + 1

            print("total temperature index ", temp_index)
            print("total weeks: ", week_counts)

            #assigning list to the dictonary
            CEW_weather_data_dic['tempf'] = lst_tempf
            CEW_weather_data_dic['dwpf'] = lst_dwpf
            CEW_weather_data_dic['drct'] = lst_drct
            CEW_weather_data_dic['feel'] = lst_feel
            CEW_weather_data_dic['date'] = lst_date

            #return CEW_weather_data_dic
            #print(df_tempf)
```

In [141]:
```
#calling getweekly_temperature
CEW_weather_data_dic = {}
getweekly_temperature(df_tempf,CEW_weather_data_dic)
```

```
total temperature index  642
total weeks:  47922
```

In [142]:
```
#creating data frame needed for equation
#print(CEW_weather_data_dic)
df_CEW_weather = pd.DataFrame(CEW_weather_data_dic)
```

In [143]:
```
df_CEW_weather.head()
```

Out[143]:

	tempf	dwpf	drct	feel	date
0	70.977613	63.913169	63.703704	71.301399	2006-06-19
1	71.243468	65.095887	84.233871	71.733347	2006-06-26
2	71.056485	59.203758	127.515152	71.263273	2006-07-03
3	71.830909	66.491782	59.745455	72.608255	2006-07-10
4	73.200175	66.515808	101.004367	74.150786	2006-07-17

In [144]:
```
df_CEW_weather.tail()
```

Out[144]:

	tempf	dwpf	drct	feel	date
637	6.936188	6.288330	156.129550	6.983009	2018-09-03
638	9.211018	8.905188	113.670949	9.217243	2018-09-10
639	7.760753	7.281021	110.056730	7.728195	2018-09-17
640	6.692446	6.180576	161.330935	6.548972	2018-09-24
641	7.799536	7.509856	134.674923	7.750764	2018-10-01

In [186]:
```
df_CEW['date'] = df_CEW['date'].dt.date
```

In [146]:
```
df_CEW_weather['date'] = pd.to_datetime(df_CEW_weather['date'])
```

In [147]:
```
df_CEW_weather['date'] = df_CEW_weather['date'].dt.date
```

In [148]:
```
df_CEW_final = df_CEW.merge(df_CEW_weather, on='date')
```

6 predicted tarp counts :::: CEW pests

```
In [149]: df_CEW_final.columns
```

```
Out[149]: Index(['farm', 'trap_count', 'year', 'date', 'tempf', 'dwpf', 'drct', 'feel'], dtype=
```

```
In [150]: #creating training and testign dataset
          mask = np.random.rand(len(df_CEW_final)) < 0.8
          df_train_CEW = df_CEW_final[mask]
          df_test_CEW = df_CEW_final[~mask]
```

```
In [151]: print(len(df_train_CEW))
          print(len(df_test_CEW))
```

```
2053
518
```

```
In [152]: expr_CEW = """trap_count ~ tempf + dwpf + drct + feel"""
```

```
In [153]: #Set up the X and y matrices
          y_train, X_train = dmatrices(expr_CEW, df_CEW_final, return_type='dataframe')
          y_test, X_test = dmatrices(expr_CEW, df_CEW_final, return_type='dataframe')
```

```
In [154]: #X_test
```

```
In [155]: #Using the statsmodels GLM class, train the Poisson regression model on the training
          poisson_training_results = sm.GLM(y_train, X_train, family=sm.families.Poisson()).fi
```

```
In [156]: print(poisson_training_results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:             trap_count   No. Observations:                 2571
Model:                            GLM   Df Residuals:                     2566
Model Family:                 Poisson   Df Model:                            4
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -41041.
Date:                Sun, 05 Apr 2020   Deviance:                       75750.
Time:                        22:59:34   Pearson chi2:                 1.55e+05
No. Iterations:                     6   Covariance Type:             nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.2900      0.042     54.923      0.000       2.208       2.372
tempf         -0.0907      0.008    -12.012      0.000      -0.106      -0.076
dwpf          -0.0165      0.003     -6.496      0.000      -0.021      -0.012
drct           0.0014      0.000      5.331      0.000       0.001       0.002
feel           0.1047      0.007     14.292      0.000       0.090       0.119
==============================================================================
```

```
In [157]: #Make some predictions on the test data set.
          poisson_predictions = poisson_training_results.get_prediction(X_test)

In [158]: #.summary_frame() returns a pandas DataFrame
          predictions_summary_frame = poisson_predictions.summary_frame()
          print(predictions_summary_frame)
```

	mean	mean_se	mean_ci_lower	mean_ci_upper
0	10.502287	0.121299	10.267216	10.742741
1	10.502287	0.121299	10.267216	10.742741
2	10.502287	0.121299	10.267216	10.742741
3	10.502287	0.121299	10.267216	10.742741
4	10.502287	0.121299	10.267216	10.742741
5	10.502287	0.121299	10.267216	10.742741
6	10.502287	0.121299	10.267216	10.742741
7	10.502287	0.121299	10.267216	10.742741
8	10.502287	0.121299	10.267216	10.742741
9	10.502287	0.121299	10.267216	10.742741
10	10.502287	0.121299	10.267216	10.742741
11	10.824663	0.103944	10.622843	11.030318
12	10.824663	0.103944	10.622843	11.030318
13	10.824663	0.103944	10.622843	11.030318
14	10.824663	0.103944	10.622843	11.030318
15	10.824663	0.103944	10.622843	11.030318
16	10.824663	0.103944	10.622843	11.030318
17	10.824663	0.103944	10.622843	11.030318
18	10.824663	0.103944	10.622843	11.030318
19	10.824663	0.103944	10.622843	11.030318
20	10.824663	0.103944	10.622843	11.030318
21	10.824663	0.103944	10.622843	11.030318
22	12.268939	0.116009	12.043660	12.498433
23	12.268939	0.116009	12.043660	12.498433
24	12.268939	0.116009	12.043660	12.498433
25	12.268939	0.116009	12.043660	12.498433
26	12.268939	0.116009	12.043660	12.498433
27	12.268939	0.116009	12.043660	12.498433
28	12.268939	0.116009	12.043660	12.498433
29	12.268939	0.116009	12.043660	12.498433
...	...	...	...	...
2541	11.342433	0.197355	10.962146	11.735912
2542	11.342433	0.197355	10.962146	11.735912
2543	11.342433	0.197355	10.962146	11.735912
2544	12.080906	0.207162	11.681623	12.493836
2545	12.080906	0.207162	11.681623	12.493836
2546	12.080906	0.207162	11.681623	12.493836
2547	12.080906	0.207162	11.681623	12.493836
2548	12.080906	0.207162	11.681623	12.493836
2549	12.080906	0.207162	11.681623	12.493836

```
2550  12.080906  0.207162      11.681623      12.493836
2551  12.080906  0.207162      11.681623      12.493836
2552  12.080906  0.207162      11.681623      12.493836
2553  12.080906  0.207162      11.681623      12.493836
2554  12.080906  0.207162      11.681623      12.493836
2555  12.080906  0.207162      11.681623      12.493836
2556  12.080906  0.207162      11.681623      12.493836
2557  12.080906  0.207162      11.681623      12.493836
2558  12.080906  0.207162      11.681623      12.493836
2559  12.080906  0.207162      11.681623      12.493836
2560  12.080906  0.207162      11.681623      12.493836
2561  11.680781  0.185481      11.322844      12.050034
2562  11.680781  0.185481      11.322844      12.050034
2563  11.680781  0.185481      11.322844      12.050034
2564  11.680781  0.185481      11.322844      12.050034
2565  11.680781  0.185481      11.322844      12.050034
2566  11.680781  0.185481      11.322844      12.050034
2567  11.680781  0.185481      11.322844      12.050034
2568  11.680781  0.185481      11.322844      12.050034
2569  11.680781  0.185481      11.322844      12.050034
2570  11.680781  0.185481      11.322844      12.050034

[2571 rows x 4 columns]
```

```
In [159]: predicted_counts=predictions_summary_frame['mean']
          actual_counts = y_test['trap_count']

In [160]: predicted_counts

Out[160]: 0       10.502287
          1       10.502287
          2       10.502287
          3       10.502287
          4       10.502287
          5       10.502287
          6       10.502287
          7       10.502287
          8       10.502287
          9       10.502287
          10      10.502287
          11      10.824663
          12      10.824663
          13      10.824663
          14      10.824663
          15      10.824663
          16      10.824663
          17      10.824663
```

```
18      10.824663
19      10.824663
20      10.824663
21      10.824663
22      12.268939
23      12.268939
24      12.268939
25      12.268939
26      12.268939
27      12.268939
28      12.268939
29      12.268939
          ...
2541    11.342433
2542    11.342433
2543    11.342433
2544    12.080906
2545    12.080906
2546    12.080906
2547    12.080906
2548    12.080906
2549    12.080906
2550    12.080906
2551    12.080906
2552    12.080906
2553    12.080906
2554    12.080906
2555    12.080906
2556    12.080906
2557    12.080906
2558    12.080906
2559    12.080906
2560    12.080906
2561    11.680781
2562    11.680781
2563    11.680781
2564    11.680781
2565    11.680781
2566    11.680781
2567    11.680781
2568    11.680781
2569    11.680781
2570    11.680781
Name: mean, Length: 2571, dtype: float64
```

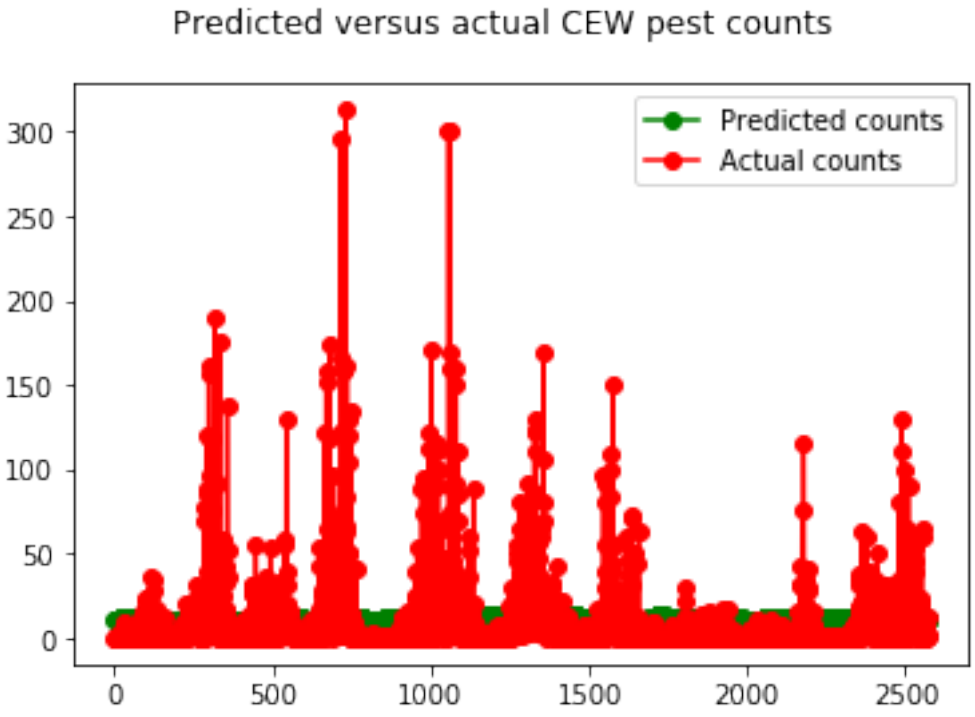
```
In [161]: print(actual_counts)
```

```
0       0.0
1       0.0
```

```
2        0.0
3        0.0
4        1.0
5        0.0
6        0.0
7        0.0
8        0.0
9        0.0
10       0.0
11       0.0
12       0.0
13       0.0
14       0.0
15       1.0
16       0.0
17       0.0
18       0.0
19       0.0
20       0.0
21       0.0
22       0.0
23       3.0
24       0.0
25       0.0
26       4.0
27      10.0
28       1.0
29       3.0
        ...
2541     0.0
2542     2.0
2543     5.0
2544    21.0
2545    10.0
2546     8.0
2547    20.0
2548    12.0
2549     8.0
2550    23.0
2551    60.0
2552    65.0
2553     1.0
2554     1.0
2555     5.0
2556     4.0
2557     3.0
2558     1.0
2559     2.0
```

```
2560     1.0
2561     6.0
2562     2.0
2563     0.0
2564     0.0
2565     8.0
2566     2.0
2567     6.0
2568    12.0
2569     1.0
2570     1.0
Name: trap_count, Length: 2571, dtype: float64
```

In [162]:
```
#Mlot the predicted counts versus the actual counts for the test data.
fig = plt.figure()
fig.suptitle('Predicted versus actual CEW pest counts ')
predicted, = plt.plot(X_test.index, predicted_counts, 'go-', label='Predicted counts
actual, = plt.plot(X_test.index, actual_counts, 'ro-', label='Actual counts')
plt.legend(handles=[predicted, actual])
plt.show()
```



In [163]:
```
#writing dataframe into csv file
```

In [164]:
```
df_CEW_final['predicted_counts'] = predicted_counts
```

```
In [195]: #df_CEW_final

In [166]: #writing into the csv file
          df_CEW_final.to_csv("CEW_predicted_count.csv", index = False, sep = ',')
```

7 selecting weather dataframe within the date range - ECB Pest

```
In [177]: print(search_ecb_STdate)
          print(search_ecb_ENDdate)
```

```
2006-06-19 00:00:00
2018-10-15 00:00:00
```

```
In [178]: df_equation_ECB = df_equation.loc[(df_equation['valid'] >= search_ecb_STdate) & (df_e

In [179]: #need to get all the columns from here and sum up fro every week
          #select date and each column, and find the average for each column
          df_tempf_ECB = df_equation_ECB[['tmpf','dwpf', 'drct','feel','valid']]
          df_tempf_ECB["tmpf"] = df_tempf_ECB["tmpf"].fillna(0)
          df_tempf_ECB["dwpf"] = df_tempf_ECB["dwpf"].fillna(0)
          df_tempf_ECB["drct"] = df_tempf_ECB["drct"].fillna(0)
          df_tempf_ECB["feel"] = df_tempf_ECB["feel"].fillna(0)

In [180]: #calling getweekly_temperature
          ECB_weather_data_dic = {}
          getweekly_temperature(df_tempf_ECB,ECB_weather_data_dic)
```

```
total temperature index  642
total weeks:  47922
```

```
In [182]: df_ECB_weather = pd.DataFrame(ECB_weather_data_dic)

In [189]: df_ECB['date'] = df_ECB['date'].dt.date

In [190]: #combining weather data and pest count data based on the date -----
          #creating data frame needed for equation
          df_ECB_weather['date'] = pd.to_datetime(df_ECB_weather['date'])
          df_ECB_weather['date'] = df_ECB_weather['date'].dt.date
          df_ECB_final = df_ECB.merge(df_ECB_weather, on='date')

In [191]: df_ECB_final.head()
```

Out[191]:

	farm	trap_count	year	date	tempf	dwpf	\
0	Litchfield-W	16	2006	2006-06-19	70.977613	63.913169	
1	Hollis-B	7	2006	2006-06-19	70.977613	63.913169	
2	Mason-B	11	2006	2006-06-19	70.977613	63.913169	
3	Litchfield-W	25	2006	2006-06-26	71.243468	65.095887	

```
4      Hollis-B          31  2006  2006-06-26  71.243468  65.095887

         drct       feel
0  63.703704  71.301399
1  63.703704  71.301399
2  63.703704  71.301399
3  84.233871  71.733347
4  84.233871  71.733347
```

8 predicting trap counts :::: ECB pest

```
In [192]: #creating training and testign dataset
          mask_ecb = np.random.rand(len(df_ECB_final)) < 0.8
          df_train_ECB = df_ECB_final[mask_ecb]
          df_test_ECB = df_ECB_final[~mask_ecb]

In [194]: expr_ECB = """trap_count ~ tempf + dwpf + drct + feel"""

In [197]: #Set up the X and y matrices
          y_train_ecb, X_train_ecb = dmatrices(expr_ECB, df_ECB_final, return_type='dataframe')
          y_test_ecb, X_test_ecb = dmatrices(expr_ECB, df_ECB_final, return_type='dataframe')

In [199]: #Using the statsmodels GLM class, train the Poisson regression model on the training
          poisson_training_results_ecb = sm.GLM(y_train_ecb, X_train_ecb, family=sm.families.Po

In [200]: print(poisson_training_results_ecb.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:             trap_count   No. Observations:                 4020
Model:                            GLM   Df Residuals:                     4015
Model Family:                 Poisson   Df Model:                            4
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -20982.
Date:                Sun, 05 Apr 2020   Deviance:                       36540.
Time:                        23:28:44   Pearson chi2:                 8.28e+04
No. Iterations:                     6   Covariance Type:             nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.2365      0.070     17.763      0.000       1.100       1.373
tempf         -0.0035      0.004     -0.925      0.355      -0.011       0.004
dwpf           0.0308      0.004      7.917      0.000       0.023       0.038
drct          -0.0022      0.000     -5.133      0.000      -0.003      -0.001
feel          -0.0221      0.002    -10.177      0.000      -0.026      -0.018
==============================================================================
```

In [202]:
```
#Make some predictions on the test data set.
poisson_predictions_ecb = poisson_training_results_ecb.get_prediction(X_test_ecb)
```

In [203]:
```
#.summary_frame() returns a pandas DataFrame
predictions_summary_frame_ecb = poisson_predictions_ecb.summary_frame()
print(predictions_summary_frame_ecb)
```

```
          mean   mean_se  mean_ci_lower  mean_ci_upper
0     3.484824  0.060365       3.368497       3.605168
1     3.484824  0.060365       3.368497       3.605168
2     3.484824  0.060365       3.368497       3.605168
3     3.420675  0.049845       3.324362       3.519779
4     3.420675  0.049845       3.324362       3.519779
5     3.420675  0.049845       3.324362       3.519779
6     2.625314  0.040086       2.547910       2.705069
7     2.625314  0.040086       2.547910       2.705069
8     2.625314  0.040086       2.547910       2.705069
9     3.686626  0.071031       3.550004       3.828506
10    3.686626  0.071031       3.550004       3.828506
11    3.686626  0.071031       3.550004       3.828506
12    3.244769  0.050942       3.146445       3.346165
13    3.244769  0.050942       3.146445       3.346165
14    3.244769  0.050942       3.146445       3.346165
15    3.111367  0.057636       3.000428       3.226408
16    3.111367  0.057636       3.000428       3.226408
17    3.111367  0.057636       3.000428       3.226408
18    2.909125  0.040040       2.831697       2.988670
19    2.909125  0.040040       2.831697       2.988670
20    2.909125  0.040040       2.831697       2.988670
21    2.155955  0.067482       2.027669       2.292358
22    2.155955  0.067482       2.027669       2.292358
23    2.155955  0.067482       2.027669       2.292358
24    2.958304  0.050285       2.861370       3.058521
25    2.958304  0.050285       2.861370       3.058521
26    2.958304  0.050285       2.861370       3.058521
27    3.267339  0.039274       3.191264       3.345228
28    3.267339  0.039274       3.191264       3.345228
29    3.267339  0.039274       3.191264       3.345228
...        ...       ...            ...            ...
3990  2.797668  0.081575       2.642266       2.962210
3991  2.797668  0.081575       2.642266       2.962210
3992  2.797668  0.081575       2.642266       2.962210
3993  2.797668  0.081575       2.642266       2.962210
3994  2.797668  0.081575       2.642266       2.962210
3995  2.797668  0.081575       2.642266       2.962210
3996  2.785602  0.085558       2.622859       2.958443
3997  2.785602  0.085558       2.622859       2.958443
3998  2.785602  0.085558       2.622859       2.958443
```

```
3999  2.785602  0.085558       2.622859       2.958443
4000  2.785602  0.085558       2.622859       2.958443
4001  2.785602  0.085558       2.622859       2.958443
4002  2.785602  0.085558       2.622859       2.958443
4003  2.785602  0.085558       2.622859       2.958443
4004  2.785602  0.085558       2.622859       2.958443
4005  2.785602  0.085558       2.622859       2.958443
4006  2.785602  0.085558       2.622859       2.958443
4007  2.785602  0.085558       2.622859       2.958443
4008  2.785602  0.085558       2.622859       2.958443
4009  2.785602  0.085558       2.622859       2.958443
4010  2.785602  0.085558       2.622859       2.958443
4011  2.785602  0.085558       2.622859       2.958443
4012  2.785602  0.085558       2.622859       2.958443
4013  2.481624  0.074581       2.339670       2.632191
4014  2.481624  0.074581       2.339670       2.632191
4015  2.481624  0.074581       2.339670       2.632191
4016  2.481624  0.074581       2.339670       2.632191
4017  2.481624  0.074581       2.339670       2.632191
4018  2.481624  0.074581       2.339670       2.632191
4019  2.481624  0.074581       2.339670       2.632191

[4020 rows x 4 columns]
```

```
In [204]: predicted_counts_ecb=predictions_summary_frame_ecb['mean']
          actual_counts_ecb = y_test_ecb['trap_count']
```

```
In [241]: print(predicted_counts_ecb)
```

```
0       3.484824
1       3.484824
2       3.484824
3       3.420675
4       3.420675
5       3.420675
6       2.625314
7       2.625314
8       2.625314
9       3.686626
10      3.686626
11      3.686626
12      3.244769
13      3.244769
14      3.244769
15      3.111367
16      3.111367
17      3.111367
```

```
18      2.909125
19      2.909125
20      2.909125
21      2.155955
22      2.155955
23      2.155955
24      2.958304
25      2.958304
26      2.958304
27      3.267339
28      3.267339
29      3.267339
          ...
3990    2.797668
3991    2.797668
3992    2.797668
3993    2.797668
3994    2.797668
3995    2.797668
3996    2.785602
3997    2.785602
3998    2.785602
3999    2.785602
4000    2.785602
4001    2.785602
4002    2.785602
4003    2.785602
4004    2.785602
4005    2.785602
4006    2.785602
4007    2.785602
4008    2.785602
4009    2.785602
4010    2.785602
4011    2.785602
4012    2.785602
4013    2.481624
4014    2.481624
4015    2.481624
4016    2.481624
4017    2.481624
4018    2.481624
4019    2.481624
Name: mean, Length: 4020, dtype: float64
```

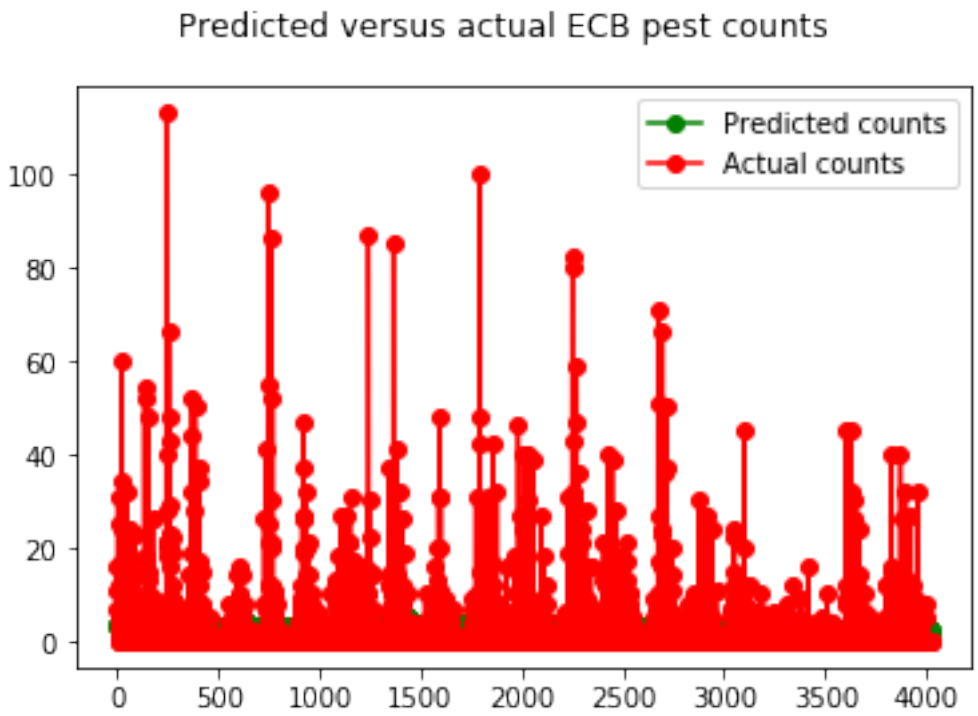
```
In [206]: print(actual_counts_ecb)
```

```
0       16.0
```

```
1        7.0
2       11.0
3       25.0
4       31.0
5        7.0
6        3.0
7        4.0
8        1.0
9       12.0
10       0.0
11       0.0
12       4.0
13       3.0
14       0.0
15       5.0
16      24.0
17       1.0
18      13.0
19      34.0
20       5.0
21      17.0
22      60.0
23       1.0
24       1.0
25      12.0
26       0.0
27       1.0
28       2.0
29       1.0
        ...
3990     0.0
3991     0.0
3992     0.0
3993     0.0
3994     0.0
3995     0.0
3996     0.0
3997     5.0
3998     0.0
3999     0.0
4000     0.0
4001     0.0
4002     0.0
4003     8.0
4004     0.0
4005     0.0
4006     0.0
4007     0.0
```

```
4008     0.0
4009     0.0
4010     0.0
4011     0.0
4012     0.0
4013     0.0
4014     0.0
4015     0.0
4016     0.0
4017     0.0
4018     0.0
4019     0.0
Name: trap_count, Length: 4020, dtype: float64
```

In [208]:
```
#Mlot the predicted counts versus the actual counts for the test data.
fig = plt.figure()
fig.suptitle('Predicted versus actual ECB pest counts ')
predicted, = plt.plot(X_test_ecb.index, predicted_counts_ecb, 'go-', label='Predicte
actual, = plt.plot(X_test_ecb.index, actual_counts_ecb, 'ro-', label='Actual counts'
plt.legend(handles=[predicted, actual])
plt.show()
```



In [210]:
```
df_ECB_final['predicted_counts_ecb'] = predicted_counts_ecb
```

```
In [211]: #writing into the csv file
          df_ECB_final.to_csv("ECB_predicted_count.csv", index = False, sep = ',')
```

9 selecting weather dataframe within the date range - FAW Pest

```
In [224]: print(search_faw_STdate)
          print(search_faw_ENDdate)
```

```
2006-06-19 00:00:00
2018-10-15 00:00:00
```

```
In [225]: df_equation_FAW = df_equation.loc[(df_equation['valid'] >= search_faw_STdate) & (df_e
```

```
In [226]: #need to get all the columns from here and sum up fro every week
          #select date and each column, and find the average for each column
          df_tempf_FAW = df_equation_FAW[['tmpf','dwpf', 'drct','feel','valid']]
          df_tempf_FAW["tmpf"] = df_tempf_FAW["tmpf"].fillna(0)
          df_tempf_FAW["dwpf"] = df_tempf_FAW["dwpf"].fillna(0)
          df_tempf_FAW["drct"] = df_tempf_FAW["drct"].fillna(0)
          df_tempf_FAW["feel"] = df_tempf_FAW["feel"].fillna(0)
```

```
In [227]: #calling getweekly_temperature
          FAW_weather_data_dic = {}
          getweekly_temperature(df_tempf_FAW,FAW_weather_data_dic)
```

```
total temperature index  642
total weeks:  47922
```

```
In [228]: df_FAW_weather = pd.DataFrame(FAW_weather_data_dic)
```

```
In [229]: #convert orginal faw date into correct dataframe date
          df_FAW['date'] = df_FAW['date'].dt.date
```

```
In [230]: #combining weather data and pest count data based on the date -----
          #creating data frame needed for equation
          df_FAW_weather['date'] = pd.to_datetime(df_FAW_weather['date'])
          df_FAW_weather['date'] = df_FAW_weather['date'].dt.date
          df_FAW_final = df_FAW.merge(df_FAW_weather, on='date')
```

```
In [232]: df_FAW_final.head()
```

Out[232]:

	farm	trap_count	year	date	tempf	dwpf
0	Pelham-G	0	2006	2006-06-19	70.977613	63.913169
1	Litchfield-W	0	2006	2006-06-19	70.977613	63.913169
2	Litchfield-M	0	2006	2006-06-19	70.977613	63.913169
3	Merrimack-T	0	2006	2006-06-19	70.977613	63.913169
4	Hollis-L	0	2006	2006-06-19	70.977613	63.913169

```
        drct       feel
0  63.703704  71.301399
1  63.703704  71.301399
2  63.703704  71.301399
3  63.703704  71.301399
4  63.703704  71.301399
```

10 predicting trap counts :::: FAW pest

In [233]:
```
#creating training and testign dataset
mask_faw = np.random.rand(len(df_FAW_final)) < 0.8
df_train_FAW = df_FAW_final[mask_faw]
df_test_FAW = df_FAW_final[~mask_faw]
```

In [234]:
```
expr_FAW = """trap_count ~ tempf + dwpf + drct + feel"""
```

In [235]:
```
#Set up the X and y matrices
y_train_faw, X_train_faw = dmatrices(expr_FAW, df_FAW_final, return_type='dataframe')
y_test_faw, X_test_faw = dmatrices(expr_FAW, df_FAW_final, return_type='dataframe')
```

In [236]:
```
#Using the statsmodels GLM class, train the Poisson regression model on the training
poisson_training_results_faw = sm.GLM(y_train_faw, X_train_faw, family=sm.families.Po
```

In [237]:
```
print(poisson_training_results_faw.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:             trap_count   No. Observations:                 2634
Model:                            GLM   Df Residuals:                     2629
Model Family:                 Poisson   Df Model:                            4
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -13336.
Date:                Mon, 06 Apr 2020   Deviance:                       23704.
Time:                        08:28:51   Pearson chi2:                 7.29e+04
No. Iterations:                     6   Covariance Type:             nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.0710      0.084     12.699      0.000       0.906       1.236
tempf          0.0774      0.011      7.154      0.000       0.056       0.099
dwpf          -0.1255      0.005    -25.234      0.000      -0.135      -0.116
drct          -0.0004      0.001     -0.773      0.440      -0.001       0.001
feel           0.0298      0.010      2.952      0.003       0.010       0.050
==============================================================================
```

In [238]:
```
#Make some predictions on the test data set.
poisson_predictions_faw = poisson_training_results_faw.get_prediction(X_test_faw)
```

```
In [239]: #.summary_frame() returns a pandas DataFrame
          predictions_summary_frame_faw = poisson_predictions_faw.summary_frame()
          print(predictions_summary_frame_faw)
```

	mean	mean_se	mean_ci_lower	mean_ci_upper
0	1.901839	0.048888	1.808393	2.000113
1	1.901839	0.048888	1.808393	2.000113
2	1.901839	0.048888	1.808393	2.000113
3	1.901839	0.048888	1.808393	2.000113
4	1.901839	0.048888	1.808393	2.000113
5	1.901839	0.048888	1.808393	2.000113
6	1.901839	0.048888	1.808393	2.000113
7	1.901839	0.048888	1.808393	2.000113
8	1.901839	0.048888	1.808393	2.000113
9	1.901839	0.048888	1.808393	2.000113
10	1.901839	0.048888	1.808393	2.000113
11	1.681121	0.038086	1.608107	1.757450
12	1.681121	0.038086	1.608107	1.757450
13	1.681121	0.038086	1.608107	1.757450
14	1.681121	0.038086	1.608107	1.757450
15	1.681121	0.038086	1.608107	1.757450
16	1.681121	0.038086	1.608107	1.757450
17	1.681121	0.038086	1.608107	1.757450
18	1.681121	0.038086	1.608107	1.757450
19	1.681121	0.038086	1.608107	1.757450
20	1.681121	0.038086	1.608107	1.757450
21	1.681121	0.038086	1.608107	1.757450
22	3.362671	0.057343	3.252139	3.476960
23	3.362671	0.057343	3.252139	3.476960
24	3.362671	0.057343	3.252139	3.476960
25	3.362671	0.057343	3.252139	3.476960
26	3.362671	0.057343	3.252139	3.476960
27	3.362671	0.057343	3.252139	3.476960
28	3.362671	0.057343	3.252139	3.476960
29	3.362671	0.057343	3.252139	3.476960
...	...	...	...	...
2604	2.568579	0.090812	2.396617	2.752880
2605	2.568579	0.090812	2.396617	2.752880
2606	2.566849	0.089726	2.396879	2.748873
2607	2.566849	0.089726	2.396879	2.748873
2608	2.566849	0.089726	2.396879	2.748873
2609	2.566849	0.089726	2.396879	2.748873
2610	2.566849	0.089726	2.396879	2.748873
2611	2.566849	0.089726	2.396879	2.748873
2612	2.566849	0.089726	2.396879	2.748873
2613	2.566849	0.089726	2.396879	2.748873
2614	2.566849	0.089726	2.396879	2.748873
2615	2.566849	0.089726	2.396879	2.748873

```
2616  2.566849  0.089726       2.396879       2.748873
2617  2.566849  0.089726       2.396879       2.748873
2618  2.566849  0.089726       2.396879       2.748873
2619  2.566849  0.089726       2.396879       2.748873
2620  2.566849  0.089726       2.396879       2.748873
2621  2.566849  0.089726       2.396879       2.748873
2622  2.566849  0.089726       2.396879       2.748873
2623  2.566849  0.089726       2.396879       2.748873
2624  2.479989  0.080162       2.327747       2.642188
2625  2.479989  0.080162       2.327747       2.642188
2626  2.479989  0.080162       2.327747       2.642188
2627  2.479989  0.080162       2.327747       2.642188
2628  2.479989  0.080162       2.327747       2.642188
2629  2.479989  0.080162       2.327747       2.642188
2630  2.479989  0.080162       2.327747       2.642188
2631  2.479989  0.080162       2.327747       2.642188
2632  2.479989  0.080162       2.327747       2.642188
2633  2.479989  0.080162       2.327747       2.642188

[2634 rows x 4 columns]
```

```
In [240]: predicted_counts_faw=predictions_summary_frame_faw['mean']
          actual_counts_faw = y_test_faw['trap_count']

In [242]: print(predicted_counts_faw)
```

```
0       1.901839
1       1.901839
2       1.901839
3       1.901839
4       1.901839
5       1.901839
6       1.901839
7       1.901839
8       1.901839
9       1.901839
10      1.901839
11      1.681121
12      1.681121
13      1.681121
14      1.681121
15      1.681121
16      1.681121
17      1.681121
18      1.681121
19      1.681121
20      1.681121
```

```
21      1.681121
22      3.362671
23      3.362671
24      3.362671
25      3.362671
26      3.362671
27      3.362671
28      3.362671
29      3.362671
          ...
2604    2.568579
2605    2.568579
2606    2.566849
2607    2.566849
2608    2.566849
2609    2.566849
2610    2.566849
2611    2.566849
2612    2.566849
2613    2.566849
2614    2.566849
2615    2.566849
2616    2.566849
2617    2.566849
2618    2.566849
2619    2.566849
2620    2.566849
2621    2.566849
2622    2.566849
2623    2.566849
2624    2.479989
2625    2.479989
2626    2.479989
2627    2.479989
2628    2.479989
2629    2.479989
2630    2.479989
2631    2.479989
2632    2.479989
2633    2.479989
Name: mean, Length: 2634, dtype: float64
```

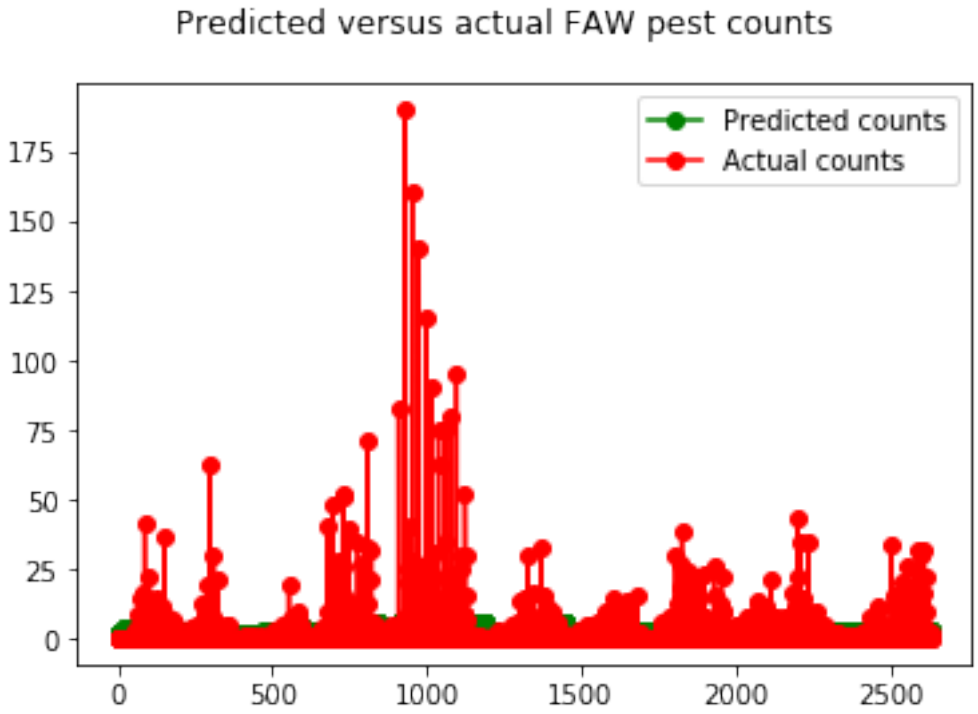
```
In [243]: print(actual_counts_faw)
```

```
0        0.0
1        0.0
2        0.0
```

```
3        0.0
4        0.0
5        0.0
6        0.0
7        0.0
8        0.0
9        0.0
10       0.0
11       0.0
12       0.0
13       0.0
14       0.0
15       0.0
16       0.0
17       0.0
18       0.0
19       0.0
20       0.0
21       0.0
22       0.0
23       0.0
24       0.0
25       0.0
26       0.0
27       0.0
28       0.0
29       0.0
        ...
2604     1.0
2605     0.0
2606     5.0
2607    16.0
2608    32.0
2609     0.0
2610     2.0
2611     0.0
2612     0.0
2613     2.0
2614    22.0
2615    10.0
2616     0.0
2617     0.0
2618     0.0
2619     1.0
2620     2.0
2621     0.0
2622     1.0
2623     0.0
```

```
2624     0.0
2625     2.0
2626     0.0
2627     0.0
2628     0.0
2629     2.0
2630     0.0
2631     2.0
2632     0.0
2633     0.0
Name: trap_count, Length: 2634, dtype: float64
```

In [245]:
```
#Mlot the predicted counts versus the actual counts for the test data.
fig = plt.figure()
fig.suptitle('Predicted versus actual FAW pest counts ')
predicted, = plt.plot(X_test_faw.index, predicted_counts_faw, 'go-', label='Predicte
actual, = plt.plot(X_test_faw.index, actual_counts_faw, 'ro-', label='Actual counts'
plt.legend(handles=[predicted, actual])
plt.show()
```



In [246]:
```
df_FAW_final['predicted_counts_faw'] = predicted_counts_faw
```

In [247]:
```
#writing into the csv file
df_FAW_final.to_csv("FAW_predicted_count.csv", index = False, sep = ',')
```