# Hurricane

November 3, 2023

## 1 Data import and continued cleaning

[81]:
```
%%bash
head -n 2 Hurricanes.csv
```

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

[82]:
```
import csv

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)
```

[83]:
```
print(header)
```

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

[84]:
```
from datetime import datetime, timedelta
format_string = "%Y-%m-%d %H:%M:%S"
datetime.strptime(data[0][1],format_string)
```

[84]:
```
datetime.datetime(1851, 6, 25, 0, 0)
```

[85]:
```
origin_date = [datetime.strptime(line[1],format_string) for line in data]
```

[86]:
```
for i in range(0,10):
    print(header[2] + ': ' + data[i][2])
```

```
Time: 0
Time: 1200
Time: 1200
Time: 0
Time: 0
Time: 1200
Time: 1200
```

```
Time: 600
Time: 1200
Time: 0
```

[87]:

```
origin_date_time = []
for i in range(0,len(origin_date)):
    h = 0
    if(len(data[i][2]) == 3):
        h = int(data[i][2][0])
    if(len(data[i][2]) == 4):
        h = int(data[i][2][0:2])
    time_change = timedelta(hours=h)
    origin_date_time.append(origin_date[i] + time_change)

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 $i+1$. We assume: * $\{X_i\}$ is i.i.d * The rate of occurance ($\lambda$) doesn't change over time so 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 is

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

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

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

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 $\lambda$ result in a positive derivative (of the risk function) and larger $\lambda$ results in a negative derivative meaning we have found a minimum.

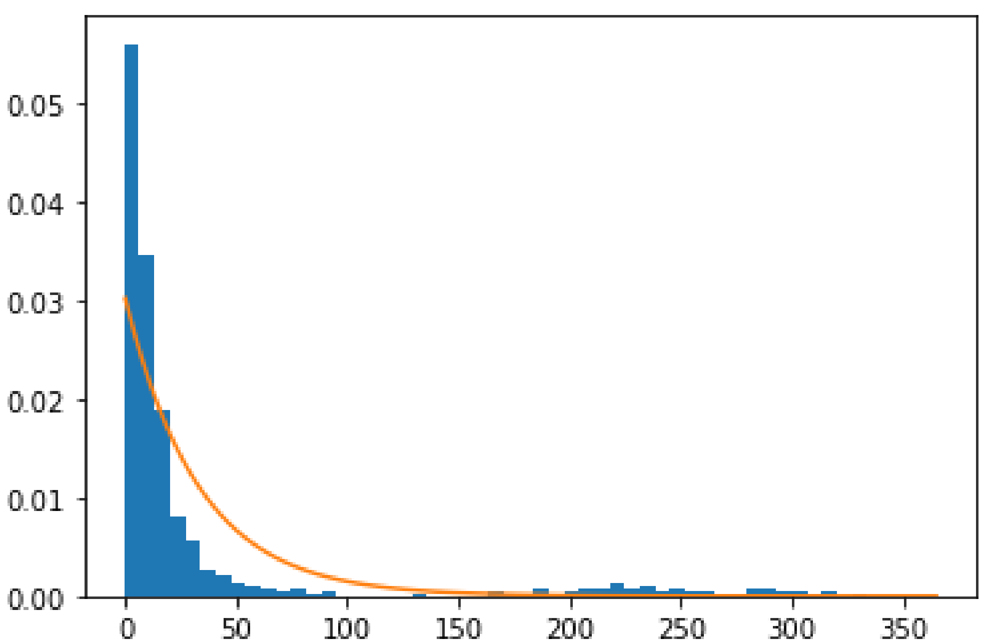
[88]:
```
time_differences = []
#X_i = time_differences[i]
for i in range(0,(len(origin_date_time)-1)):
        time_differences.append((origin_date_time[i+1]-origin_date_time[i]).
 ↪total_seconds()/3600/24)
#time_differences are now represented as days since seconds/3600 = hours and␣
 ↪hours/24 = days
```

[89]:
```
import numpy as np
lambda_empirical = 1/np.mean(time_differences)
```

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

[90]:
```
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))
```

[90]:
```
[<matplotlib.lines.Line2D at 0x7f847de08e20>]
```



## 3 Questioning Assumptions

That doesn't look very good. First, our model underestimates how many short times there would be meaning $\lambda$ needs to be smaller. Secondly, our model underestimates how many long times there should be meaning $\lambda$ 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 almost a year 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.

[91]:
```
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
```

[92]:
```
monthlist
```

[92]:
```
[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.

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. Further, let's assume that the rate of occurance only varies within years but not between years and let $t_1$ and $t_2$ occur at the start and end of any 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

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

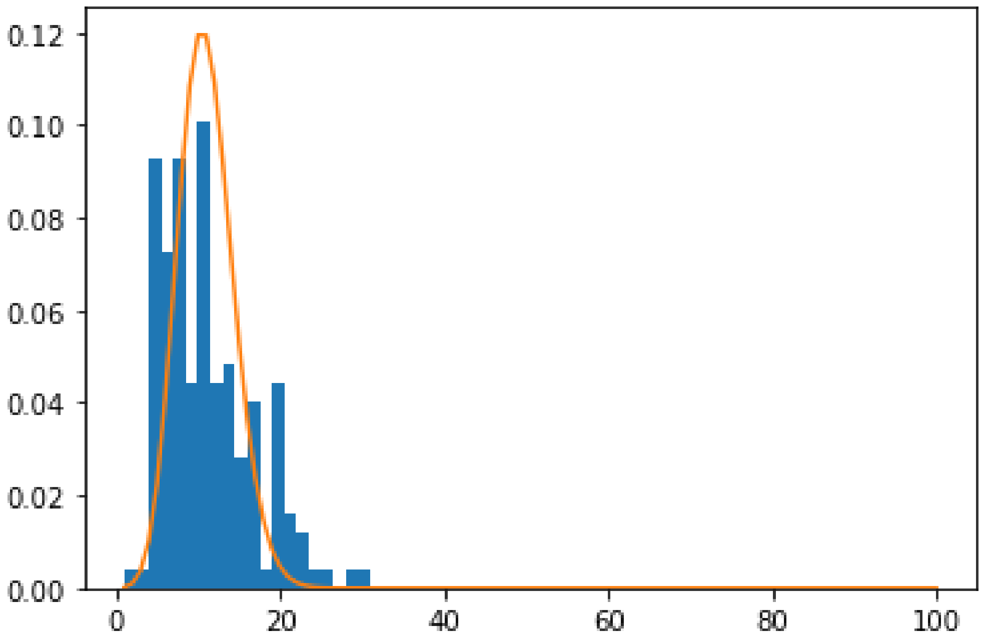
where $n$ is the number of years we meassured (n = the last year - the first year + 1). So let's calculate the maximum likelihood estimator and compare our sample to the probability mass function of a $Poi(\hat{\mu})$ variable.

[93]:
```
n = (origin_date_time[-1].year-origin_date_time[0].year+1)
#n = last year - first year + 1
occurances_per_year=[0] * n
#occurance_per_year[i] will be x[i], that is 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]
```

[94]:
```
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)
```

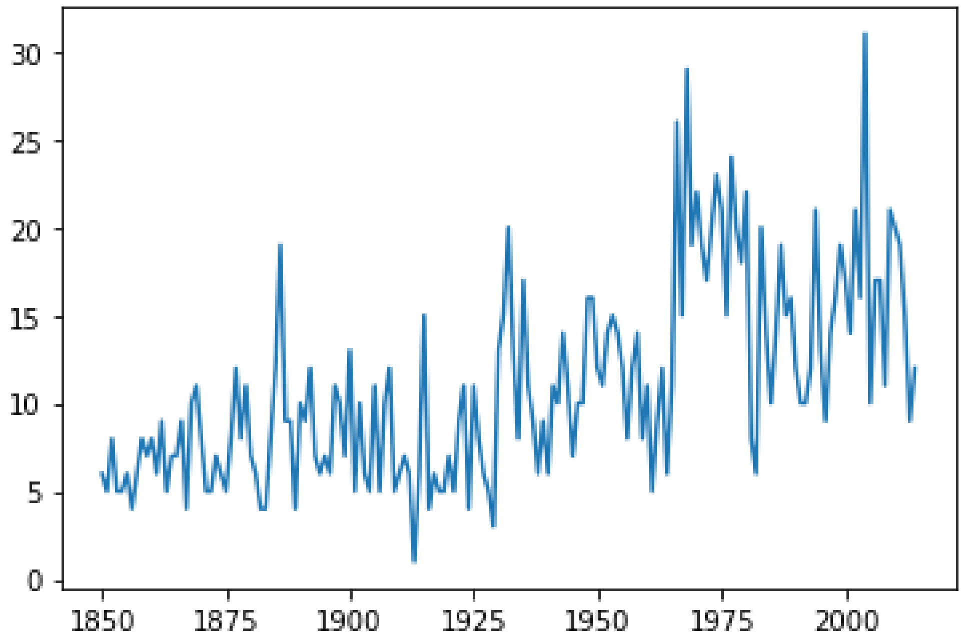
[94]:
```
[<matplotlib.lines.Line2D at 0x7f847e2deb30>]
```




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 $\mu$ is stable over the years might be wrong, maybe we are seing an increase in hurricanes due to changing climate. Let's investigate.

[95]:
```python
x_points = []
for i in range(0, len(occurances_per_year)):
    x_points.append(i+1850)

plt.plot(x_points, occurances_per_year)
```

[95]:
```
[<matplotlib.lines.Line2D at 0x7f84805ad540>]
```




Yeah, the assumption that the rate of hurricanes 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 and also build a simple simulation tool. 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

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

Now $X_i \sim Poi(\lambda_i)$ and we wish to model how $\lambda_i$ changes with $i$. As before we let $x_i$ be the observed value of $X_i$ but now we also let $\hat{\lambda}_i$ be the estimated value of $\lambda_i$.

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 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 $\alpha$ and $\beta$. 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.

[96]:
```
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)

#df is 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 data and then forget about it.
data = {
  "year": times,
  "x": occurances_per_year
}
df = pd.DataFrame(data)

#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=136
Testing data set length=29

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]
```

[97]:

```
#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:                  136
Model:                            GLM   Df Residuals:                      134
Model Family:                 Poisson   Df Model:                            1
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -380.41
Date:                Fri, 03 Nov 2023   Deviance:                       200.20
Time:                        14:58:01   Pearson chi2:                     200.
No. Iterations:                     4   Pseudo R-squ. (CS):             0.7052
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -10.9560      1.059    -10.344      0.000     -13.032      -8.880
year           0.0069      0.001     12.663      0.000       0.006       0.008
==============================================================================
```

## 5 Test

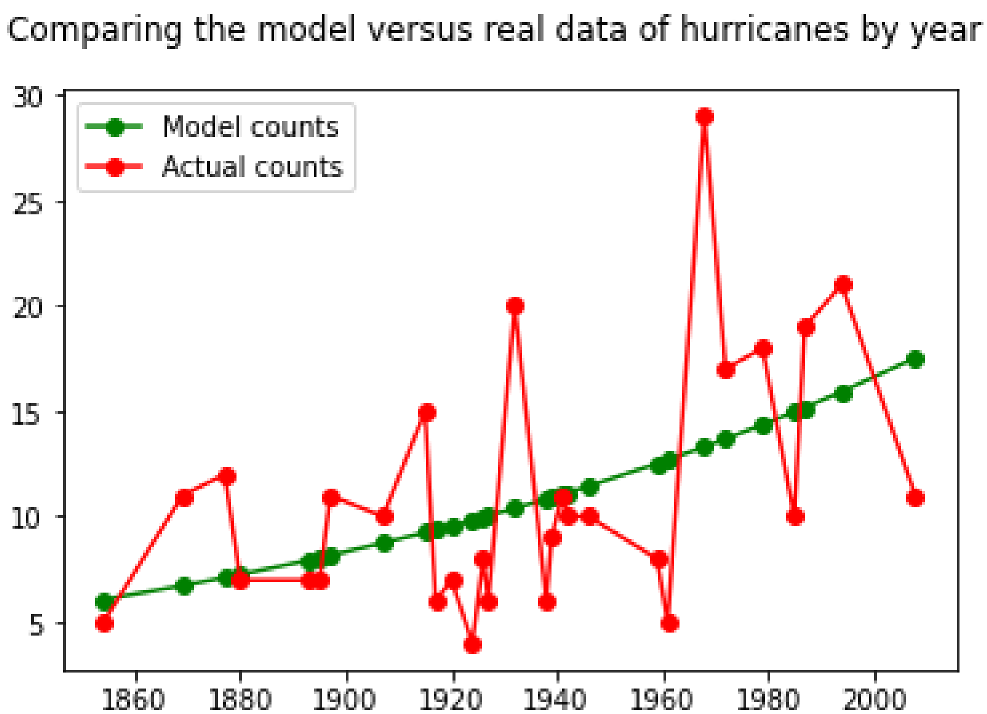
Now we have our trained model so lets try to compare it to our test set and print a visual to see how well it actually fits.

[98]:
```
#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
4     6.074135  0.348345       5.428365       6.796729
19    6.734722  0.338136       6.103552       7.431161
27    7.115931  0.331143       6.495621       7.795479
30    7.264384  0.328272       6.648652       7.937138
43    7.944305  0.314621       7.350984       8.585515
45    8.054414  0.312405       7.464806       8.690592
47    8.166049  0.310178       7.580189       8.797189
57    8.747865  0.299224       8.180623       9.354439
65    9.243026  0.291461       8.689068       9.832301
67    9.371135  0.289797       8.820013       9.956694
70    9.566635  0.287594       9.019245      10.147247
74    9.833662  0.285328       9.290034      10.409102
76    9.969957  0.284540       9.427579      10.543539
77   10.038812  0.284245       9.496878      10.611670
82   10.390283  0.283898       9.848489      10.961882
88   10.828329  0.286468      10.281170      11.404607
89   10.903111  0.287267      10.354369      11.480935
91   11.054229  0.289216      10.501664      11.635869
92   11.130572  0.290375      10.575753      11.714497
96   11.441251  0.296311      10.874985      12.037003
109  12.512113  0.331984      11.878066      13.180006
111  12.685532  0.339911      12.036511      13.369549
118  13.311646  0.373087      12.600130      14.063341
122  13.683204  0.395895      12.928856      14.481566
129  14.358560  0.442594      13.516776      15.252769
135  14.963906  0.489498      14.034615      15.954728
137  15.171307  0.506543      14.210288      16.197317
144  15.920110  0.571760      14.838012      17.081122
158  17.530416  0.728477      16.159224      19.017960
```

[104]:
```
#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()
```




[113]:
```
#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 = predict.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')
```