Hurricane

November 4, 2023

1 Data import and continued cleaning

[31]:
```
%%bash
head -n 2 Hurricanes.csv
```

```
"ID";"Date";"Time"
"AL011851";"1851-06-25 00:00:00";"0"
```

[32]:
```
import csv

#read the csv into the variables 'header' and 'data'
with open("Hurricanes.csv",mode='r') as csvFile:
    reader = csv.reader(csvFile,delimiter=';',skipinitialspace=True)
    header = next(reader)
    #data = [i for i in reader]
    data = list(reader)
```

[33]:
```
print(header)
```

```
['ID', 'Date', 'Time']
```

[34]:
```
from datetime import datetime, timedelta
format_string = "%Y-%m-%d %H:%M:%S"
print(datetime.strptime(data[0][1],format_string))
```

```
1851-06-25 00:00:00
```

[35]:
```
#Make a variable 'origin_date' which is a datetime object instead of just a␣
↪string
origin_date = [datetime.strptime(line[1],format_string) for line in data]
```

[36]:
```
#Let's print all the unique entries of time.
times = []
for i in range(0,len(data)):
    times.append(data[i][2])
set_times = set(times)
print(set_times)
```

```
{'100', '600', '1700', '900', '1645', '230', '1830', '1200', '1930', '0',
'2315', '1500', '800', '700', '2000', '1630', '1000', '1800', '730', '300',
'1400', '2100', '1600', '2200', '1900', '1300', '1100', '2300'}
```

These are the unique times. We want to combine these times with the origin_date.

[37]:
```python
origin_date_time = []
for i in range(0,len(origin_date)):
    #hour and minute to add to origin_date
    h = 0
    m = 0

    #Looking at the unique times from the printout of the earlier cell we see␣
 ↪that the length of the time string is
    # either 1, 3 or 4. We handle these cases seperately.
    #Note that data[i][2][1:3] is the i:th entry in data, the second column␣
 ↪which is the time string and [1:3]
    # gives letter number 1 and letter number 2 but not letter number 3.
    if(len(data[i][2]) == 3):
        h = int(data[i][2][0])
        m = int(data[i][2][1:3])
    if(len(data[i][2]) == 4):
        h = int(data[i][2][0:2])
        m = int(data[i][2][2:4])
    time_change = timedelta(hours=h,minutes=m)
    origin_date_time.append(origin_date[i] + time_change)
```

[38]:
```python
#print the origin_date_time for the first 10 recorded hurricanes.
for i in range(0,10):
    print(origin_date_time[i])
```

```
1851-06-25 00:00:00
1851-07-05 12:00:00
1851-07-10 12:00:00
1851-08-16 00:00:00
1851-09-13 00:00:00
1851-10-16 12:00:00
1852-08-29 12:00:00
1852-09-06 06:00:00
1852-09-13 12:00:00
1852-09-26 00:00:00
```

2 Standard Poisson

When modelling occurances of (presumably) independent events we often make assumptions that result in the time between events being exponentially distributed and the number of events in a certain time frame being poisson distributed. Let's try to do that since it is a standard approach and then evaluate. Let $X_i :=$ the time from hurricane number i to hurricane number $i+1$ and let

n be the number of such X_i. We assume: * $\{X_i\}$ is independent and identically distributed (i.i.d) * The rate of occurance (λ) doesn't change over time.

The second assumption means that $X_i \sim Exp(\lambda)$. This means our model space would be $\mathcal{M} = \{f_\lambda(x) = \lambda e^{-\lambda x} : \lambda > 0\}$. A commonly used loss function for the exponential distribution is log-loss $L(f_\lambda, x) = -\ln(f_\lambda(x))$. The risk is the expected loss $R(f_\lambda) = E[L(f_\lambda, X)]$. Let x_i be the observed value of X_i. Now, because of our assumption of independence, the empirical risk

$$\hat{R}(f_\lambda) = \frac{1}{n}\sum_{i=1}^{n} L(f_\lambda, x_i)$$

It is the empirical risk that we wish to minimize with respect to λ. We therefore differentiate and find the λ where the derivative is 0.

$$\frac{d}{d\lambda}\hat{R}(f_\lambda) = \frac{1}{n}\sum_{i=1}^{n}(x_i - 1/\lambda) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) - \frac{1}{\lambda}$$

We see that

$$\hat{\lambda} = 1/\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{n}{\sum x_i}$$

gives us a 0 derivative. It is also relatively straightforward to see that smaller λ result in a positive derivative (of the risk function) and larger λ results in a negative derivative meaning we have found a minimum.

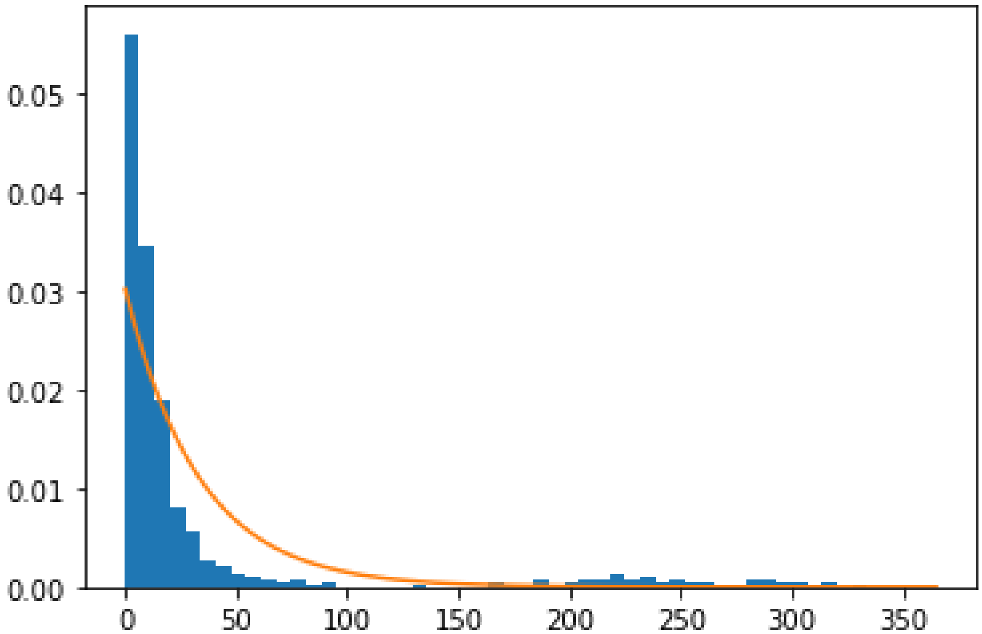
[39]:
```
time_differences = []
#x_i = time_differences[i]
seconds_in_a_day = 60*60*24
for i in range(0,(len(origin_date_time)-1)):
        time_differences.append((origin_date_time[i+1]-origin_date_time[i]).
 ↪total_seconds()/seconds_in_a_day)
#time_differences are now represented as the number of days but as a float.
```

[40]:
```
import numpy as np
lambda_empirical = 1/np.mean(time_differences)
```

Now we have our best estimate for λ. Now it's time to plot the real data compared to our model.

[41]:
```
import matplotlib.pyplot as plt
_=plt.hist(time_differences,bins=50,density=True)
x_plot = np.linspace(0,365,100)
plt.plot(x_plot,lambda_empirical*np.exp(-lambda_empirical*x_plot))
```

[41]:
```
[<matplotlib.lines.Line2D at 0x7f4925bb7e20>]
```



3 Questioning Assumptions

That doesn't look very good. First, our model underestimates how many short times there would be meaning $\hat{\lambda}$ needs to be smaller. Secondly, our model underestimates how many long times there should be meaning $\hat{\lambda}$ should be larger. This is a problem because any adjustment we do to our model, without changing our model space $\mathcal{M}$, will help with one problem but will make the other problem worse. It is also worth noting that this happens even though we haven't done a train-test split but rather used the entire set for both train and test. So our model should fit the data really well.

Remember that we made some standard assumptions earlier on and those assumptions were used as a basis for our model. A model which turned out to be bad. This makes me think that our assumptions were bad and we should do some open ended exploration.

First we may note that there are suprisingly many variables in the $200 - 300$ days range meaning we are suprisingly often getting $7 - 10$ months between hurricanes. A first thought might be that there are certain times of the year when hurricanes are less likely. Lets check that by plotting how many hurricanes we have at various months.

[42]:
```
monthlist = [0]*12
#monthlist[i] will be the number of hurricanes which took place on month i.
#By our earlier assumptions each entry in monthlist should be about the same␣
 ↪number.
for i in range(0,len(origin_date_time)):
    monthlist[origin_date_time[i].month-1] +=1
```

[43]:
```
monthlist
```

[43]:
```
[4, 1, 1, 5, 29, 119, 153, 379, 639, 369, 97, 18]
```

Okey… That was a pretty big effect. It definately seems like the second assumption was unreasonable because the rate of occurance seems to change over time. The independens assumption is also suspect since, for example, if X_i is 250 days then X_{i+1} will likely be small (it is happening in the beginning of hurricane season).

3.1 Change of model

We should try to model this in a different way. Let X_0 be the number of hurricanes year 1851 and X_i be the number of hurricanes year $i + 1851$. Then we assume: * $\{X_i\}$ is i.i.d * $X_i \sim Poi(\lambda(t))$ where we allow the rate of occurance $\lambda(t)$ to vary over time. * $\lambda(t)$ is periodic with period of 1 year, meaning that it varies by month but not year.

Luckily for us if we integrate the rate of occurance $\lambda(t)$ from time t_1 to time t_2 and get $\mu_{(t_1,t_2)}$, then X_i will be $Poi(\mu_{(t_1,t_2)})$-distributed. Therefore let t_1 and t_2 occur at the start and end of the same year. Then we can get a constant

$$\mu = \int_{t_1}^{t_2} \lambda(t)\, d\lambda$$

and each

$$X_i \sim Poi(\mu)$$

.

With a collection of Poisson distributed i.i.d variables we are back in well traveled water. We may for example follow the example set forth in https://www.statlect.com/fundamentals-of-statistics/Poisson-distribution-maximum-likelihood. They give us the maximum likelihood estimator $\hat{\mu}$ given by

$$\hat{\mu} = \frac{1}{n}\sum_{i=0}^{n} x_i$$

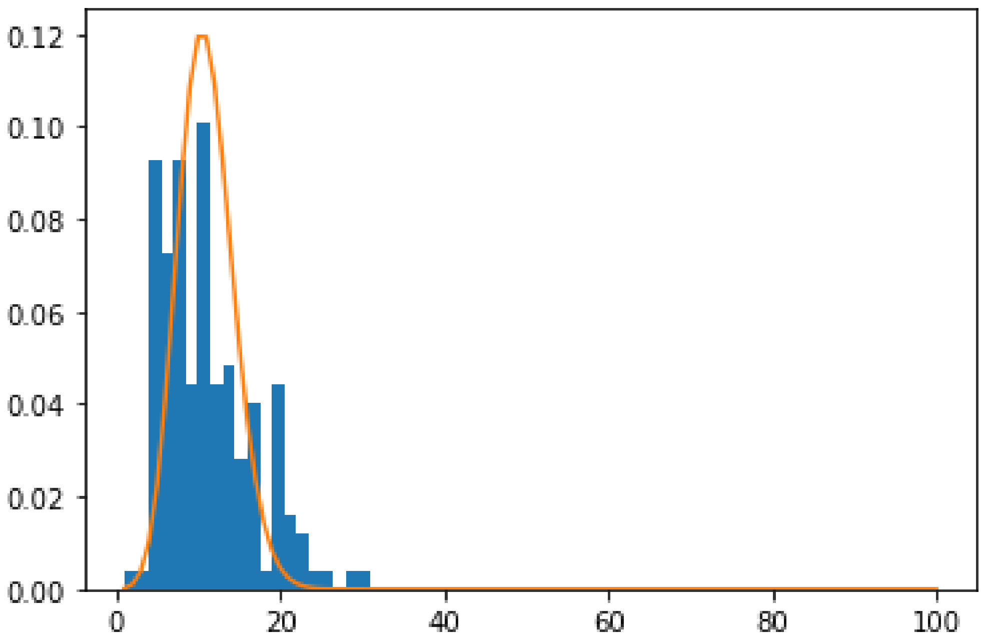
where n is the number of years we meassured (n = the last year - the first year + 1). So let's calculate the maximum likelihood estimator and compare our sample to the probability mass function of a $Poi(\hat{\mu})$ variable.

[44]:
```
n = (origin_date_time[-1].year-origin_date_time[0].year+1)
occurances_per_year=[0] * n
#occurance_per_year[i] will be x[i], i.e. the sampled value of X[i].

for i in range(0, len(origin_date_time)):
    occurances_per_year[origin_date_time[i].year-1851] += 1

#Lets check x_i.
print(occurances_per_year[0:10])
```

```
[6, 5, 8, 5, 5, 6, 4, 6, 8, 7]
```

[45]:
```
import math
mu_estimate = np.mean(occurances_per_year)

_=plt.hist(occurances_per_year,bins=20,density=True)
x_plot = np.linspace(1,100,100)
x_plot_factorial = [1]
for i in range(1,100):
    x_plot_factorial.append(math.factorial(i+1))
plt.plot(x_plot,np.exp(-mu_estimate)/x_plot_factorial*mu_estimate**x_plot)
```

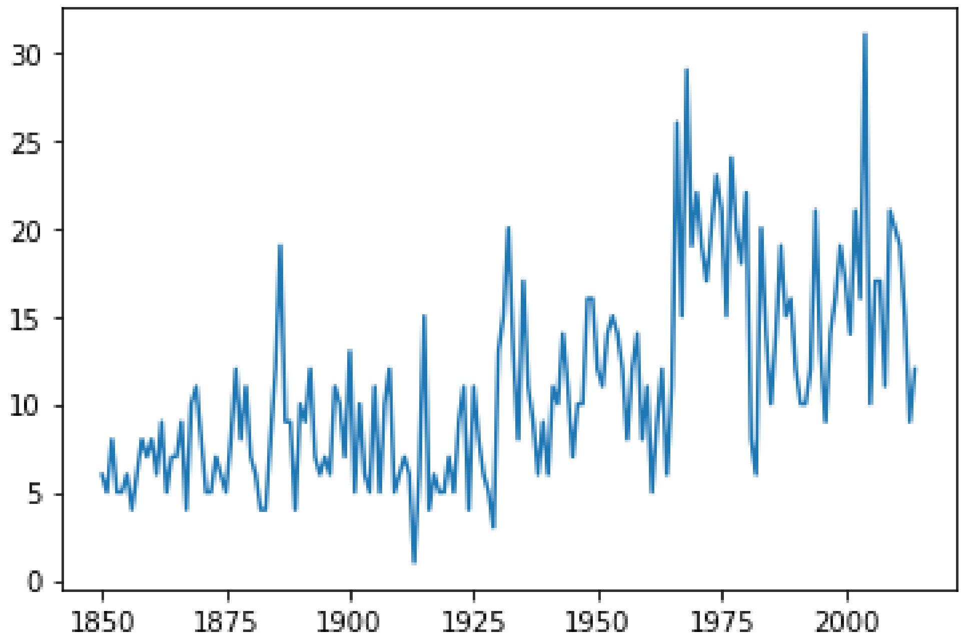
[45]:
```
[<matplotlib.lines.Line2D at 0x7f48bd021c90>]
```



Damn it. This has a similar problem, too many small values and too many large values. This means a big variance but since variance = mean for a Poi-distributed variable we cannot improve this. What else could be going on? Maybe our assumption that the μ is stable over the years might be wrong, maybe we are seing an increase in hurricanes due to changing climate. Let's investigate.

[46]:
```
x_points = []
for i in range(0, len(occurances_per_year)):
    x_points.append(i+1850)

plt.plot(x_points, occurances_per_year)
```

[46]:
```
[<matplotlib.lines.Line2D at 0x7f48bceaef20>]
```



Yeah, the assumption that $\lambda(t)$ is stable over the years may have been rather misguided.

4 Investigate change over time

Okey. This project seems to be evolving into something new. We should probably investigate changes of rate of occurance over the years. Let's try to model this change as well as predict what will happen until maybe 2050. It would probably also be useful to prove statistically that there has been a change. We start with the modelling.

Let's start by getting rigoruous about our time variable because there are quite a few things going on with it. We want to investigate changes over the years but hide changes within years so our definitions should be made such that they help us do that. Let t_0 be the start of 1851 and let t_i be the start of year $1851 + i$. Let T_i be the intervall $[t_i, t_{i+1})$. Let X_i be the number of hurricanes during time T_i and let $\lambda(t)$ be the rate of occurance at time t. Let

$$\lambda_i := \int_{T_i} \lambda(t)\, dt$$

This makes $X_i \sim Poi(\lambda_i)$ and we are very interested in how λ_i changes with i. As before we let x_i be the observed value of X_i and also let $\widehat{\lambda}_i$ be the estimated value of λ_i.

We are still not in unknown waters. There is research and best practices on this type of situation and the standard thing to do is to model $\lambda_i = e^{(\alpha+\beta i)}$ for some $\alpha, \beta \in \mathbb{R}$. It is called poisson regression(or a generalized linear model) and is described at https://bookdown.org/roback/bookdown-BeyondMLR/ch-poissonreg.html for example. A more common way to state my assumption is that the logarithm of the expectation of X_i is linear with regard to the variables (in our case just the time variable i). I just find this description to be confusing and less useful.

So we assume $\lambda_i = e^{\alpha+\beta i}$ and now we wish to find the maximum likelihood estimator. Let's use the log loss again but this time we need to minimize the risk with respect to α and β. There is no closed form expression for this minimum so we have to find it algorithmicly. Luckily there are ways to do that using methods in statsmodels.api. We also import pandas, patsy and numpy to more efficiently handle the data. At this point we should probably also do a train-test divide. Due to our independence assumption this can be done in a completely random way.

[47]:
```
import pandas as pd
from patsy import dmatrices
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt

#t_i = times[i]
times = []
for i in range(0,len(occurances_per_year)):
    times.append(i+1850)

#we define df as a pandas dataframe with our t_i and x_i. The easiest way to␣
↪create this is to start by
#putting the data into a dictionary, we call it dataDict and then forget about␣
↪it.
dataDict = {
  "year": times,
  "x": occurances_per_year
}
df = pd.DataFrame(dataDict)

#Now we put approximately 80% of our data into a training set and reserve the␣
↪rest for the test set.
mask = np.random.rand(len(df)) < 0.8
df_train = df[mask]
df_test = df[~mask]


print('Training data set length='+str(len(df_train)))
print('Testing data set length='+str(len(df_test)))
print('')
print('The data frame df is equal to: ')
print(df)
```

```
Training data set length=135
Testing data set length=30

The data frame df is equal to:
     year   x
0    1850   6
1    1851   5
```

```
2    1852   8
3    1853   5
4    1854   5
..    …    ..
160  2010  20
161  2011  19
162  2012  15
163  2013   9
164  2014  12

[165 rows x 2 columns]
```

[48]:

```python
#Set up the expression in patsy notation.
#This expression means that 'x' is the dependent variable and 'year' is the predictor
expr = """x ~ year"""

#create dmatrices as these will help us use the statsmodel training methods for GLM(generalized linear model)
x_train, year_train = dmatrices(expr, df_train, return_type='dataframe')
x_test, year_test = dmatrices(expr, df_test, return_type='dataframe')

#Train the model and print a summary of the training results from statsmodels
model_trained = sm.GLM(x_train, year_train, family=sm.families.Poisson()).fit()
print(model_trained.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      x   No. Observations:                  135
Model:                            GLM   Df Residuals:                      133
Model Family:                 Poisson   Df Model:                            1
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -383.63
Date:                Sat, 04 Nov 2023   Deviance:                       209.36
Time:                        23:46:26   Pearson chi2:                     216.
No. Iterations:                     4   Pseudo R-squ. (CS):             0.6626
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -10.6441      1.099     -9.681      0.000     -12.799      -8.489
year           0.0067      0.001     11.911      0.000       0.006       0.008
==============================================================================
```

First we note that the 95% CI for the ‘year’ variable is (0.006,0.008) meaning that we have a statistical assurance that the year has an effect on the number of hurricanes.

5 Test

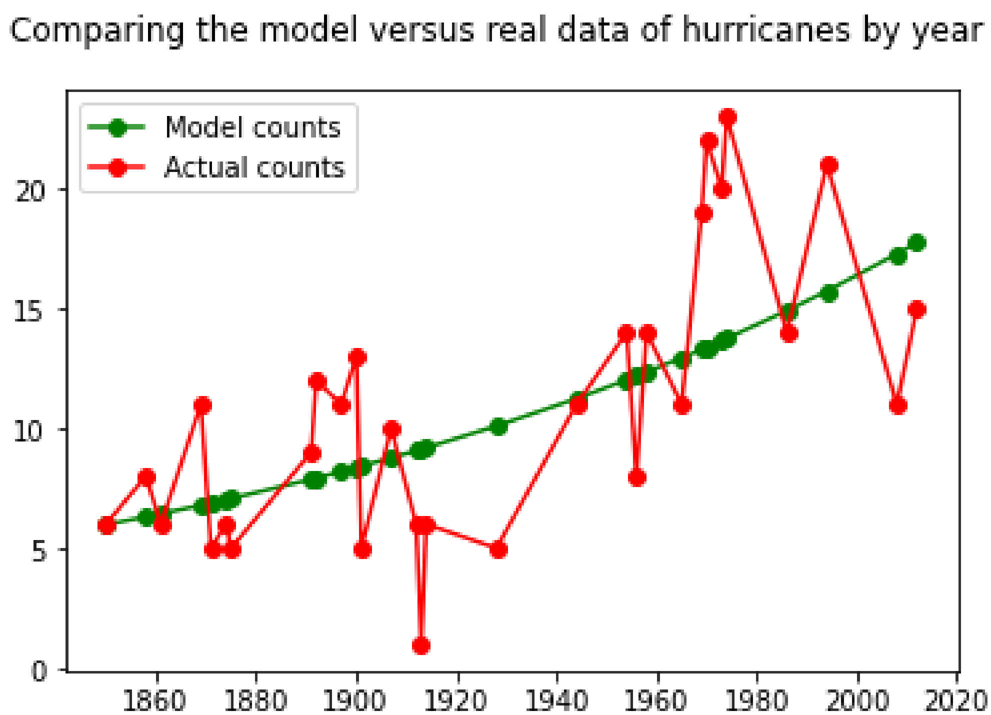
Now we have our trained model so lets try to compare it to our test set and print a visual to see how well it actually fits.

[49]:
```
#Get predictions to compare to the test set.
model_test = model_trained.get_prediction(year_test)
#summary_frame() returns a pandas DataFrame with standard error and a 95%␣
↪confidence interval
model_test_summary_frame = model_test.summary_frame()
print(model_test_summary_frame)
```

```
          mean   mean_se  mean_ci_lower  mean_ci_upper
0     5.976606  0.365703       5.301155       6.738121
8     6.306700  0.360322       5.638588       7.053977
11    6.435133  0.357989       5.770385       7.176460
19    6.790552  0.350939       6.136412       7.514423
21    6.882433  0.348993       6.231311       7.601591
24    7.022590  0.345940       6.376263       7.734431
25    7.069940  0.344888       6.425282       7.779279
41    7.872467  0.326050       7.258670       8.538167
42    7.925548  0.324775       7.313892       8.588356
47    8.196369  0.318311       7.595645       8.844604
50    8.363284  0.314401       7.769224       9.002768
51    8.419674  0.313100       7.827840       9.056255
57    8.766091  0.305422       8.187456       9.385620
62    9.065634  0.299393       8.497423       9.671841
63    9.126760  0.298252       8.560524       9.730450
64    9.188298  0.297139       8.623989       9.789533
78   10.094697  0.286083       9.549273      10.671275
94   11.240572  0.292823      10.681056      11.829398
104  12.021893  0.314313      11.421368      12.653992
106  12.184557  0.320570      11.572176      12.829344
108  12.349422  0.327519      11.723895      13.008324
115  12.944216  0.357440      12.262268      13.664089
119  13.296873  0.378514      12.575315      14.059833
120  13.386528  0.384238      12.654228      14.161207
123  13.659138  0.402501      12.892600      14.471251
124  13.751236  0.408952      12.972618      14.576587
136  14.906067  0.500448      13.956783      15.919918
144  15.729345  0.575775      14.640377      16.899311
158  17.280999  0.735269      15.898352      18.783893
162  17.751809  0.787434      16.273654      19.364228
```

[50]:
```
#Now let's plot the predicted and actual hurricanes per year
model_counts=model_test_summary_frame['mean']
actual_counts = x_test['x']
```

```
#Create the figure
fig = plt.figure()
model, = plt.plot(year_test['year'].to_numpy(), model_counts.to_numpy(), 'go-',␣
 ↪label='Model counts')
actual, = plt.plot(year_test['year'].to_numpy(), actual_counts.to_numpy(),␣
 ↪'ro-', label='Actual counts')

#Make it prettier & show it off
plt.legend(handles=[model, actual])
fig.suptitle('Comparing the model versus real data of hurricanes by year')
plt.show()
```



[51]:
```
#Print test to csv
actual_counts.to_csv('actual_counts.csv')
model_counts.to_csv('model_counts.csv')

#We also bring the whole data into a csv
df.to_csv('data.csv')

#We create a file of predictions for the time period of the data as well as␣
 ↪some years into the future
prediction_years = []
for i in range(1851, 2050):
```

```
    prediction_years.append(i)
data_prediction = {
  "year": prediction_years,
  "x": 0
}
df_prediction = pd.DataFrame(data_prediction)
x_prediction, year_prediction = dmatrices(expr, df_prediction,␣
 ↪return_type='dataframe')
prediction = model_trained.get_prediction(year_prediction)
prediction_summary_frame = prediction.summary_frame()
prediction_counts=prediction_summary_frame['mean']
prediction_counts.to_csv('prediction_counts.csv')

#We also wish to save the number of hurricanes per month into a csv.
months = {
    'Month' : ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August', 'September', 'Oktober', 'November', 'December'],
    'Hurricanes' : monthlist
}
monthsdf = pd.DataFrame(months)
monthsdf.to_csv('months.csv')
```