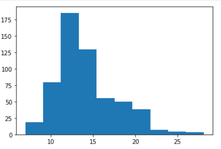
In [77]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import statsmodels.api as sm
import yfinance as yf
import math
import seaborn as sns

from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
from statsmodels import regression
from pandas_datareader import data as pdr
from scipy.stats import norm,rayleigh
from numpy import linspace
from pylab import plot,show,hist,figure,title
import scipy
from sklearn.preprocessing import StandardScaler
import scipy.stats
import matplotlib.pyplot as plt
%matplotlib inline
# Load data and select first column

from sklearn import datasets

yf.pdr_override()

df12 = pdr.get_data_yahoo("^GSPC", start="2000-07-03", end="2019-06-03").Close.rename('GSPC')

lreturns=np.log(df12/df12.shift(1))
lreturns.dropna(inplace=True)
# Create an index array (x) for data

x = np.arange(len(lreturns))
size = len(lreturns)
```

```
[**********************100%**********************]  1 of 1 downloaded
```

In [78]:
```
plt.hist(y)
plt.show()
y_df = pd.DataFrame(y, columns=['GSPC'])
y_df.describe()
```



Out[78]:

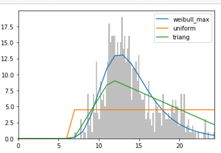
	GSPC
count	569.000000
mean	14.127292
std	3.524049
min	6.981000
25%	11.700000
50%	13.370000
75%	15.780000
max	28.110000

In [79]:
```
sc=StandardScaler()
yy = y.reshape (-1,1)
sc.fit(yy)
y_std =sc.transform(yy)
y_std = y_std.flatten()
y_std
del yy
```

In [80]:
```
# Set list of distributions to test
import warnings
warnings.filterwarnings("ignore")

dist_names = ['beta',
              'cauchy',
              'expon',
              'gamma',
              'lognorm',
              'norm',
              'pearson3',
              't',
              'triang',
              'uniform',
              'weibull_min',
              'weibull_max']

# Set up empty lists to stroe results
chi_square = []
p_values = []

# Set up 50 bins for chi-square test
# Observed data will be approximately evenly distrubuted aross all bins
percentile_bins = np.linspace(0,100,51)
percentile_cutoffs = np.percentile(y_std, percentile_bins)
observed_frequency, bins = (np.histogram(y_std, bins=percentile_cutoffs))
cum_observed_frequency = np.cumsum(observed_frequency)

# Loop through candidate distributions

for distribution in dist_names:
    # Set up distribution and get fitted distribution parameters
    dist = getattr(scipy.stats, distribution)
    param = dist.fit(y_std)

    # Obtain the KS test P statistic, round it to 5 decimal places
    p = scipy.stats.kstest(y_std, distribution, args=param)[1]
    p = np.around(p, 5)
    p_values.append(p)

    # Get expected counts in percentile bins
    # This is based on a 'cumulative distrubution function' (cdf)
    cdf_fitted = dist.cdf(percentile_cutoffs, *param[:-2], loc=param[-2],
                          scale=param[-1])
    expected_frequency = []
    for bin in range(len(percentile_bins)-1):
        expected_cdf_area = cdf_fitted[bin+1] - cdf_fitted[bin]
        expected_frequency.append(expected_cdf_area)

    # calculate chi-squared
    expected_frequency = np.array(expected_frequency) * size
    cum_expected_frequency = np.cumsum(expected_frequency)
    ss = sum (((cum_expected_frequency - cum_observed_frequency) ** 2) / cum_observed_frequency)
    chi_square.append(ss)

# Collate results and sort by goodness of fit (best at top)

results = pd.DataFrame()
results['Distribution'] = dist_names
results['chi_square'] = chi_square
results['p_value'] = p_values
results.sort_values(['chi_square'], inplace=True)

# Report results

print ('\nDistributions sorted by goodness of fit:')
print ('----------------------------------------')
print (results)
```

```
Distributions sorted by goodness of fit:
----------------------------------------
   Distribution    chi_square  p_value
11  weibull_max  2.936327e+04  0.00000
9       uniform  3.716549e+05  0.00000
8        triang  5.329889e+05  0.00000
5          norm  6.424674e+05  0.00000
1        cauchy  6.442738e+05  0.00000
7             t  6.881627e+05  0.00373
3         gamma  7.686847e+05  0.06151
6      pearson3  7.686880e+05  0.06152
0          beta  7.703603e+05  0.06558
4       lognorm  7.764086e+05  0.17957
2         expon  1.500323e+06  0.00000
10  weibull_min  2.292651e+06  0.00000
```

In [81]:
```
# Divide the observed data into 100 bins for plotting (this can be changed)
number_of_bins = 100
bin_cutoffs = np.linspace(np.percentile(y,0), np.percentile(y,99),number_of_bins)

# Create the plot
h = plt.hist(y, bins = bin_cutoffs, color='0.75')

# Get the top three distributions from the previous phase
number_distributions_to_plot = 3
dist_names = results['Distribution'].iloc[0:number_distributions_to_plot]

# Create an empty list to stroe fitted distribution parameters
parameters = []

# Loop through the distributions ot get line fit and paraemters

for dist_name in dist_names:
    # Set up distribution and store distribution paraemters
    dist = getattr(scipy.stats, dist_name)
    param = dist.fit(y)
    parameters.append(param)

    # Get line for each distribution (and scale to match observed data)
    pdf_fitted = dist.pdf(x, *param[:-2], loc=param[-2], scale=param[-1])
    scale_pdf = np.trapz (h[0], h[1][:-1]) / np.trapz (pdf_fitted, x)
    pdf_fitted *= scale_pdf

    # Add the line to the plot
    plot.plot(pdf_fitted, label=dist_name)

    # Set the plot x axis to contain 99% of the data
    # This can be removed, but sometimes outlier data makes the plot less clear
    plt.xlim(0,np.percentile(y,99))

# Add legend and display plot

plt.legend()
plt.show()

# Store distribution paraemters in a dataframe (this could also be saved)
dist_parameters = pd.DataFrame()
dist_parameters['Distribution'] = (
        results['Distribution'].iloc[0:number_distributions_to_plot])
dist_parameters['Distribution parameters'] = parameters

# Print parameter results
print ('\nDistribution parameters:')
print ('------------------------')

for index, row in dist_parameters.iterrows():
    print ('\nDistribution:', row[0])
    print ('Parameters:', row[1] )
```




```
Distribution parameters:
------------------------

Distribution: weibull_max
Parameters: (32310587.27708838, 88871942.46048762, 88871929.93109035)

Distribution: uniform
Parameters: (6.981, 21.128999999999998)

Distribution: triang
Parameters: (0.2148479672419735, 6.897186391486481, 21.28391041716865)
```

In [ ]:

In [ ]: