modelENSO_vs_historical_Comparison–ShowResults

November 6, 2018

Weather Derivatites
Historical Precipitation Comparison
Developed by Jesus Solano
05 November 2018

```
In [2]: # Import needed libraries.
        import numpy as np
        import pandas as pd
        import random as rand
        import matplotlib.pyplot as plt
        import time
        from io import StringIO
        import datetime
        import pickle
        from scipy import stats
```

1 Import Datasets

```
In [3]: # Import total dataset.
        # Configure path to read txts.

        path = '../datasets/'

        # Download the update dataset.

        import os

        if not os.path.exists(path+'/fullDataset/completeDailyDataset.pickle'):

          ! wget https://github.com/jesugome/WeatherDerivates/raw/master/datasets/fullDataset/co

        allDataDataframe = pickle.load(open(path+'/fullDataset/completeDailyDataset.pickle','rb'
```

```
In [4]: allDataDataframe.head(30)
```

```
Out[4]:             Prep Month    nino34 probNeutral probNino probNina state nextState
        2005-01-01     0     1  0.606186        0.15     0.85        0     0         0
```

```
2005-01-02    0     1  0.599358        0.15     0.85        0     0         0
2005-01-03    0     1  0.646784        0.15     0.85        0     0         1
2005-01-04  1.6     1  0.663696        0.15     0.85        0     1         0
2005-01-05    0     1   0.71133        0.15     0.85        0     0         0
2005-01-06    0     1  0.679185        0.15     0.85        0     0         0
2005-01-07    0     1  0.558135        0.15     0.85        0     0         0
2005-01-08    0     1  0.451021        0.15     0.85        0     0         0
2005-01-09    0     1  0.593434        0.15     0.85        0     0         1
2005-01-10  0.4     1  0.672419        0.15     0.85        0     1         0
2005-01-11    0     1  0.757247        0.15     0.85        0     0         0
2005-01-12    0     1  0.755326        0.15     0.85        0     0         0
2005-01-13    0     1  0.670963        0.15     0.85        0     0         1
2005-01-14  0.4     1  0.486574        0.15     0.85        0     1         1
2005-01-15  0.2     1  0.443672        0.15     0.85        0     1         1
2005-01-16  1.1     1  0.494552        0.15     0.85        0     1         1
2005-01-17  0.3     1  0.543425        0.15     0.85        0     1         0
2005-01-18    0     1  0.536013        0.15     0.85        0     0         0
2005-01-19    0     1   0.58257        0.15     0.85        0     0         1
2005-01-20    2     1  0.618696        0.15     0.85        0     1         0
2005-01-21    0     1  0.638987        0.15     0.85        0     0         0
2005-01-22    0     1  0.715913        0.15     0.85        0     0         0
2005-01-23    0     1   0.65339        0.15     0.85        0     0         0
2005-01-24    0     1  0.608979        0.15     0.85        0     0         0
2005-01-25    0     1  0.576707        0.15     0.85        0     0         0
2005-01-26    0     1  0.636912        0.15     0.85        0     0         0
2005-01-27    0     1  0.572322        0.15     0.85        0     0         1
2005-01-28    1     1  0.432275        0.15     0.85        0     1         1
2005-01-29  2.4     1  0.240318        0.15     0.85        0     1         0
2005-01-30    0     1  0.138489        0.15     0.85        0     0         1
```

```
In [5]: # Download Data

        # Configure path to read txts.

        path = '../datasets/'

        # Download the update dataset.

        import os

        if not os.path.exists(path+'precipitationAllTime.csv'):

          ! wget https://github.com/jesugome/WeatherDerivates/raw/master/datasets/precipitationA

          ! wget https://github.com/jesugome/WeatherDerivates/raw/master/datasets/precipitationR


        precipitationAllTime = pd.read_csv(path+'precipitationAllTime.csv',header=None, names =
        precipitationAllTime['Date'] = pd.to_datetime(precipitationAllTime['Date'])
        precipitationAllTime = precipitationAllTime.set_index('Date')

        precipitationAllTime.head(10)
```

```
Out[5]:             Prep
        Date
        1972-01-01   0.0
        1972-01-02   0.7
        1972-01-03   0.0
        1972-01-04   0.0
        1972-01-05   0.0
        1972-01-06   0.0
        1972-01-07   0.0
        1972-01-08   0.2
        1972-01-09   2.2
        1972-01-10   0.0
```

2 Historical Histograms

```
In [6]: # Creates a function to plot a month historical accumulated rainfall over years.

        def monthAccumulatedHistogram(month,allDataDataframe,bins,color):

            # Defines dates for specify month.

            monthDataRange = pd.date_range(start = '1972-'+str(format(month,'02'))+'-01', end =

            # Historical month Rainfall per year.
            monthTotalRainfall=[]

            for date in monthDataRange:
                tempDate=pd.date_range(date,end=date+1,freq='MS')
                tempDateRange= pd.date_range(start=date,end=tempDate[0]+1,freq='D')

                # Define accumulated rainfall.
                rainfallSum = 0
                for day in tempDateRange[:-1]:

                    #print(allDataDataframe.loc[date.strftime('%Y-%m-%d'),'Prep'])
                    rainfallSum+= allDataDataframe.loc[day.strftime('%Y-%m-%d'),'Prep']

                monthTotalRainfall.append(rainfallSum)

            fig = plt.figure(figsize=(20, 10))
```

```
    plt.hist(monthTotalRainfall,facecolor=color,bins=bins, density=True,
             histtype='stepfilled', edgecolor = 'black' , hatch = '+')

    plt.title('Rainfall Simulation -->  Month '+str(month))
    plt.xlabel('Rainfall Accumulated Amount [mm]')
    plt.ylabel('Probability ')
    plt.grid()
    plt.show()

    ## Print Statistics.

    print(stats.describe(monthTotalRainfall),'\n\n')
```

3 Simulated Data

3.0.1 Download Data

```
In [7]: ### Load transitions and amount parameters.

        # Transitions probabilites.
        transitionsParametersDry = pd.read_csv('../results/visibleMarkov/transitionsParametersDr
        transitionsParametersDry.index += 1
        transitionsParametersDry

        transitionsParametersWet = pd.read_csv('../results/visibleMarkov/transitionsParametersWe
        transitionsParametersWet.index += 1
        transitionsParametersWet

        amountParametersGamma = pd.read_csv('../results/visibleMarkov/amountGammaPro.csv', sep =
        amountParametersGamma.index += 1

        ### ENSO probabilistic forecast.

        # Open saved data.
        ensoForecast = pickle.load(open('../datasets/ensoForecastProb/ensoForecastProbabilities.
```

3.0.2 Simulation Core

```
In [8]: # Import needed libraries.
        import numpy as np
        import pandas as pd
        import random as rand
        import matplotlib.pyplot as plt
        from scipy.stats import bernoulli
        from scipy.stats import gamma
        import pickle
        import time
import datetime
from scipy import stats
```

In [9]:

```
### Build the simulation core.

# Updates the state of the day based on yesterday state.
def updateState(yesterdayIndex, simulationDataFrame, transitionsParametersDry, transitio

    # Additional data of day.
    yesterdayState = simulationDataFrame['state'][yesterdayIndex]
    yesterdayPrep = simulationDataFrame['Prep'][yesterdayIndex]
    yesterdayProbNino = simulationDataFrame['probNino'][yesterdayIndex]
    yesterdayProbNina = simulationDataFrame['probNina'][yesterdayIndex]
    yesterdayMonth = simulationDataFrame['Month'][yesterdayIndex]

    # Calculate transition probability.
    if yesterdayState == 0:
        # Includes month factor + probNino value + probNino value.
        successProbabilityLogit = transitionsParametersDry['value'][1]+transitionsParame

        if yesterdayMonth==1:
            # Includes month factor + probNino value + probNino value.
            successProbabilityLogit = transitionsParametersDry['value'][yesterdayMonth]

        successProbability = (np.exp(successProbabilityLogit))/(1+np.exp(successProbabil

    elif yesterdayState == 1:
        # Includes month factor + probNino value + probNino value + prep value .
        successProbabilityLogit = transitionsParametersDry['value'][1]+ transitionsParam

        if yesterdayMonth==1:
            # Includes month factor + probNino value + probNino value + prep value .
            successProbabilityLogit = transitionsParametersDry['value'][yesterdayMonth]

        successProbability = (np.exp(successProbabilityLogit))/(1+np.exp(successProbabil
    else:
        print('State of date: ', simulationDataFrame.index[yesterdayIndex],' not found.'

    #print(successProbability)
    #successProbability = monthTransitions['p'+str(yesterdayState)+'1'][yesterdayMonth]

    todayState = bernoulli.rvs(successProbability)

    return todayState



# Simulates one run of simulation.
```

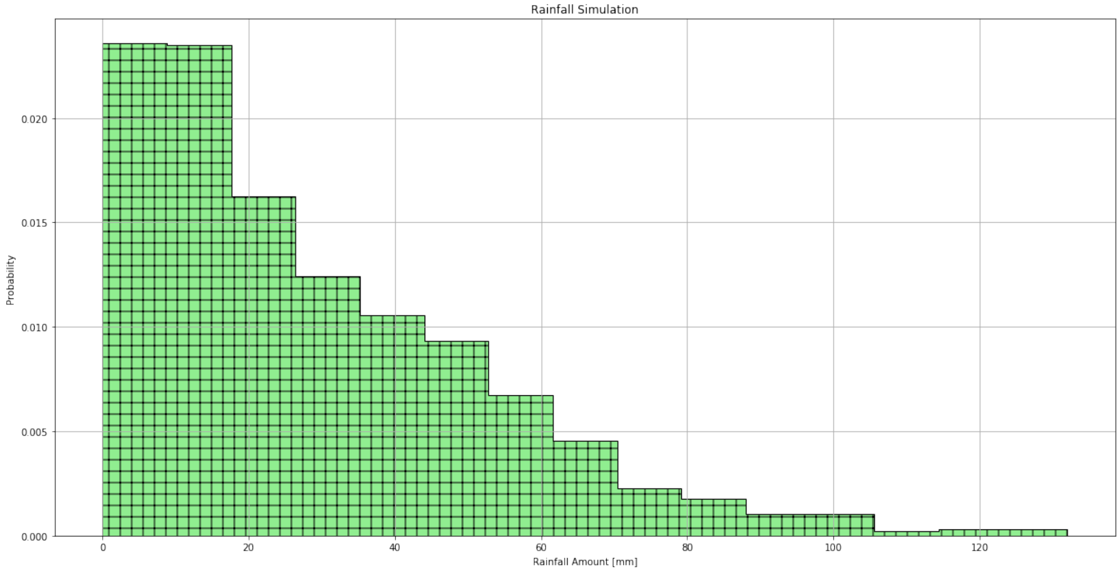
```
def oneRun(simulationDataFrame, transitionsParametersDry, transitionsParametersWet, amou

    # Define the total rainfall amount over the simulation.
    rainfall = 0

    # Total rainfall days.
    wetDays = 0

    # Loop over days in simulation to calculate rainfall ammount.
    for day in range(1,len(simulationDataFrame)):

        # Get today date.
        dateOfDay = datetime.datetime.strptime(simulationDataFrame.index[day],'%Y-%m-%d'


        # Update today state based on the yesterday state.
        todayState = updateState(day-1, simulationDataFrame, transitionsParametersDry, t

        # Write new day information.
        simulationDataFrame['state'][day] = todayState
        simulationDataFrame['nextState'][day-1] = todayState

        # Computes total accumulated rainfall.
        if todayState == 1:

            # Sum wet day.
            wetDays+=1

            # Additional data of day.
            todayProbNino = simulationDataFrame['probNino'][day]
            todayProbNina = simulationDataFrame['probNina'][day]
            todayMonth = simulationDataFrame['Month'][day]



            # Calculates gamma log(mu).
            gammaLogMu = amountParametersGamma['mu'][1] + amountParametersGamma['mu'][to
            #print(gammaMu)
            # Calculates gamma scale
            gammaLogShape = amountParametersGamma['shape'][1] + amountParametersGamma['s
            #print(gammaShape)

            if todayMonth==1:
                # Calculates gamma log(mu).
                gammaLogMu =  amountParametersGamma['mu'][todayMonth]+ todayProbNino*amo
                #print(gammaMu)
                # Calculates gamma scale
                gammaLogShape = amountParametersGamma['shape'][todayMonth]+ todayProbNin
                #print(gammaShape)

            # Update mu
            gammaMu = np.exp(gammaLogMu)

            # Update shape
            gammaShape = np.exp(gammaLogShape)

            # Calculate gamma scale.
            gammaScale = gammaMu / gammaShape

            # Generate random rainfall.
            todayRainfall = gamma.rvs(a = gammaShape, scale = gammaScale)

            '''

            # !!!!!! Delete !!!!!!!!!!!!!11.
            todayRainfall = gamma.rvs(amountParametersGamma['Shape'][0],amountParameters

            '''


            # Write new day information.
            simulationDataFrame['Prep'][day] = todayRainfall

            # Updates rainfall amount.
            rainfall += todayRainfall

        else:
            # Write new day information.
            simulationDataFrame['Prep'][day] = 0


        yesterdayState = todayState


    return rainfall,wetDays

# Run total iterations.
def totalRun(simulationDataFrame, transitionsParametersDry, transitionsParametersWet, am

    # Initialize time
    startTime = time.time()

    # Array to store all precipitations.
    rainfallPerIteration = [None]*iterations

    wetDaysPerIteration = [None]*iterations
    # Loop over each iteration(simulation)

    for i in range(iterations):

        simulationDataFrameC = simulationDataFrame.copy()

        iterationRainfall,wetDays = oneRun(simulationDataFrameC, transitionsParametersDr

        rainfallPerIteration[i] = iterationRainfall

        wetDaysPerIteration[i] = wetDays

    # Calculate time
    currentTime = time.time() - startTime

    # Print mean of wet days.

    #print('The mean of wet days is: ', np.mean(wetDaysPerIteration))

    # Logging time.
    #print('The elapsed time over simulation is: ', currentTime, ' seconds.')

    return rainfallPerIteration



def createTotalDataFrame(daysNumber, startDate , initialState , initialPrep , ensoForeca
    # Set variables names.
    totalDataframeColumns = ['state','Prep','Month','probNina','probNino', 'nextState']

    # Create dataframe.

    allDataDataframe = pd.DataFrame(columns=totalDataframeColumns)

    # Number of simulation days(i.e 30, 60)
    daysNumber = daysNumber

    # Simulation start date ('1995-04-22')
    startDate = startDate

    # State of rainfall last day before start date --> Remember 0 means dry and 1 means
    initialState = initialState
    initialPrep = initialPrep   # Only fill when initialState == 1


    dates = pd.date_range(startDate, periods = daysNumber + 2 , freq='D')
    for date in dates:

        # Fill precipitation amount.
        allDataDataframe.loc[date.strftime('%Y-%m-%d'),'Prep'] = np.nan

        # Fill month of date
        allDataDataframe.loc[date.strftime('%Y-%m-%d'),'Month'] = date.month

        tempDate = None
        if optionMonthTerm==1:
            tempDate = date
        else:
            tempDate = date - pd.DateOffset(months=optionMonthTerm-1)

        # Fill El Nino ENSO forecast probability.
        allDataDataframe.loc[date.strftime('%Y-%m-%d'),'probNino'] = float(ensoForecast[

        # Fill La Nina ENSO forecast probability.
        allDataDataframe.loc[date.strftime('%Y-%m-%d'),'probNina'] = float(ensoForecast[

        # Fill State.
        allDataDataframe.loc[date.strftime('%Y-%m-%d'),'state'] = np.nan


    simulationDataFrame = allDataDataframe[:-1]

    # Fill initial conditions.
    simulationDataFrame['state'][0] = initialState
    if initialState == 1:
        simulationDataFrame['Prep'][0] = initialPrep
    else:
        simulationDataFrame['Prep'][0] = 0.0

    return simulationDataFrame


def plotRainfallDistribution(rainfallSimulated):

    # Create Figure.
    fig = plt.figure(figsize=(20, 10))

    # Plot histogram.
    plt.hist(rainfallSimulated,facecolor='lightgreen',bins=15, density=True,
           histtype='stepfilled', edgecolor = 'black' , hatch = '+')

    # Add axis names.
    plt.title('Rainfall Simulation')
    plt.xlabel('Rainfall Amount [mm]')
    plt.ylabel('Probability ')
    plt.grid()
    plt.show()


def optionRainfallCalculator(iterations, startDate, transitionsParametersDry, transition

    ## Generates initial conditions.

    # Defines initial state based on proportions.
    successProbability = 0.5
    initialState = bernoulli.rvs(successProbability)

    # Calculates initial prepicipitation.
    if initialState == 1:
        initialPrep = 1.0
    else:
        initialPrep = 0.0

    ## Create dataframe to simulate.
    simulationDataFrame = createTotalDataFrame(daysNumber= 30, startDate = startDate, in

    ## Run all iterations.
    rainfallPerIteration = totalRun(simulationDataFrame, transitionsParametersDry, trans



    return rainfallPerIteration
```

4 Final Results

```
In [12]: for month in range(1,13):
             print('\n \n Current Month is:',format(month,'02'))


             print('\n Simulated: \n ')
             monthRainfall=[]

             startYear = 2005
             endYear = 2016

             for year in range(startYear,endYear):
                 #print(year)
                 monthRainfallYear = optionRainfallCalculator(iterations=100,
                                               startDate=str(year)+'-'+str(format(month,'02'))+'-01',
                                               transitionsParametersDry= transitionsParametersDry ,
                                               transitionsParametersWet = transitionsParametersWet,
                          amountParametersGamma = amountParametersGamma,
                          optionMonthTerm = 1)
    monthRainfall.extend(monthRainfallYear)

## Plot histogram.

plotRainfallDistribution(monthRainfall)


## Print Statistics.

print(stats.describe(monthRainfall))

print('\n Historical: \n ')
monthAccumulatedHistogram(month = month,allDataDataframe=precipitationAllTime, bins
```

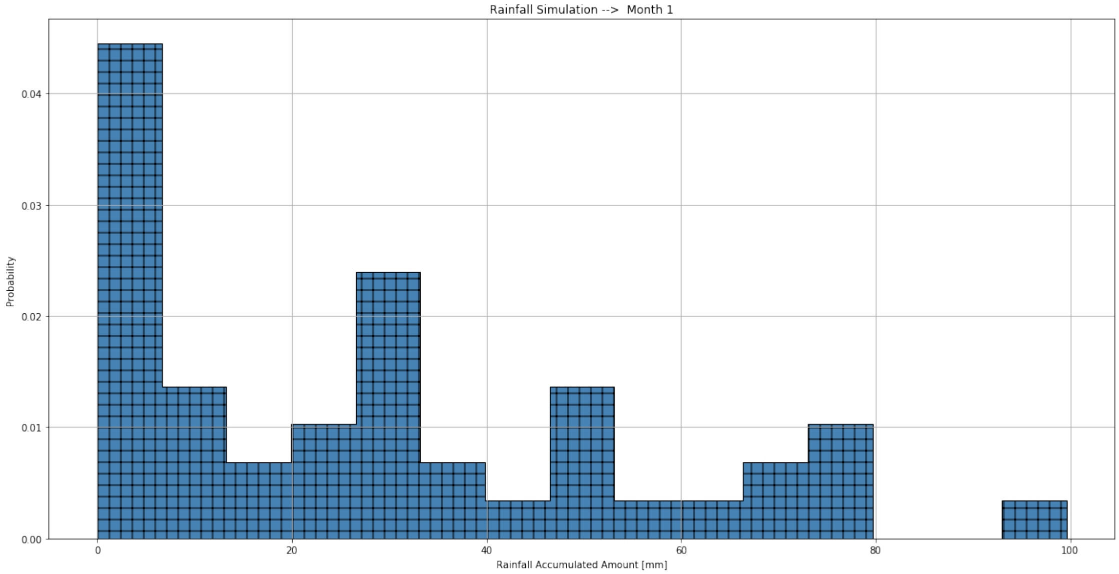
```
 Current Month is: 01

 Simulated:
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel_launcher.py:259: 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#
```



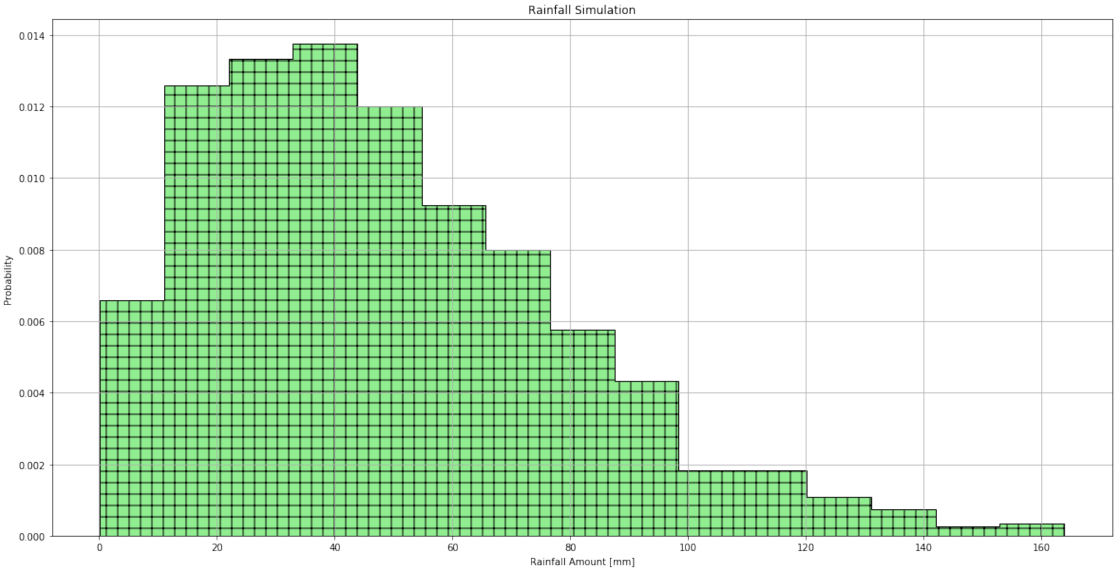
```
DescribeResult(nobs=1100, minmax=(0.05513549987306987, 131.94273673752022), mean=29.103333773055
```

```
 Historical:
```



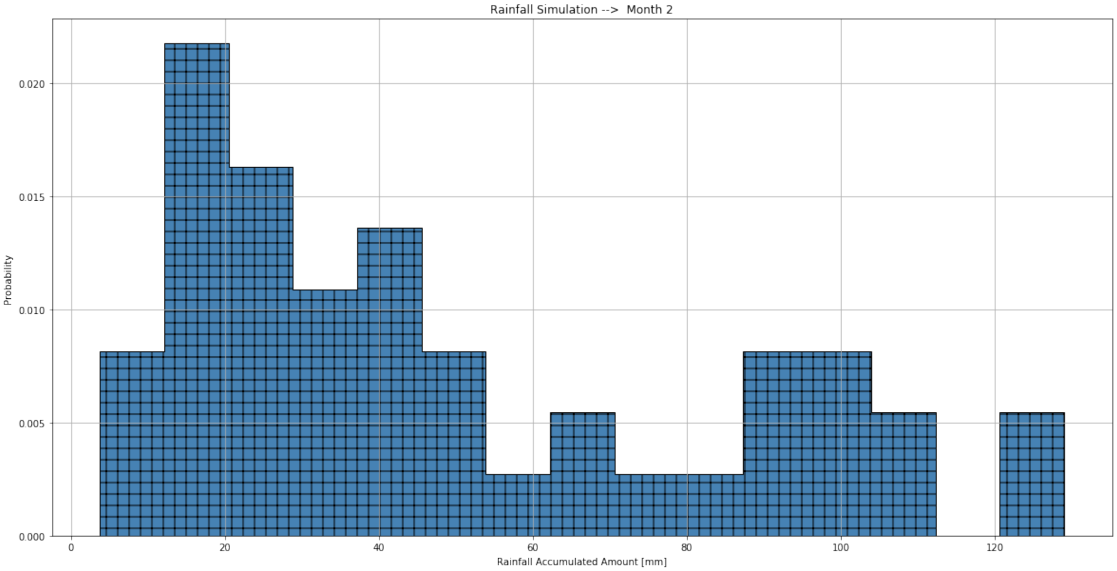
```
DescribeResult(nobs=44, minmax=(0.0, 99.6), mean=29.443181818181817, variance=700.718789640592,
```

```
 Current Month is: 02

 Simulated:
```



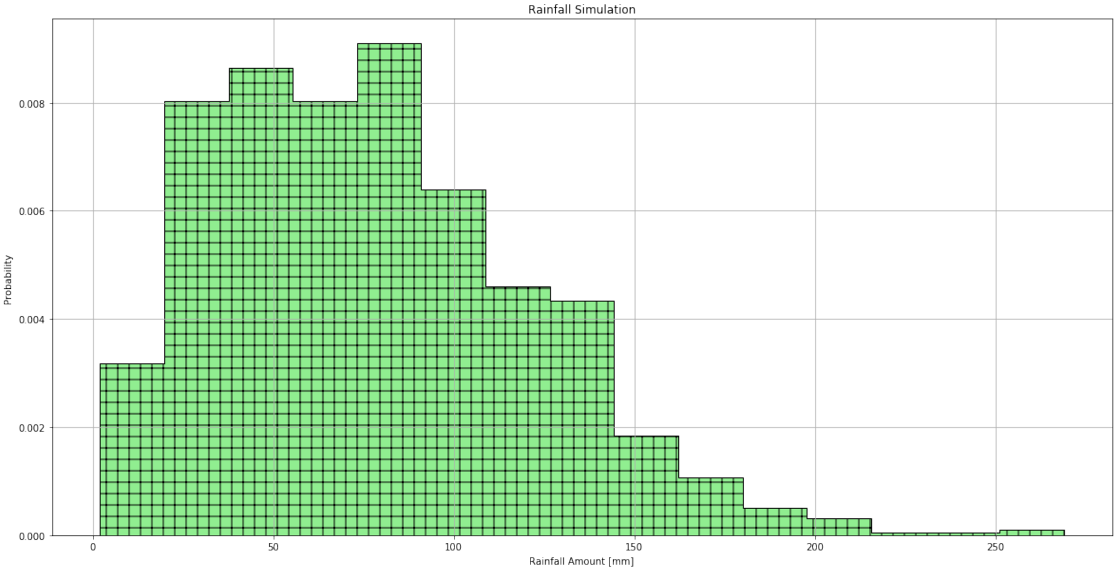
```
DescribeResult(nobs=1100, minmax=(0.17644626901446755, 163.78708252807257), mean=48.406537792354
```

```
 Historical:
```



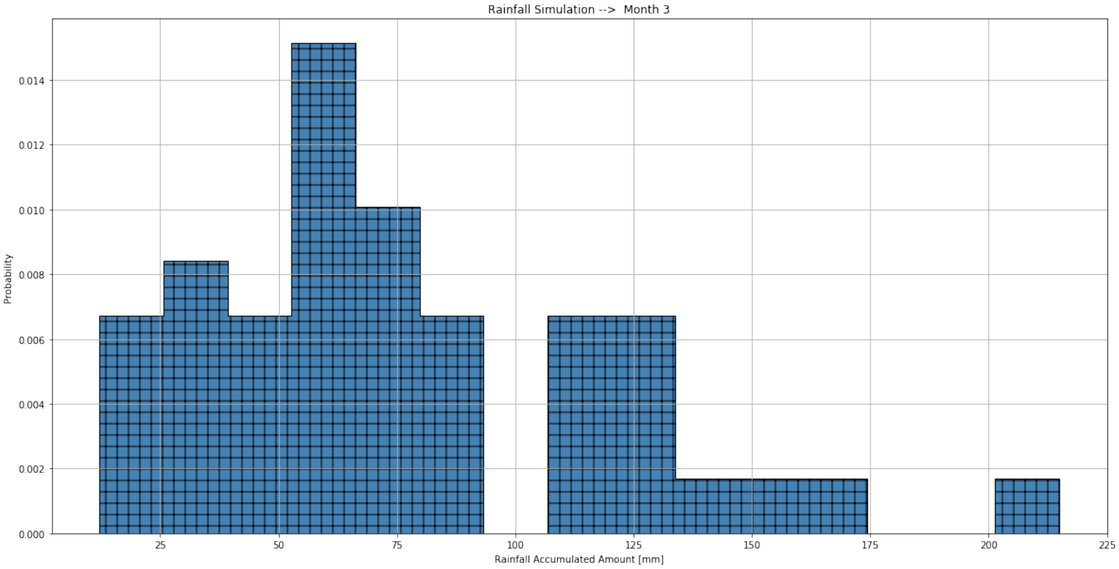
```
DescribeResult(nobs=44, minmax=(3.8000000000000003, 129.0), mean=49.93636363636363, variance=124
Current Month is: 03
```

```
Simulated:
```



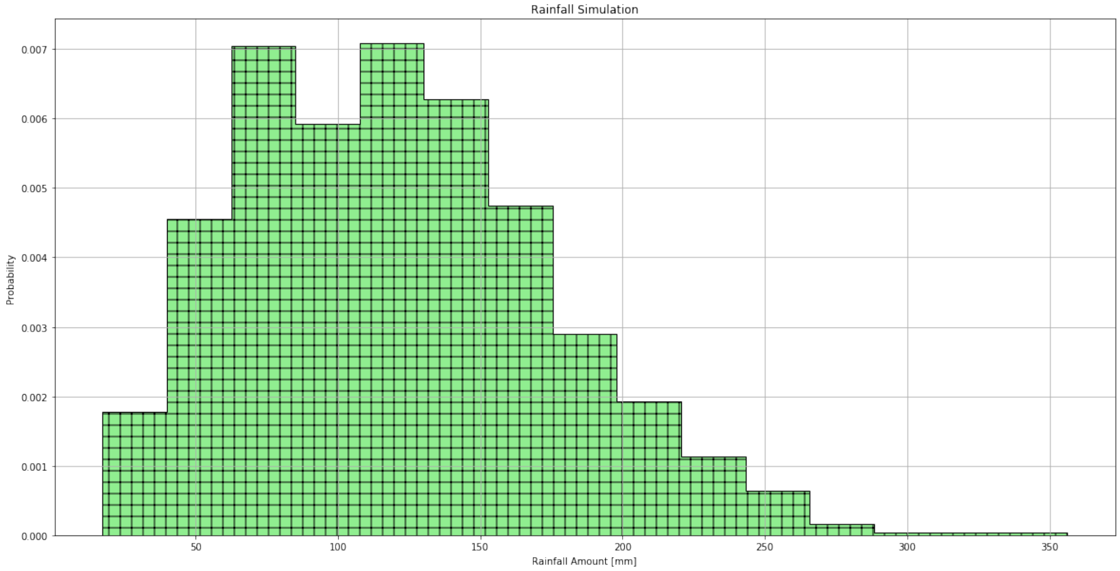
```
DescribeResult(nobs=1100, minmax=(2.0496224054395915, 268.8657497499154), mean=78.1211559132012,
```

```
Historical:
```



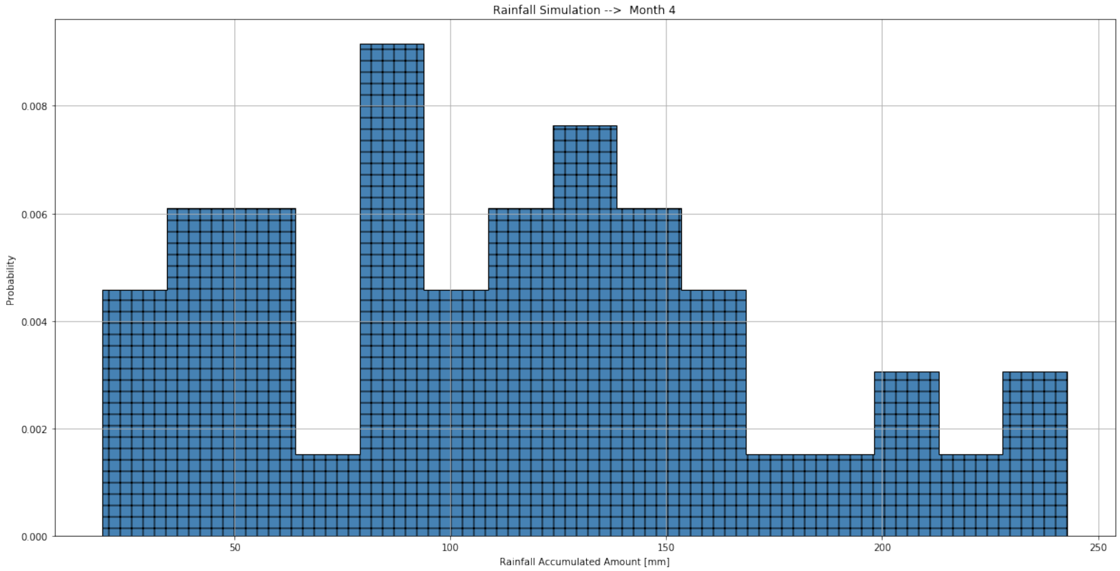
```
DescribeResult(nobs=44, minmax=(12.2, 214.90000000000003), mean=76.22500000000001, variance=1944
```

```
 Current Month is: 04

 Simulated:
```



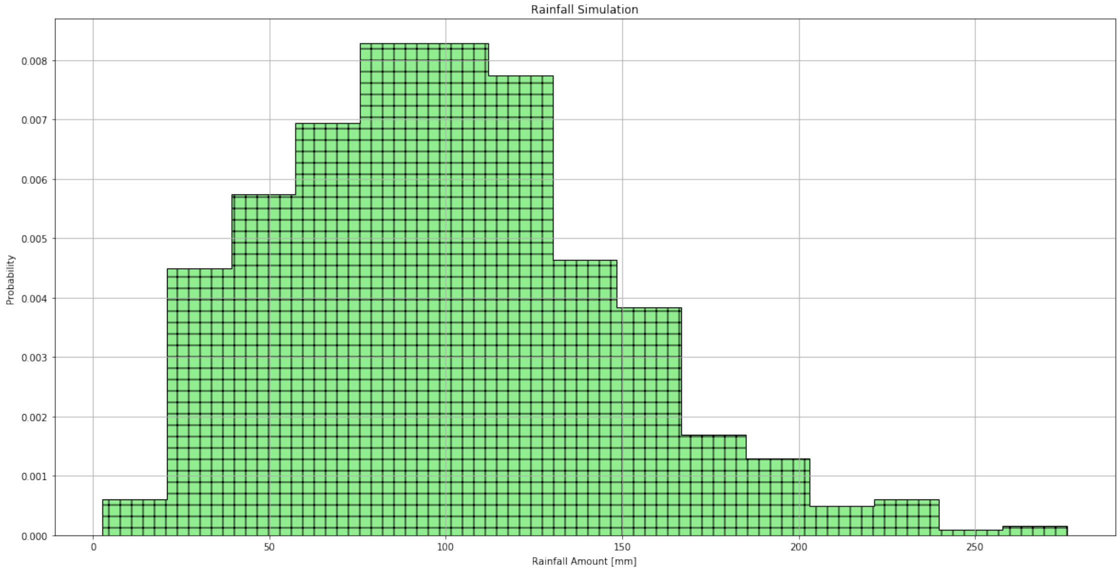
```
DescribeResult(nobs=1100, minmax=(17.243828036218734, 356.1151182315309), mean=120.5176214372795
```

```
 Historical:
```



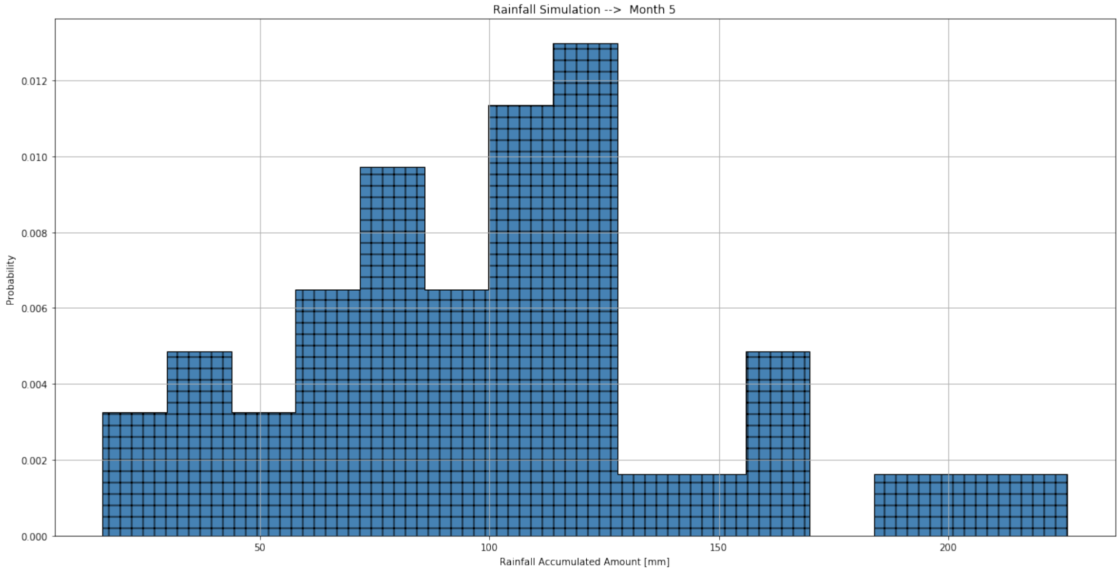
```
DescribeResult(nobs=44, minmax=(19.4, 242.8), mean=113.5431818181818, variance=3283.112743128964
```

```
Current Month is: 05

Simulated:
```



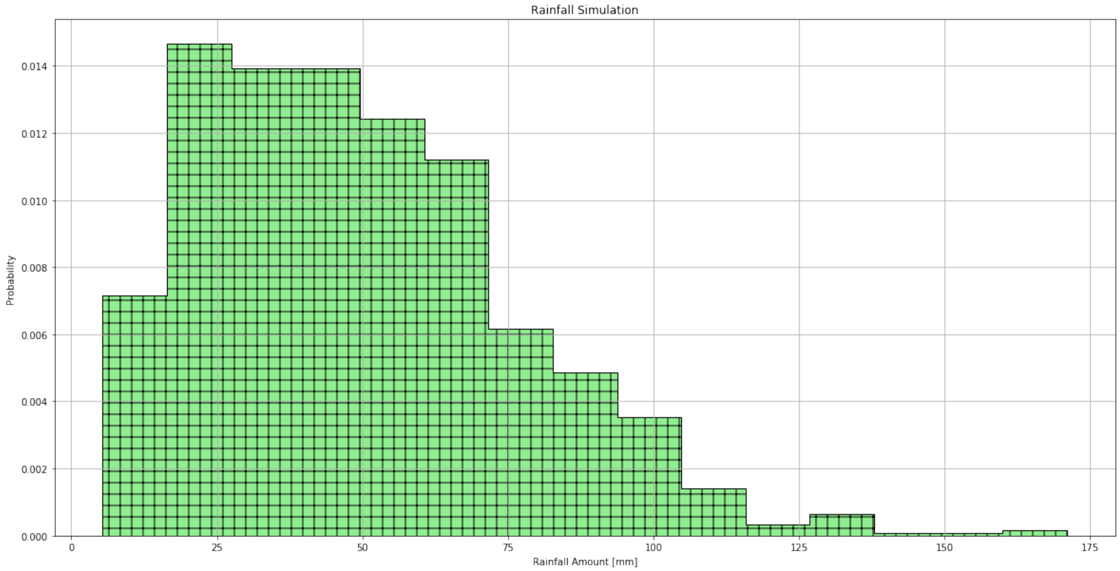
```
DescribeResult(nobs=1100, minmax=(2.741309551834239, 276.0568456950362), mean=100.12707108882195
```

```
 Historical:
```



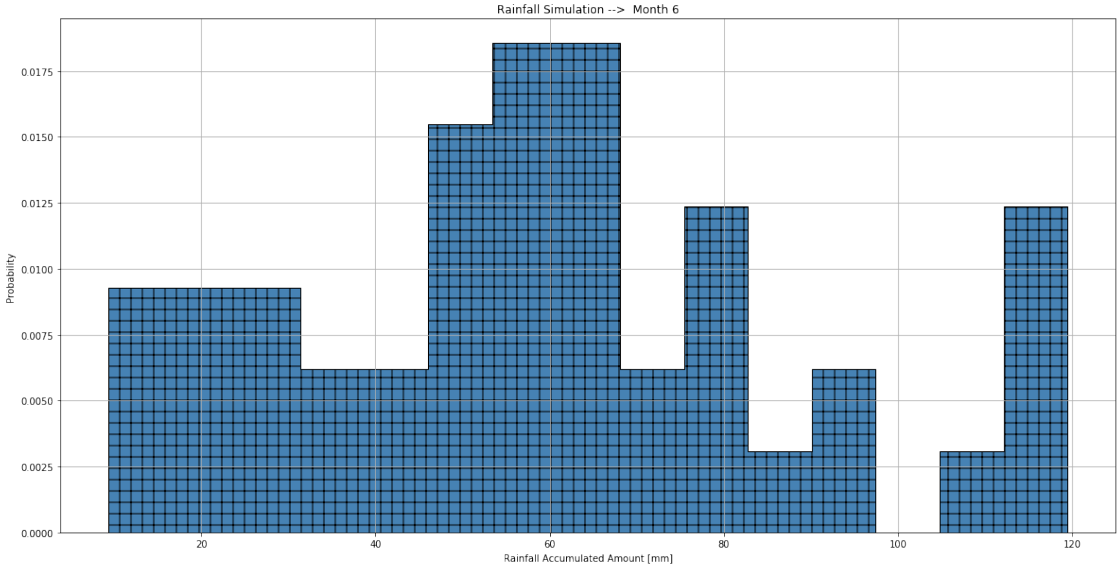
```
DescribeResult(nobs=44, minmax=(15.700000000000001, 225.9), mean=100.61136363636363, variance=21
```

```
 Current Month is: 06

 Simulated:
```

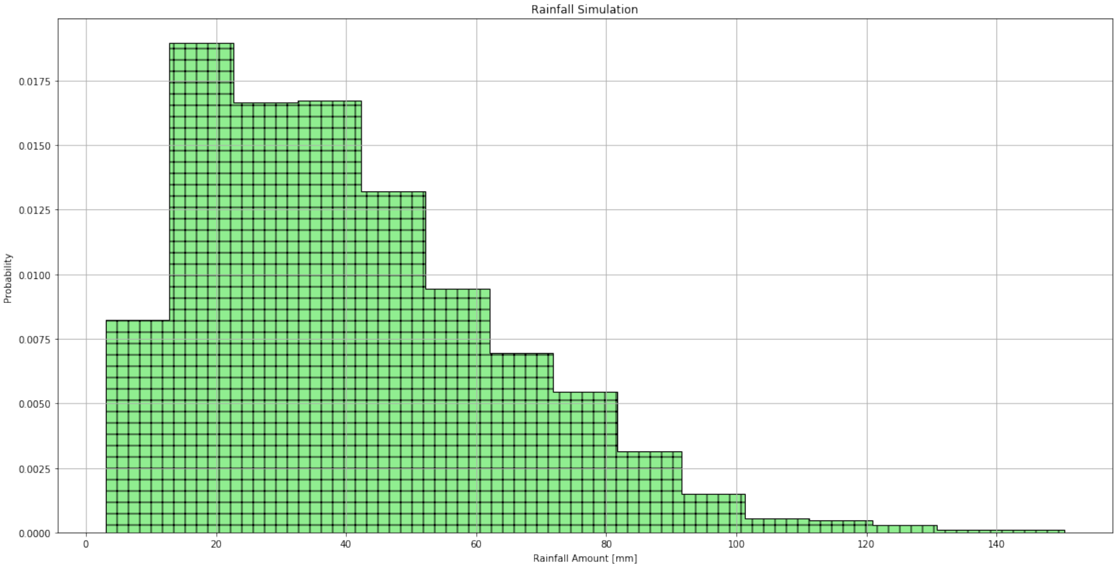


```
DescribeResult(nobs=1100, minmax=(5.386509859477281, 171.01672337750227), mean=49.45852516962775
```

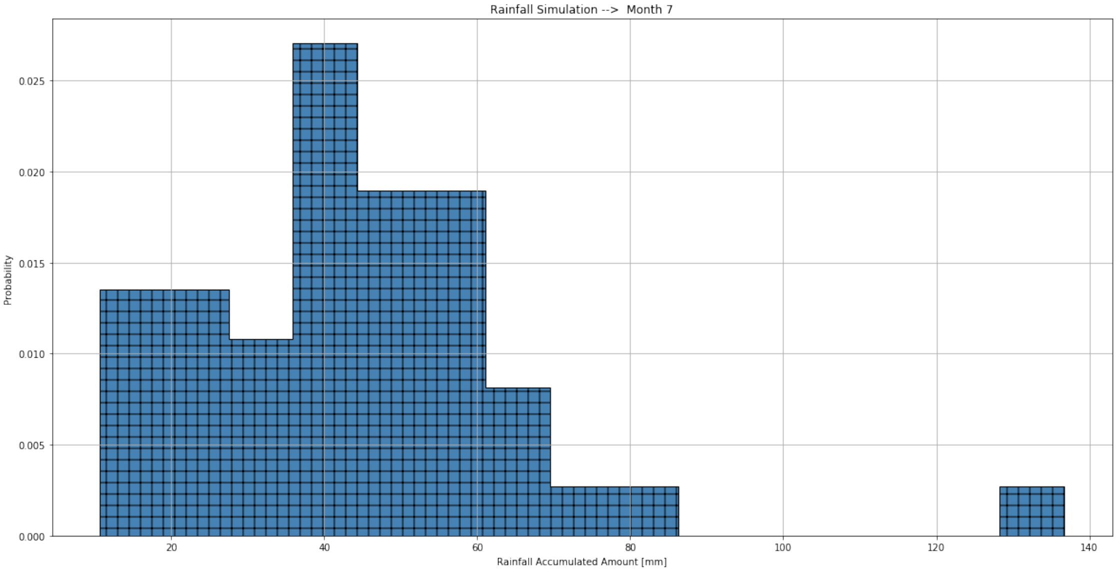
```
 Historical:
```



```
DescribeResult(nobs=44, minmax=(9.3, 119.5), mean=59.70681818181818, variance=866.8080919661734,
Current Month is: 07

Simulated:
```

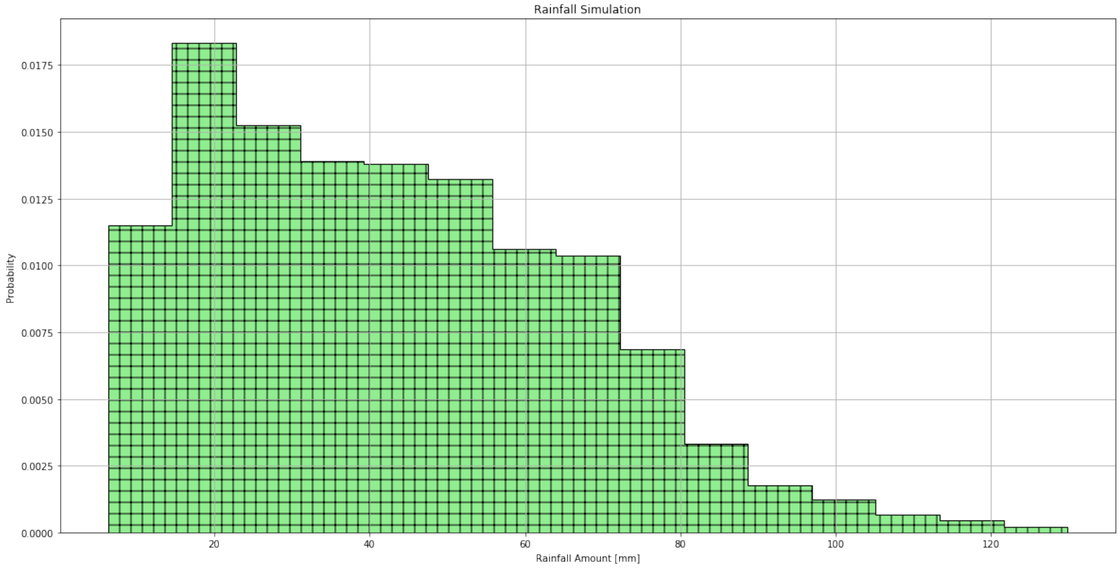


```
DescribeResult(nobs=1100, minmax=(3.029424013858446, 150.49867998637222), mean=40.5176409295524,

Historical:
```



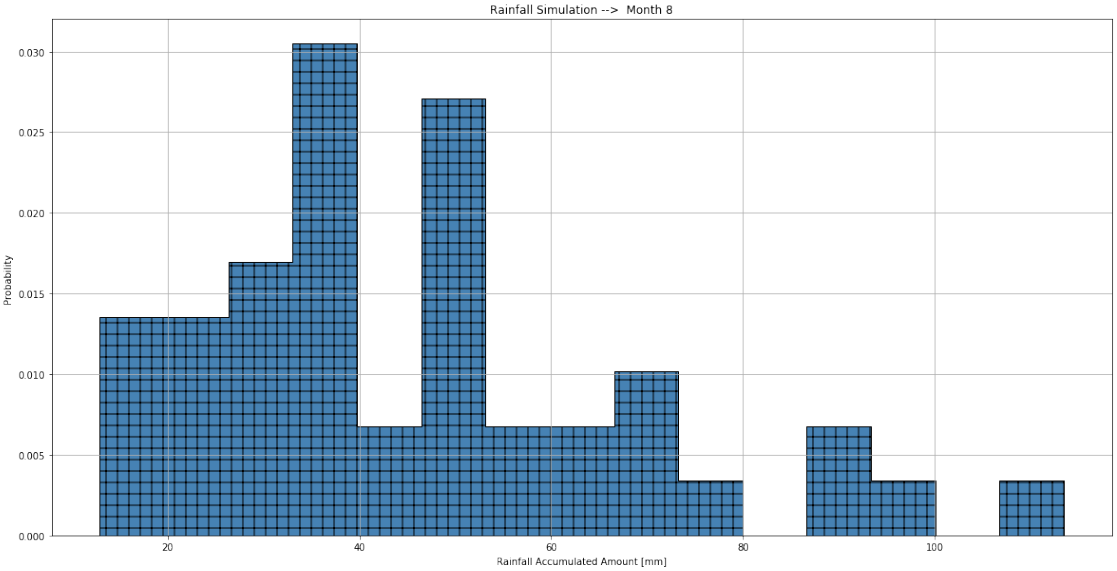
```
DescribeResult(nobs=44, minmax=(10.700000000000001, 136.7), mean=43.28181818181818, variance=490
```

```
 Current Month is: 08

 Simulated:
```



```
DescribeResult(nobs=1100, minmax=(6.348701620415779, 129.85821703841322), mean=43.07145981600916
```

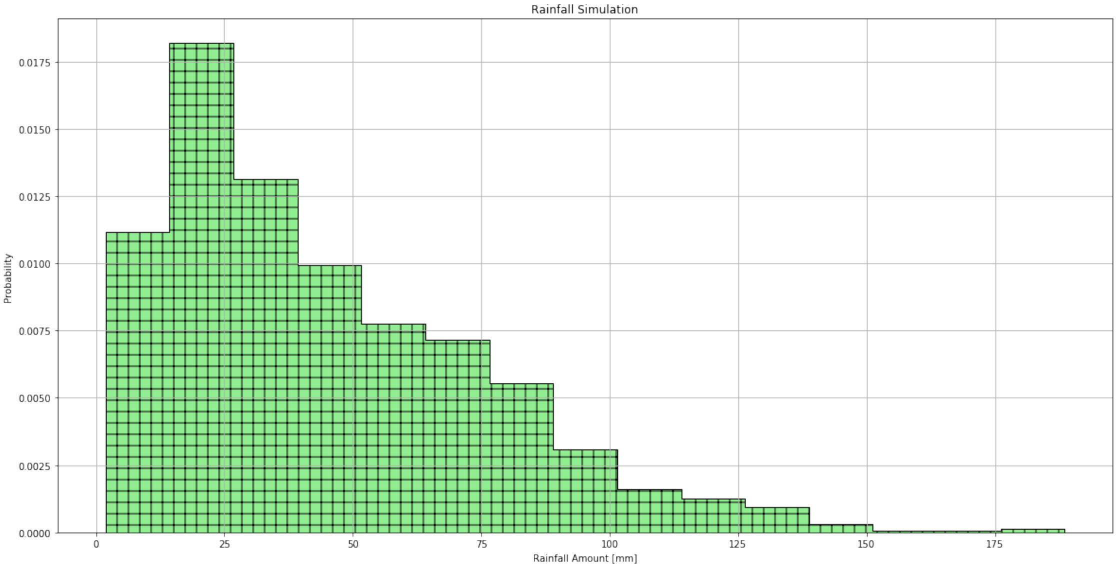
```
 Historical:
```



```
DescribeResult(nobs=44, minmax=(12.9, 113.49999999999999), mean=46.090909090909086, variance=525
```

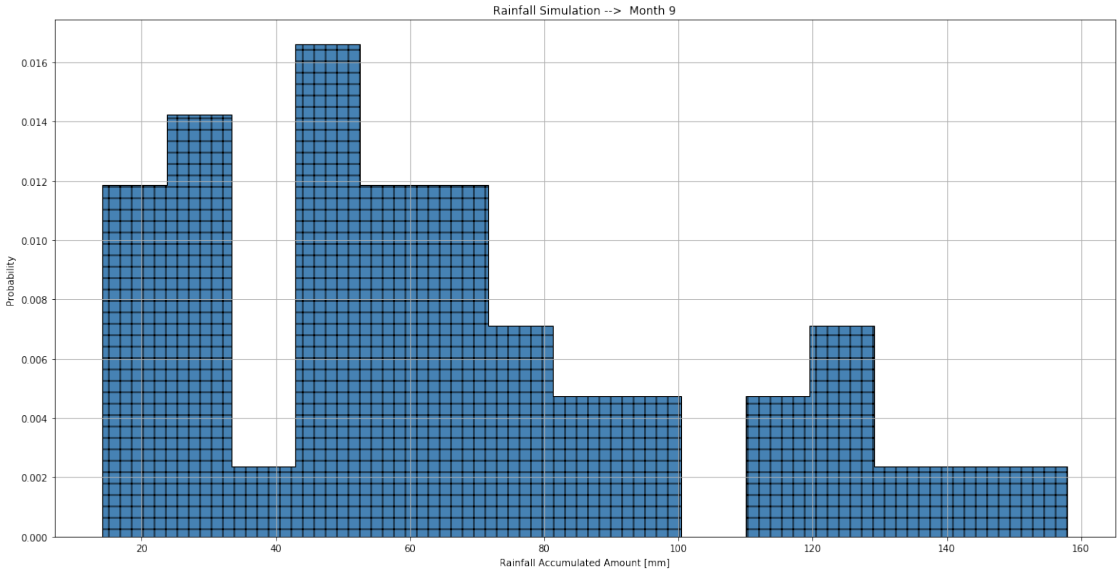
```
Current Month is: 09
```

```
Simulated:
```



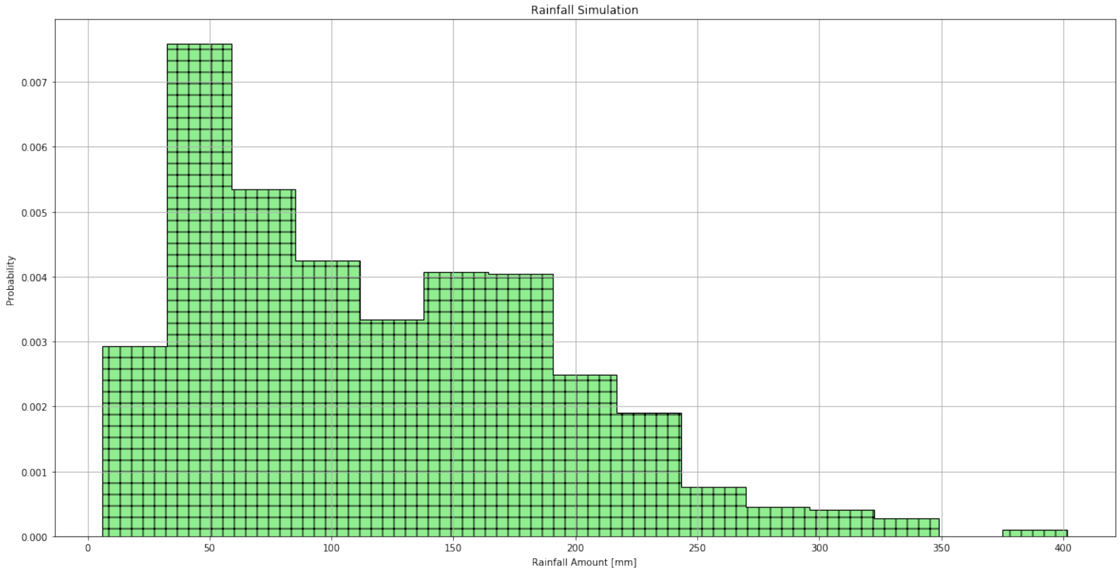
```
DescribeResult(nobs=1100, minmax=(1.8793159041974303, 188.58157675448538), mean=44.7288319425942
```

```
 Historical:
```



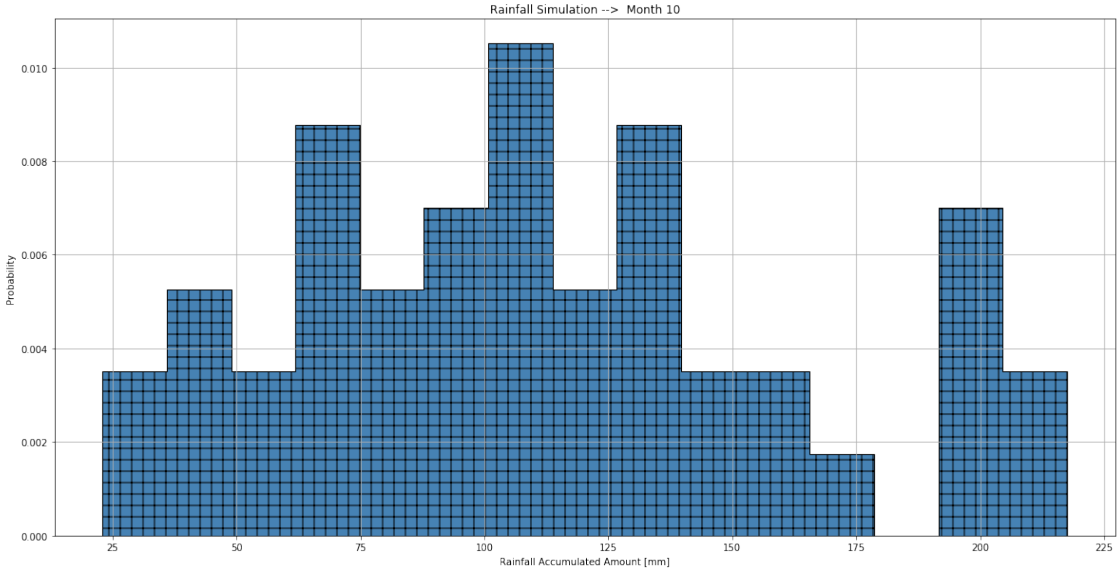
```
DescribeResult(nobs=44, minmax=(14.2, 157.89999999999998), mean=64.9409090909091, variance=1386.
```

```
 Current Month is: 10

 Simulated:
```



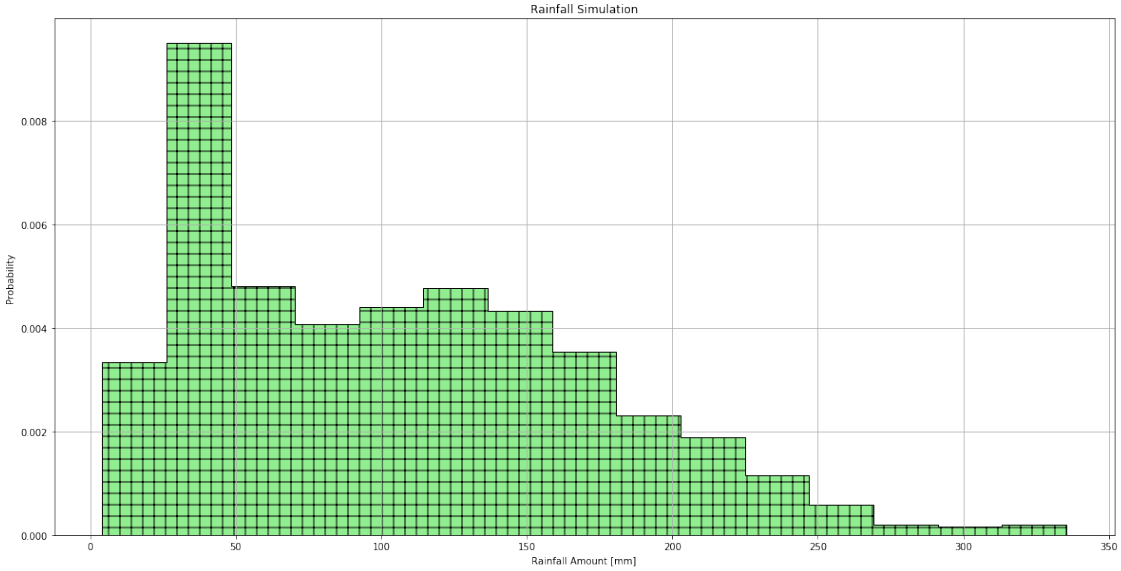
```
DescribeResult(nobs=1100, minmax=(6.123307497089689, 401.5869173452125), mean=118.53637349671315
```

```
 Historical:
```



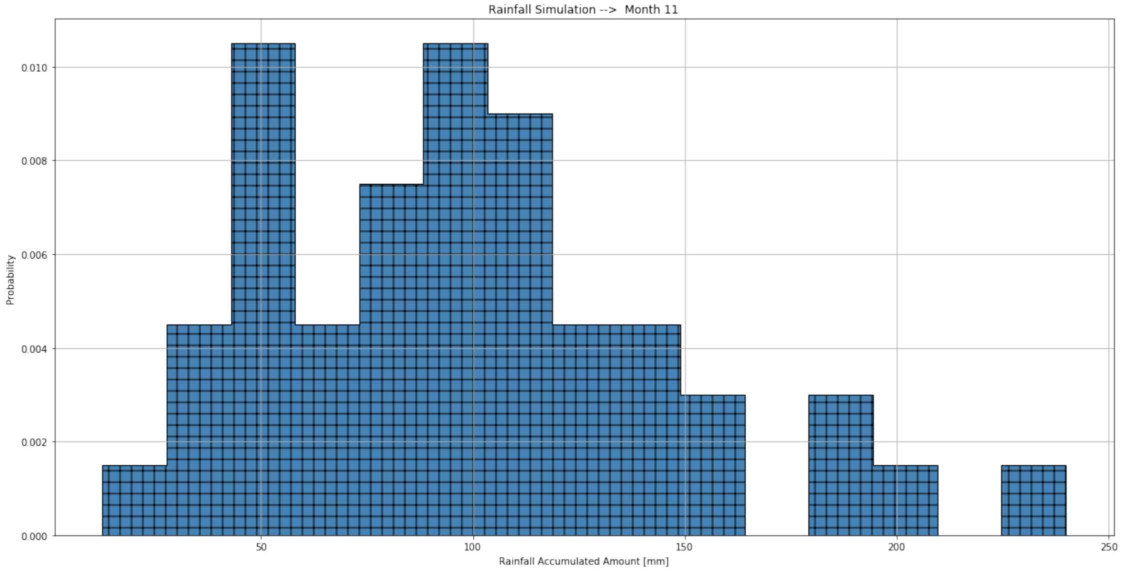
```
DescribeResult(nobs=44, minmax=(22.999999999999993, 217.50000000000003), mean=112.00227272727271
Current Month is: 11
```

```
Simulated:
```



```
DescribeResult(nobs=1100, minmax=(4.1145640644393895, 335.31966999611296), mean=105.282594937152
```

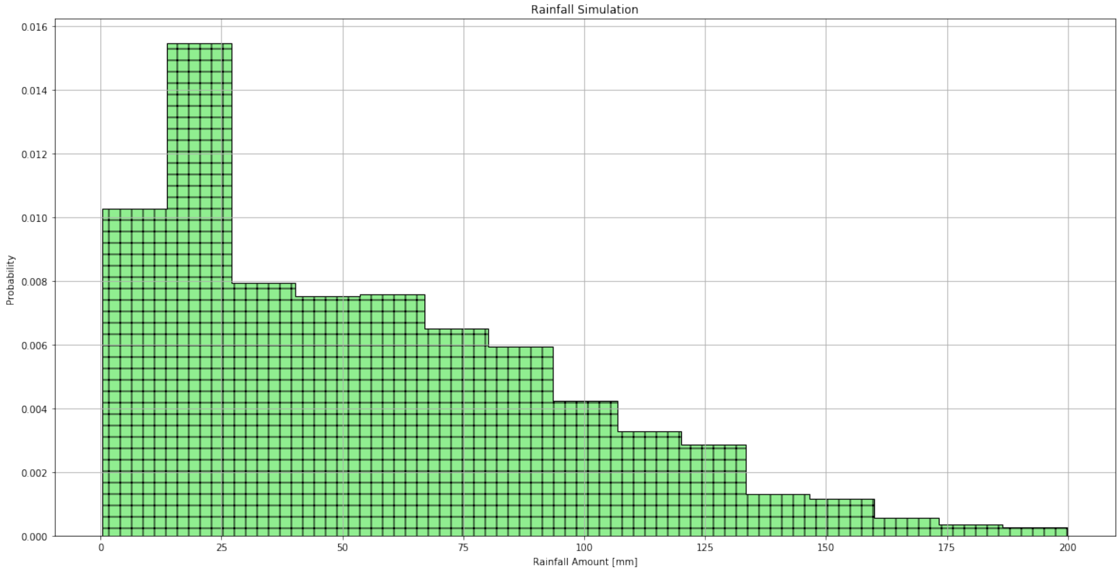
```
Historical:
```



```
DescribeResult(nobs=44, minmax=(12.6, 239.8), mean=98.42727272727275, variance=2329.687145877379
```

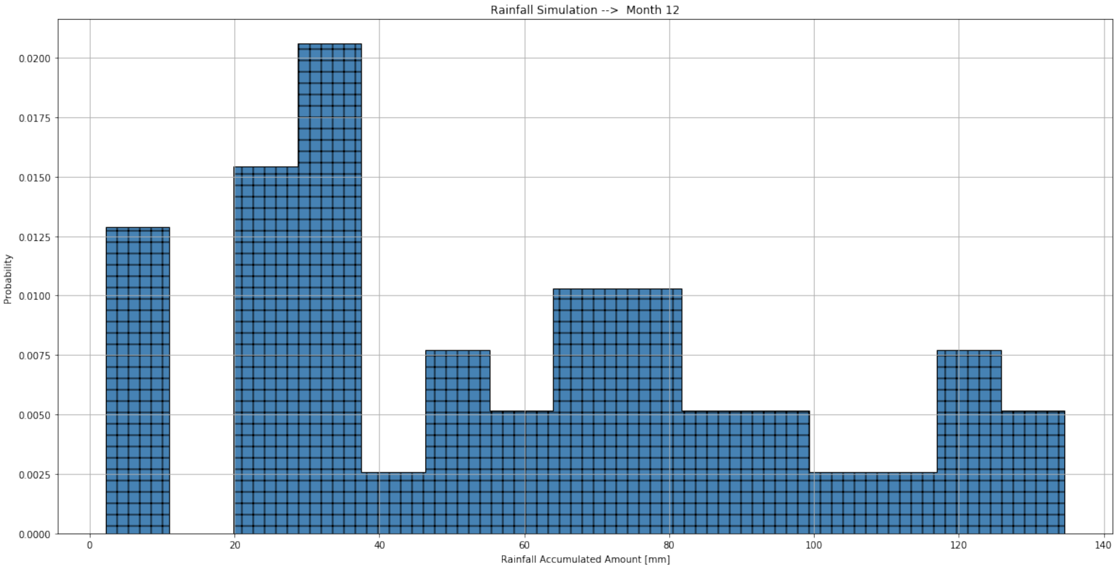
```
Current Month is: 12
```

```
Simulated:
```



```
DescribeResult(nobs=1100, minmax=(0.4432249118827425, 199.82581255590833), mean=55.619660405321
```

```
Historical:
```



```
DescribeResult(nobs=44, minmax=(2.2, 134.6), mean=57.525, variance=1420.5512209302326, skewness=
```