In [21]:
```
from scipy.stats import beta
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from sympy import *
from sympy.stats import Normal, cdf, density
import numpy as np
import pandas as pd
import math
import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint, adfuller
from statsmodels import regression,stats
import matplotlib.pyplot as plt
pd.set_option('display.expand_frame_repr', False)
pd.set_option("display.precision", 4)
```

In [2]:
```
def getoption(symbol, start_time, end_time, option_type, resolution):
    qb = QuantBook()
    contract = qb.AddOption(symbol)
    contract.SetFilter(-1, +1, timedelta(days=0), timedelta(days=60))
    option = qb.GetOptionHistory(contract.Symbol, start_time, end_time)
    option_history = option.GetAllData()
    if len(option_history) == 0:
        return [None, None]
    stock_history = option_history
    option_history.reset_index(inplace=True)
    stock_data = stock_history[stock_history["expiry"].astype(str)==
"NaT"][::resolution]
    expiry = option.GetExpiryDates()[-1]
    option_history = option_history[option_history['expiry'] == expiry]
    option_data = option_history[option_history['type']==option_type]
[::resolution]
    option_data =
option_data[option_data["strike"]==option_data["strike"].median().round()]

    return [option_data, stock_data]
```

In [3]:
```
def call_iv (r,C,K,S,dT):
    iv = Symbol('\sigma')
```

```
    d1 = (log (S/K)+(r+iv*iv/2)*dT)/iv/sqrt(dT)
    d2 = d1 - iv * sqrt(dT)
    x = Symbol ('x')
    X = Normal ('x', 0, 1)
    expr = S * simplify(cdf(X))(d1) - exp(-r * dT) * K *
simplify(cdf(X))(d2) - C
    func_np = lambdify(iv, expr, modules=['numpy'])
    max_iter=10
    curr_iter=0
    start_value=10
    check = False
    while check==False and curr_iter<max_iter:
        ans=fsolve(func_np,start_value)[0]
        check = np.isclose(func_np(ans),0)
        start_value=start_value/2
        curr_iter=curr_iter+1
    if check == False:
        print ("warning, not converge")
    return [ans,check]
```

In [4]:

```
def call_greeks (r,iv,K,S,dT):
    r_sym, iv_sym, K_sym, S_sym, dT_sym = symbols ('r \sigma K S dT')
    x_sym = Symbol ('x')
    X = Normal ('x', 0, 1)
    d1 = (log (S_sym/K_sym)+
(r_sym+iv_sym*iv_sym/2)*dT_sym)/iv_sym/sqrt(dT_sym)
    d2 = d1 - iv_sym * sqrt(dT_sym)
    C = S_sym * simplify(cdf(X))(d1) - exp(-r_sym * dT_sym) * K_sym *
simplify(cdf(X))(d2)
    delta = diff(C,S_sym).evalf(subs={r_sym:r, iv_sym:iv, K_sym:K,
S_sym:S, dT_sym:dT})
    gamma = diff(diff(C,S_sym),S_sym).evalf(subs={r_sym:r, iv_sym:iv,
K_sym:K, S_sym:S, dT_sym:dT})
    vega = iv/100*diff(C,iv_sym).evalf(subs={r_sym:r, iv_sym:iv,
K_sym:K, S_sym:S, dT_sym:dT})
    theta = -1/365*diff(C,dT_sym).evalf(subs={r_sym:r, iv_sym:iv,
K_sym:K, S_sym:S, dT_sym:dT})
    rho = r/100*diff(C,r_sym).evalf(subs={r_sym:r, iv_sym:iv, K_sym:K,
S_sym:S, dT_sym:dT})
    return np.array([delta, gamma, vega, theta, rho]).astype('float64')
```

In [5]:

```
start_time =  datetime(2021, 3, 1, 0, 0)
call_history = None
```

```
stock_history = None
resolution = 10
option_type = "Call"
symbol = "SPY"
for i in range(30):
    start_time = start_time + timedelta(days = i)
    end_time = start_time + timedelta(days = i+1)
    data_current = getoption(symbol, start_time, end_time, option_type, 
resolution)
    call_history = pd.concat([call_history, data_current[0]])
    stock_history = pd.concat([stock_history, data_current[1]])
call_history.reset_index(drop=True, inplace=True)
stock_history.reset_index(drop=True, inplace=True)
```

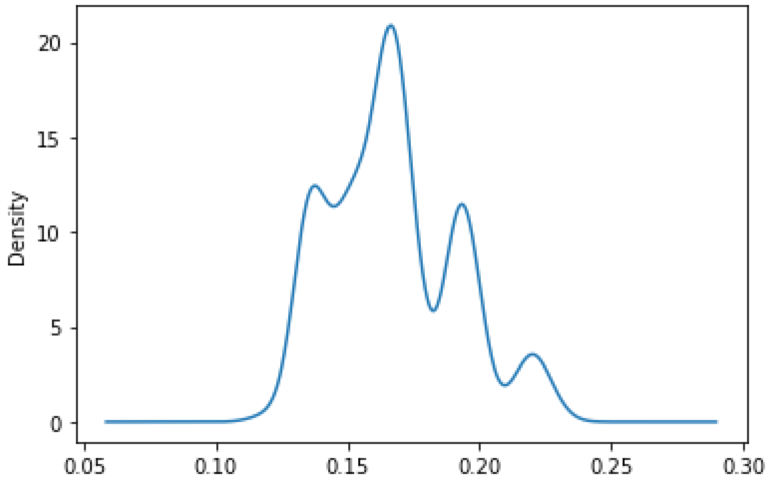
In [6]:

```
option_history = call_history
option_greeks = option_history[{'expiry','strike','type','time'}][:-1]
for newcolumn_name in ['imp_vol','delta','gamma','vega','theta','rho']:
    option_greeks[newcolumn_name]=np.nan
r = 0.0063
for i in range (len(option_history)-1):
#for i in range (10):
    Price=0.5*(option_history['askclose'][i]+option_history['bidclose']
[i])
    K=option_history['strike'][i]
    S=0.5*
(stock_history['askclose'].values[i]+stock_history['bidclose'].values[i])

    time_delta=option_history['expiry'][i]-option_history['time'][i]
    dT=(time_delta.seconds/3600/24+time_delta.days)/365 ##time measured 
in years
    opt_type=option_history['type'][i]
    if opt_type=='Call':
        iv=call_iv (r,Price,K,S,dT)[0]
        #print(call_iv (r,Price,K,S,dT)[1])
        greeks = call_greeks(r,iv,K,S,dT)
    if opt_type=='Put':
        iv=put_iv (r,Price,K,S,dT)[0]
        greeks = put_greeks(r,iv,K,S,dT)
    option_greeks.loc[i,'imp_vol']=iv
    option_greeks.loc[i,'delta':'rho']=greeks
option_greeks = 
option_greeks.set_index(option_greeks["time"]).drop(columns = "time")
```

In [7]:
```
option_greeks["imp_vol"].plot.kde()
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7faa444e2e80>



In [8]:
```
len(option_greeks)
```

Out[8]: 1208

In [118…
```
data = option_greeks["imp_vol"].values
data_trans = (data - data.min() )/ (data.max() - data.min())
x = np.arange (0, 1, 0.02)
```

In [119…
```
data_P = [sum (data_trans < i) / len(data_trans) for i in x]
```

In [120…
```
data_pdf = np.zeros (len(x))
data_pdf [0] = 0
for i in range (len(data_P)-1):
    data_pdf[i+1] = (data_P [i+1] - data_P [i] )/ (x[i+1] - x[i])
    np.outer (data_pdf)
```

Entropy Calculation

In [193…
```
def cal_entropy (pdf1, pdf2, gridsize):
    pdf_avg1 = (pdf1 [1:] + pdf1[:-1]) / 2
    pdf_avg2 = (pdf2 [1:] + pdf2[:-1]) /2
    pdf12 = np.outer (pdf_avg1, pdf_avg2)
    log_pdf12 = np.outer (pdf_avg1, pdf_avg2)
    return -(pdf12 * log_pdf12).sum() * gridsize**2
```

Beta Distribution

In [194…
```
beta_a = (exp * (1 - exp) / var - 1) * exp
```

```
beta_b = (exp * (1 - exp) /var -1) * (1 - exp)
B = math.gamma(beta_a) * math.gamma(beta_b) / math.gamma(beta_a+beta_b)
beta_pdf = (x**(beta_a-1) * (1-x)**(beta_b-1)) / B
```

In [195…

```
beta_entropy = cal_entropy (data_pdf, beta_pdf, x[1]-x[0])
print ("Beta Distribution Entropy is " + str(beta_entropy))
```

```
Beta Distribution Entropy is -2.575559882761846
```

Gamma Distribution

In [196…

```
gamma_a = exp**2 / var
gamma_b = exp / var
gamma_pdf = gamma_b ** gamma_a / math.gamma (gamma_a) * x ** (gamma_a -1
) * np.exp(-gamma_b * x)
```

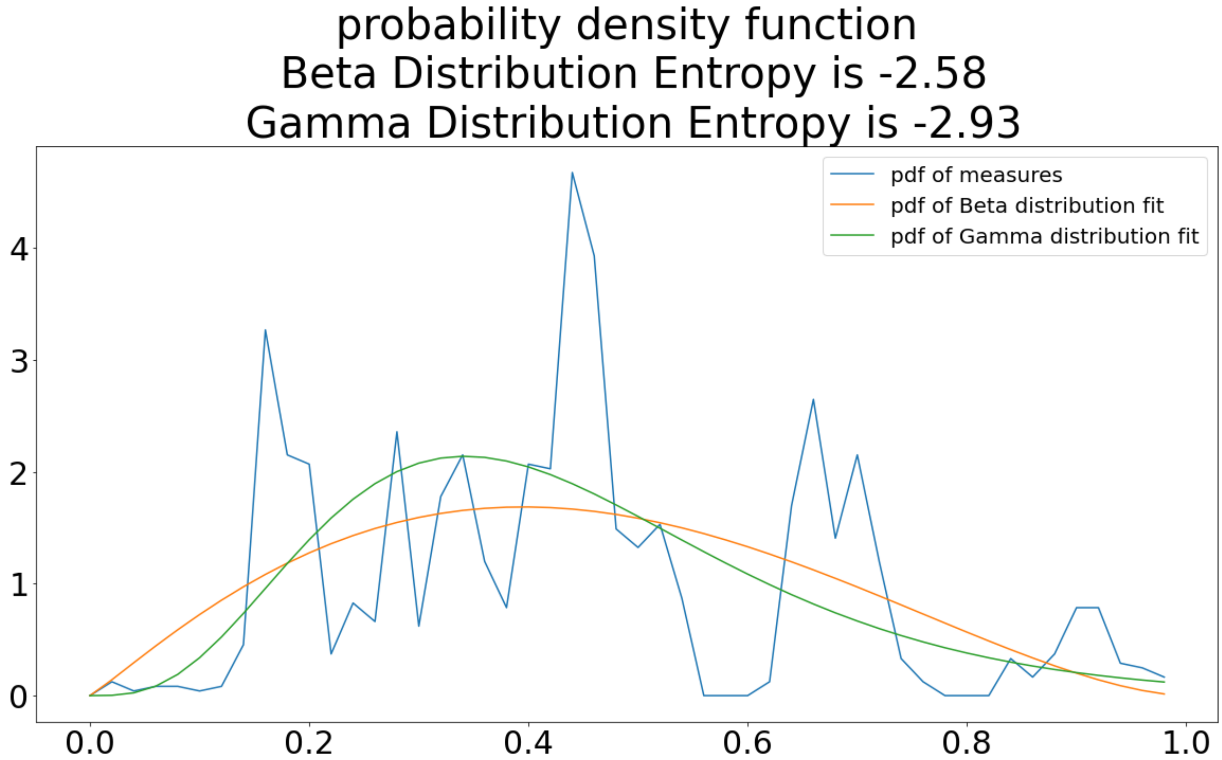
In [197…

```
gamma_entropy = cal_entropy (data_pdf, gamma_pdf, x[1]-x[0])
print ("Gamma Distribution Entropy is " + str(gamma_entropy))
```

```
Gamma Distribution Entropy is -2.930284111030331
```

In [198…

```
plt.figure(figsize=(20,10))
plt.plot(x, data_pdf, label = "pdf of measures")
plt.plot(x, beta_pdf, label = "pdf of Beta distribution fit")
plt.plot(x, gamma_pdf, label = "pdf of Gamma distribution fit")
plt.title("probability density function" + "\n Beta Distribution Entropy
is "+ str(round(beta_entropy,2))
          + "\n Gamma Distribution Entropy is "+
str(round(gamma_entropy,2)), fontsize=40)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
plt.legend(fontsize=20)
plt.show()
```



In [199…

```
print ("Gamma Distribution alpha = " + str (gamma_a))
print ("Gamma Distribution beta = " + str (gamma_b))
print ("Beta Distribution alpha = " + str (beta_a))
print ("Beta Distribution beta = " + str (beta_b))
```

```
Gamma Distribution alpha = 4.53005940022824
Gamma Distribution beta = 10.319263276474011
```

In [ ]:

```

```