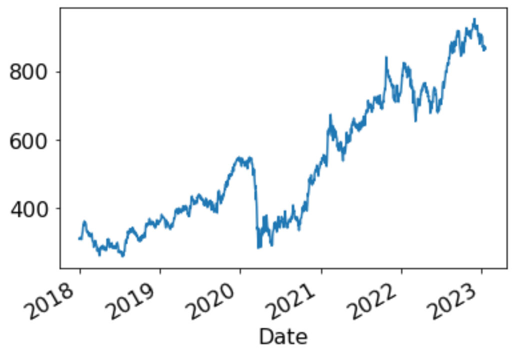
```
!pip install yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting yfinance
  Downloading yfinance-0.2.4-py2.py3-none-any.whl (51 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.4/51.4 KB 1.5 MB/s eta 0:00:00
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.8/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.3.5)
Collecting frozendict>=2.3.4
  Downloading frozendict-2.3.4-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 111.0/111.0 KB 5.6 MB/s eta 0:00:00
Collecting html5lib>=1.1
  Downloading html5lib-1.1-py2.py3-none-any.whl (112 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 112.2/112.2 KB 8.7 MB/s eta 0:00:00
Collecting cryptography>=3.3.2
  Downloading cryptography-39.0.0-cp36-abi3-manylinux_2_28_x86_64.whl (4.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 34.1 MB/s eta 0:00:00
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.8/dist-packages (from yfinance) (4.9.2)
Collecting beautifulsoup4>=4.11.1
  Downloading beautifulsoup4-4.11.1-py3-none-any.whl (128 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.2/128.2 KB 8.2 MB/s eta 0:00:00
Collecting requests>=2.26
  Downloading requests-2.28.2-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.8/62.8 KB 3.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.8/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.8/dist-packages (from yfinance) (2022.7)
Collecting soupsieve>1.2
  Downloading soupsieve-2.3.2.post1-py3-none-any.whl (37 kB)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.8/dist-packages (from cryptography>=3.3.2->yfinance) (1.15.1)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.8/dist-packages (from html5lib>=1.1->yfinance) (1.15.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.3.0->yfinance) (2.8
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (1.24
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2022.12
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from cffi>=1.12->cryptography>=3.3.2->yfinance)
Installing collected packages: soupsieve, requests, html5lib, frozendict, cryptography, beautifulsoup4, yfinance
  Attempting uninstall: requests
    Found existing installation: requests 2.25.1
    Uninstalling requests-2.25.1:
      Successfully uninstalled requests-2.25.1
  Attempting uninstall: html5lib
    Found existing installation: html5lib 1.0.1
    Uninstalling html5lib-1.0.1:
      Successfully uninstalled html5lib-1.0.1
  Attempting uninstall: beautifulsoup4
    Found existing installation: beautifulsoup4 4.6.3
    Uninstalling beautifulsoup4-4.6.3:
      Successfully uninstalled beautifulsoup4-4.6.3
Successfully installed beautifulsoup4-4.11.1 cryptography-39.0.0 frozendict-2.3.4 html5lib-1.1 requests-2.28.2 soupsieve-2.3.2.post1
```

```
import yfinance as yf
import numpy as np
import pandas as pd
import math as m
import matplotlib.pyplot as plt
import scipy
from scipy.stats import norm
```

```
data = yf.download("ICICIBANK.NS", start="2018-01-01", end="2023-01-20")
data['Close'].plot()
```

```
[*********************100%***********************]  1 of 1 completed
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb8df76be0>
```

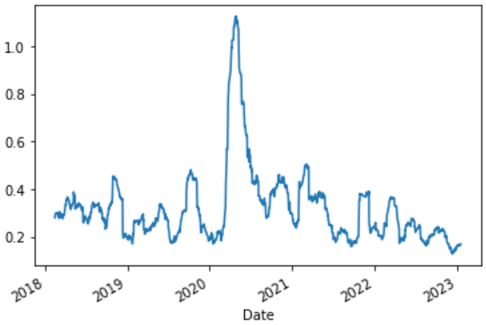



```
#Standard deviation measures how widely returns are dispersed from the average return. It’s the most common (and biased) estimator of vo
def standard_deviation(price_data, window=30, trading_periods=252, clean=True):
```

```
    log_return = (price_data["Close"] / price_data["Close"].shift(1)).apply(np.log)

    result = log_return.rolling(window=window, center=False).std() * m.sqrt(
        trading_periods
    )

    if clean:
        return result.dropna()
    else:
        return result
```

```
standard_deviation(data).plot()
```

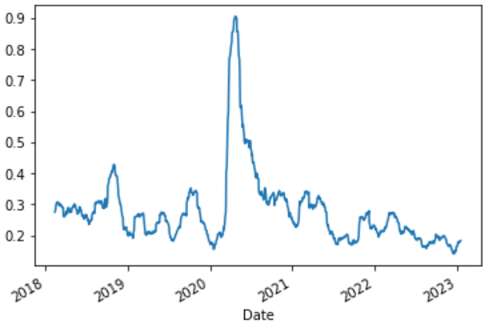
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb9438dcd0>




```
#Parkinson's volatility uses the stock's high and low price of the day rather than just close to close prices. It's useful to capture la
def parkinson(price_data, window=30, trading_periods=252, clean=True):

    rs = (1.0 / (4.0 * m.log(2.0))) * (
        (price_data["High"] / price_data["Low"]).apply(np.log)
    ) ** 2.0

    def f(v):
        return (trading_periods * v.mean()) ** 0.5

    result = rs.rolling(window=window, center=False).apply(func=f)

    if clean:
        return result.dropna()
    else:
        return result
```

```
parkinson(data).plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fdb93e654c0>

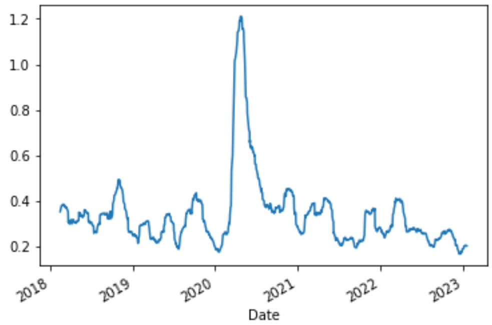



```
#volatility measure that handles both opening jumps and drift.
#It is the sum of the overnight volatility (close-to-open volatility) and a weighted average of open-to-close volatility.
#The assumption of continuous prices does mean the measure tends to slightly underestimate the volatility.
def yang_zhang(price_data, window=30, trading_periods=252, clean=True):

    log_ho = (price_data["High"] / price_data["Open"]).apply(np.log)
    log_lo = (price_data["Low"] / price_data["Open"]).apply(np.log)
    log_co = (price_data["Close"] / price_data["Open"]).apply(np.log)

    log_oc = (price_data["Open"] / price_data["Close"].shift(1)).apply(np.log)
    log_oc_sq = log_oc ** 2

    log_cc = (price_data["Close"] / price_data["Close"].shift(1)).apply(np.log)
    log_cc_sq = log_cc ** 2

    rs = log_ho * (log_ho - log_co) + log_lo * (log_lo - log_co)
```

```
    close_vol = log_cc_sq.rolling(window=window, center=False).sum() * (
        1.0 / (window - 1.0)
    )
    open_vol = log_oc_sq.rolling(window=window, center=False).sum() * (
        1.0 / (window - 1.0)
    )
    window_rs = rs.rolling(window=window, center=False).sum() * (1.0 / (window - 1.0))

    k = 0.34 / (1.34 + (window + 1) / (window - 1))
    result = (open_vol + k * close_vol + (1 - k) * window_rs).apply(
        np.sqrt
    ) * m.sqrt(trading_periods)

    if clean:
        return result.dropna()
    else:
        return result
```

```
yang_zhang(data).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb93e85b20>
```

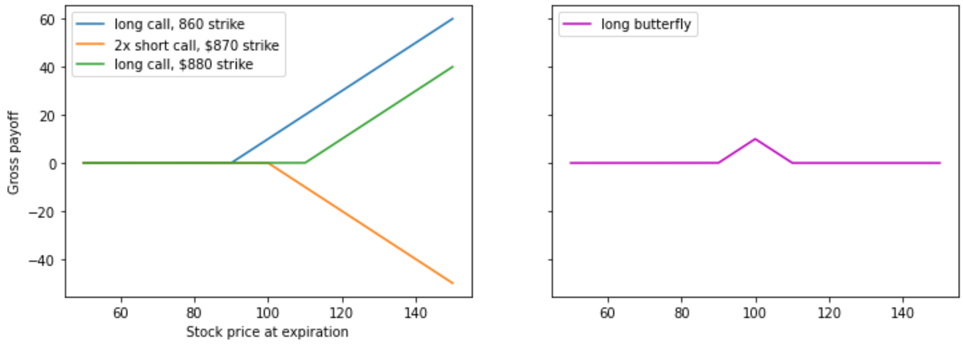



```
#Defining Black Schole Model
def option_value(option_type, S, K, sigma, t=0, r=0):
    """
    Calculate the value of an option using the Black-Scholes model

    :param option_type: "call"/"c" or "put"/"p"
    :type option_type: str
    :param S: price of the underlying
    :type S: float
    :param K: strike price of option
    :type K: float
    :param sigma: input implied volatility
    :type sigma: float
    :param t: time to expiration
    :type t: float, optional
    :param r: risk-free rate
    :type r: float, optional
    """
    with np.errstate(divide='ignore'):
        d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
        d2 = d1 - sigma * np.sqrt(t)
    if option_type.lower() in {"c", "call"}:
        return np.multiply(norm.cdf(d1),S) - np.multiply(norm.cdf(d2), K * np.exp(-r * t))
    elif option_type.lower() in {"p", "put"}:
        return -np.multiply(norm.cdf(-d1), S) + np.multiply(norm.cdf(-d2), K * np.exp(-r * t))
```

```
# Construction of a butterfly spread
S = np.linspace(50, 150, 1000)
C1 = option_value("c", S, 90, sigma=0.20)
C2 = -option_value("c", S, 100, sigma=0.20)
C3 = option_value("c", S, 110, sigma=0.20)
butterfly = C1 + 2 * C2 + C3
```

```
# (Gross) payoff diagram
fig, (ax, ax1) = plt.subplots(1,2, figsize=(12,4), sharey=True)
ax.plot(S, C1, S, C2, S, C3)
ax.set_xlabel("Stock price at expiration")
ax.set_ylabel("Gross payoff")
ax.legend(["long call, 860 strike", "2x short call, $870 strike", "long call, $880 strike"], loc="best")

ax1.plot(S, butterfly, c="m")
```

```
ax1.legend(["long butterfly"], loc="upper left")
# plt.show();
plt.savefig("long_butterfly.png", dpi=200)
```




```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving ICICI Call.xlsx to ICICI Call.xlsx
```

```
Call = pd.read_excel(uploaded.get('ICICI Call.xlsx'))
Call
```

	STRIKE	LTP	CHNG	BID	ASK	VOLUME	OI
0	590	-	-	248.40	318.65	NaN	NaN
1	600	-	-	239.40	303.95	NaN	NaN
2	610	-	-	245.40	275.50	NaN	1.0
3	620	-	-	224.15	280.95	NaN	NaN
4	630	-	-	212.25	270.45	NaN	NaN
5	640	-	-	203.05	262.25	NaN	NaN
6	650	-	-	194.55	245.55	NaN	NaN
7	660	-	-	198.70	224.55	NaN	NaN
8	670	-	-	186.25	213.90	NaN	NaN
9	680	-	-	176.90	203.25	NaN	NaN
10	690	-	-	167.35	192.80	NaN	NaN
11	700	169.45	-20.55	167.95	173.60	1.0	127.0
12	710	-	-	147.35	170.70	NaN	NaN
13	720	-	-	137.40	160.50	NaN	NaN
14	730	-	-	134.55	149.90	NaN	10.0
15	740	-	-	117.35	141.15	NaN	NaN
16	750	-	-	110.05	129.90	NaN	3.0
17	760	-	-	102.50	120.60	NaN	2.0
18	770	-	-	90.60	110.00	NaN	NaN
19	780	-	-	82.45	99.60	NaN	6.0
20	790	-	-	67.40	89.80	NaN	NaN
21	800	69	1.3	70.15	71.30	124.0	200.0
22	810	-	-	58.95	63.80	NaN	18.0
23	820	52.5	3.25	50.65	51.80	20.0	37.0
24	830	40.5	2.9	41.55	42.25	6.0	86.0
25	840	31.6	1.8	32.50	33.15	43.0	274.0
26	850	24.6	1.85	24.45	24.70	1352.0	560.0
27	860	17.55	1.35	17.45	17.70	3012.0	1133.0
28	870	12.45	1.25	12.35	12.45	9861.0	5026.0
29	880	8.4	0.9	8.30	8.40	8534.0	4950.0
30	890	5.65	0.45	5.60	5.65	4680.0	4219.0
31	900	3.75	0.1	3.70	3.75	7286.0	9775.0
32	910	2.45	0.05	2.40	2.45	4831.0	5372.0
33	920	1.7	0.1	1.65	1.70	2889.0	4828.0
34	930	1.2	0.1	1.20	1.25	1863.0	2048.0
35	940	0.85	-	0.85	0.90	734.0	1191.0
36	950	0.75	0.1	0.65	0.75	549.0	1588.0
37	960	0.55	0.05	0.55	0.60	406.0	831.0
38	970	0.45	0.05	0.40	0.45	164.0	477.0
39	980	0.35	0.05	0.35	0.40	61.0	341.0
40	990	0.3	0.05	0.25	0.40	6.0	281.0
41	1000	0.3	0.05	0.25	0.30	170.0	2347.0
42	1010	0.2	0.05	0.15	0.20	13.0	114.0
43	1020	0.15	0.05	0.05	0.15	8.0	152.0
44	1030	0.15	-	0.10	0.25	10.0	39.0
45	1040	0.05	-0.05	0.05	0.10	2.0	32.0
46	1050	0.15	-	0.10	0.15	7.0	205.0
47	1060	-	-	0.00	0.15	NaN	6.0
48	1070	-	-	0.00	0.20	NaN	1.0

49	1080	-	-	0.05	0.20	NaN	1.0
50	1090	0.25	-	0.05	0.25	1.0	NaN

```
Put = pd.read_excel(uploaded.get('ICICI Put.xlsx'))
Put
```

	STRIKE	LTP	CHNG	BID	ASK	VOLUME	OI
0	590	-	-	NaN	2.90	NaN	NaN
1	600	-	-	NaN	2.05	NaN	NaN
2	610	-	-	NaN	0.70	NaN	1.0
3	620	-	-	NaN	2.05	NaN	NaN
4	630	-	-	NaN	2.90	NaN	NaN
5	640	-	-	NaN	2.05	NaN	NaN
6	650	-	-	NaN	0.70	NaN	NaN
7	660	-	-	NaN	2.05	NaN	NaN
8	670	-	-	NaN	2.05	NaN	NaN
9	680	-	-	NaN	2.05	NaN	NaN
10	690	-	-	NaN	2.05	NaN	NaN
11	700	0.2	-	0.05	0.10	3.0	113.0
12	710	-	-	NaN	0.70	NaN	NaN
13	720	-	-	NaN	0.55	NaN	NaN
14	730	0.1	-0.05	NaN	0.30	1.0	3.0
15	740	-	-	NaN	1.05	NaN	NaN
16	750	0.2	-	0.05	0.20	1.0	87.0
17	760	-	-	NaN	0.60	NaN	5.0
18	770	0.15	-	NaN	0.10	1.0	198.0
19	780	0.1	-	0.05	0.20	6.0	80.0
20	790	0.2	0.1	0.10	0.20	11.0	360.0
21	800	0.2	-0.25	0.20	0.25	765.0	1361.0
22	810	0.35	-0.25	0.35	0.40	283.0	503.0
23	820	0.65	-0.55	0.60	0.70	868.0	653.0
24	830	1.15	-0.85	1.10	1.15	1478.0	1012.0
25	840	2.05	-1.3	2.00	2.10	1863.0	1536.0
26	850	3.75	-1.95	3.70	3.80	4155.0	3334.0
27	860	6.9	-2.3	6.85	6.95	3693.0	1791.0
28	870	11.6	-2.55	11.55	11.65	3582.0	1811.0
29	880	17.6	-3	17.50	17.70	861.0	1314.0
30	890	25.25	-1.65	24.65	25.00	145.0	728.0
31	900	33.5	-3	32.70	33.05	300.0	2279.0
32	910	42.5	-2.7	41.15	42.20	31.0	638.0
33	920	52.5	-2.45	50.30	51.05	19.0	540.0
34	930	61.5	-3.55	59.90	61.10	5.0	329.0
35	940	70	-4.6	69.40	70.75	10.0	523.0
36	950	81	-0.4	78.80	80.35	17.0	233.0
37	960	-	-	86.00	98.20	NaN	6.0
38	970	-	-	90.80	112.95	NaN	2.0
39	980	-	-	107.95	112.15	NaN	3.0
40	990	-	-	111.40	121.30	NaN	1.0
41	1000	130.5	-3.5	128.05	130.25	13.0	57.0
42	1010	-	-	130.15	152.85	NaN	NaN
43	1020	-	-	146.80	157.00	NaN	NaN
44	1030	-	-	150.15	172.85	NaN	NaN
45	1040	-	-	160.15	182.75	NaN	NaN
46	1050	-	-	170.05	192.75	NaN	NaN
47	1060	-	-	179.40	203.20	NaN	NaN
48	1070	-	-	188.85	213.90	NaN	NaN

49	1080	-	-	198.20	221.05	NaN	NaN
50	1090	-	-	193.65	245.80	NaN	NaN

```
Put.isnull().sum()
```

```
STRIKE     0
LTP        0
CHNG       0
BID       17
ASK        0
VOLUME    28
OI        22
dtype: int64
```

```
# Find midprices from bid/asks
Call["midprice"] = (Call.BID + Call.ASK)/2
Call = Call[Call.midprice > 0]
Put["midprice"] = (Put.BID + Put.ASK)/2
Put = Put[Put.midprice > 0]
Call.tail(30)
```

	STRIKE	LTP	CHNG	BID	ASK	VOLUME	OI	midprice
21	800	69	1.3	70.15	71.30	124.0	200.0	70.725
22	810	-	-	58.95	63.80	NaN	18.0	61.375
23	820	52.5	3.25	50.65	51.80	20.0	37.0	51.225
24	830	40.5	2.9	41.55	42.25	6.0	86.0	41.900
25	840	31.6	1.8	32.50	33.15	43.0	274.0	32.825
26	850	24.6	1.85	24.45	24.70	1352.0	560.0	24.575
27	860	17.55	1.35	17.45	17.70	3012.0	1133.0	17.575
28	870	12.45	1.25	12.35	12.45	9861.0	5026.0	12.400
29	880	8.4	0.9	8.30	8.40	8534.0	4950.0	8.350
30	890	5.65	0.45	5.60	5.65	4680.0	4219.0	5.625
31	900	3.75	0.1	3.70	3.75	7286.0	9775.0	3.725
32	910	2.45	0.05	2.40	2.45	4831.0	5372.0	2.425
33	920	1.7	0.1	1.65	1.70	2889.0	4828.0	1.675
34	930	1.2	0.1	1.20	1.25	1863.0	2048.0	1.225
35	940	0.85	-	0.85	0.90	734.0	1191.0	0.875
36	950	0.75	0.1	0.65	0.75	549.0	1588.0	0.700
37	960	0.55	0.05	0.55	0.60	406.0	831.0	0.575
38	970	0.45	0.05	0.40	0.45	164.0	477.0	0.425
39	980	0.35	0.05	0.35	0.40	61.0	341.0	0.375
40	990	0.3	0.05	0.25	0.40	6.0	281.0	0.325
41	1000	0.3	0.05	0.25	0.30	170.0	2347.0	0.275
42	1010	0.2	0.05	0.15	0.20	13.0	114.0	0.175
43	1020	0.15	0.05	0.05	0.15	8.0	152.0	0.100
44	1030	0.15	-	0.10	0.25	10.0	39.0	0.175
45	1040	0.05	-0.05	0.05	0.10	2.0	32.0	0.075
46	1050	0.15	-	0.10	0.15	7.0	205.0	0.125
47	1060	-	-	0.00	0.15	NaN	6.0	0.075
48	1070	-	-	0.00	0.20	NaN	1.0	0.100
49	1080	-	-	0.05	0.20	NaN	1.0	0.125
50	1090	0.25	-	0.05	0.25	1.0	NaN	0.150

```
Call.iloc[:50].reset_index(drop=True)
```

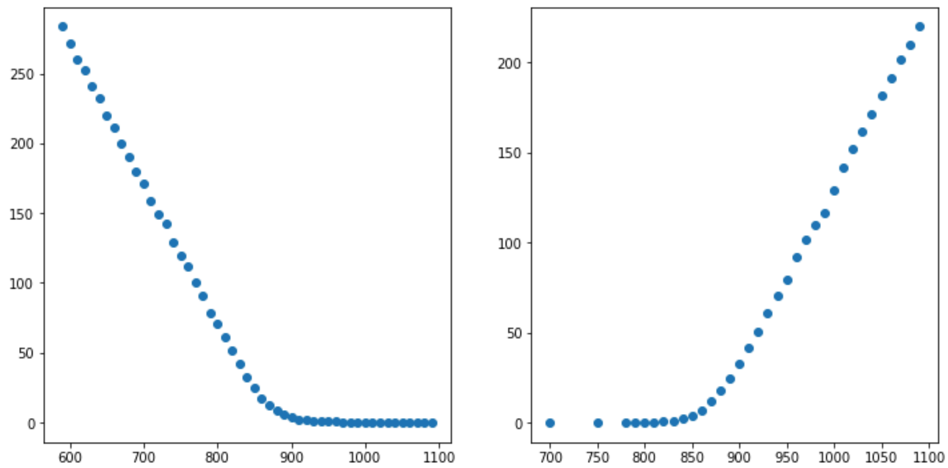
	STRIKE	LTP	CHNG	BID	ASK	VOLUME	OI	midprice
0	590	-	-	248.40	318.65	NaN	NaN	283.525
1	600	-	-	239.40	303.95	NaN	NaN	271.675
2	610	-	-	245.40	275.50	NaN	1.0	260.450
3	620	-	-	224.15	280.95	NaN	NaN	252.550
4	630	-	-	212.25	270.45	NaN	NaN	241.350
5	640	-	-	203.05	262.25	NaN	NaN	232.650
6	650	-	-	194.55	245.55	NaN	NaN	220.050
7	660	-	-	198.70	224.55	NaN	NaN	211.625
8	670	-	-	186.25	213.90	NaN	NaN	200.075
9	680	-	-	176.90	203.25	NaN	NaN	190.075
10	690	-	-	167.35	192.80	NaN	NaN	180.075
11	700	169.45	-20.55	167.95	173.60	1.0	127.0	170.775
12	710	-	-	147.35	170.70	NaN	NaN	159.025
13	720	-	-	137.40	160.50	NaN	NaN	148.950
14	730	-	-	134.55	149.90	NaN	10.0	142.225
15	740	-	-	117.35	141.15	NaN	NaN	129.250
16	750	-	-	110.05	129.90	NaN	3.0	119.975
17	760	-	-	102.50	120.60	NaN	2.0	111.550
18	770	-	-	90.60	110.00	NaN	NaN	100.300
19	780	-	-	82.45	99.60	NaN	6.0	91.025
20	790	-	-	67.40	89.80	NaN	NaN	78.600
21	800	69	1.3	70.15	71.30	124.0	200.0	70.725
22	810	-	-	58.95	63.80	NaN	18.0	61.375
23	820	52.5	3.25	50.65	51.80	20.0	37.0	51.225
24	830	40.5	2.9	41.55	42.25	6.0	86.0	41.900
25	840	31.6	1.8	32.50	33.15	43.0	274.0	32.825
26	850	24.6	1.85	24.45	24.70	1352.0	560.0	24.575
27	860	17.55	1.35	17.45	17.70	3012.0	1133.0	17.575
28	870	12.45	1.25	12.35	12.45	9861.0	5026.0	12.400
29	880	8.4	0.9	8.30	8.40	8534.0	4950.0	8.350
30	890	5.65	0.45	5.60	5.65	4680.0	4219.0	5.625
31	900	3.75	0.1	3.70	3.75	7286.0	9775.0	3.725
32	910	2.45	0.05	2.40	2.45	4831.0	5372.0	2.425
33	920	1.7	0.1	1.65	1.70	2889.0	4828.0	1.675
34	930	1.2	0.1	1.20	1.25	1863.0	2048.0	1.225
35	940	0.85	-	0.85	0.90	734.0	1191.0	0.875
36	950	0.75	0.1	0.65	0.75	549.0	1588.0	0.700
37	960	0.55	0.05	0.55	0.60	406.0	831.0	0.575
38	970	0.45	0.05	0.40	0.45	164.0	477.0	0.425
39	980	0.35	0.05	0.35	0.40	61.0	341.0	0.375
40	990	0.3	0.05	0.25	0.40	6.0	281.0	0.325
41	1000	0.3	0.05	0.25	0.30	170.0	2347.0	0.275
42	1010	0.2	0.05	0.15	0.20	13.0	114.0	0.175
43	1020	0.15	0.05	0.05	0.15	8.0	152.0	0.100
44	1030	0.15	-	0.10	0.25	10.0	39.0	0.175
45	1040	0.05	-0.05	0.05	0.10	2.0	32.0	0.075
46	1050	0.15	-	0.10	0.15	7.0	205.0	0.125
47	1060	-	-	0.00	0.15	NaN	6.0	0.075
48	1070	-	-	0.00	0.20	NaN	1.0	0.100

49	1080	-	-	0.05	0.20	NaN	1.0	0.125

```
Put.head(10)
```

	STRIKE	LTP	CHNG	BID	ASK	VOLUME	OI	midprice
11	700	0.2	-	0.05	0.10	3.0	113.0	0.075
16	750	0.2	-	0.05	0.20	1.0	87.0	0.125
19	780	0.1	-	0.05	0.20	6.0	80.0	0.125
20	790	0.2	0.1	0.10	0.20	11.0	360.0	0.150
21	800	0.2	-0.25	0.20	0.25	765.0	1361.0	0.225
22	810	0.35	-0.25	0.35	0.40	283.0	503.0	0.375
23	820	0.65	-0.55	0.60	0.70	868.0	653.0	0.650
24	830	1.15	-0.85	1.10	1.15	1478.0	1012.0	1.125
25	840	2.05	-1.3	2.00	2.10	1863.0	1536.0	2.050
26	850	3.75	-1.95	3.70	3.80	4155.0	3334.0	3.750

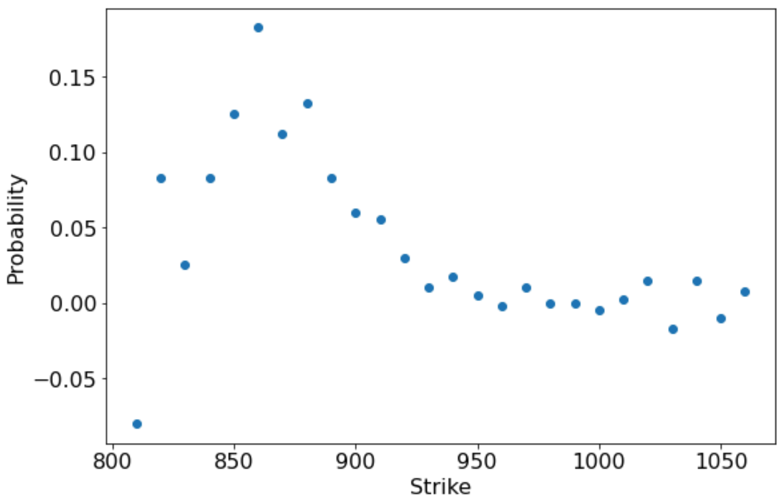
```
# Visualise put and call prices
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(12,6))
ax0.scatter(Call.STRIKE, Call.midprice);
ax1.scatter(Put.STRIKE, Put.midprice);
plt.show()
```




```
# Construct butterflies
data = []

for (_, left) ,(_,centre), (_, right) in zip(Call.iterrows(), Call.iloc[1:].iterrows(), Call.iloc[2:].iterrows()):
    # Filter out all zero volume
    if not any(vol > 0 for vol in {left.VOLUME, centre.VOLUME, right.VOLUME}):
        continue
    # Filter out any zero open interest
    if not all(oi > 0 for oi in {left.OI, centre.OI, right.OI}):
        continue
    # Equidistant on either end
    if centre.STRIKE - left.STRIKE != right.STRIKE - centre.STRIKE:
        continue
    butterfly_price = left.midprice - 2* centre.midprice + right.midprice
    max_profit = centre.STRIKE - left.STRIKE
    data.append([centre.STRIKE, butterfly_price, max_profit])

bflys = pd.DataFrame(data, columns=["strike", "price", "max_profit"])
bflys["prob"] = bflys.price / bflys.max_profit
bflys
```

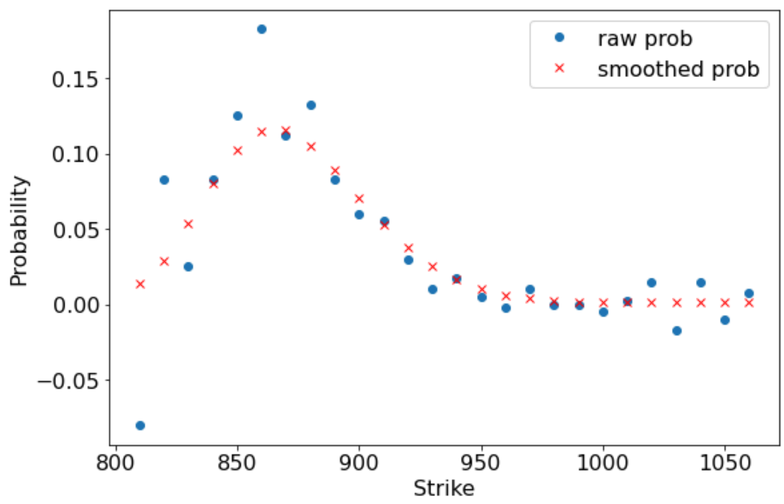
	strike	price	max_profit	prob
0	810	-8.000000e-01	10	-8.000000e-02
1	820	8.250000e-01	10	8.250000e-02
2	830	2.500000e-01	10	2.500000e-02
3	840	8.250000e-01	10	8.250000e-02
4	850	1.250000e+00	10	1.250000e-01
5	860	1.825000e+00	10	1.825000e-01
6	870	1.125000e+00	10	1.125000e-01
7	880	1.325000e+00	10	1.325000e-01
8	890	8.250000e-01	10	8.250000e-02
9	900	6.000000e-01	10	6.000000e-02
10	910	5.500000e-01	10	5.500000e-02
11	920	3.000000e-01	10	3.000000e-02
12	930	1.000000e-01	10	1.000000e-02
13	940	1.750000e-01	10	1.750000e-02
14	950	5.000000e-02	10	5.000000e-03
15	960	-2.500000e-02	10	-2.500000e-03
16	970	1.000000e-01	10	1.000000e-02
17	980	5.551115e-17	10	5.551115e-18
18	990	0.000000e+00	10	0.000000e+00
19	1000	-5.000000e-02	10	-5.000000e-03
20	1010	2.500000e-02	10	2.500000e-03
21	1020	1.500000e-01	10	1.500000e-02
22	1030	-1.750000e-01	10	-1.750000e-02
23	1040	1.500000e-01	10	1.500000e-02
24	1050	-1.000000e-01	10	-1.000000e-02
25	1060	7.500000e-02	10	7.500000e-03

```
# ICICIBANK was trading around 921.75 when this data was collected
plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(9,6))
plt.scatter(bflys.strike, bflys.prob);
plt.xlabel("Strike")
plt.ylabel("Probability")
plt.show()
# plt.savefig("ICICIBANK_raw_bfly_prob.png", dpi=300)
```

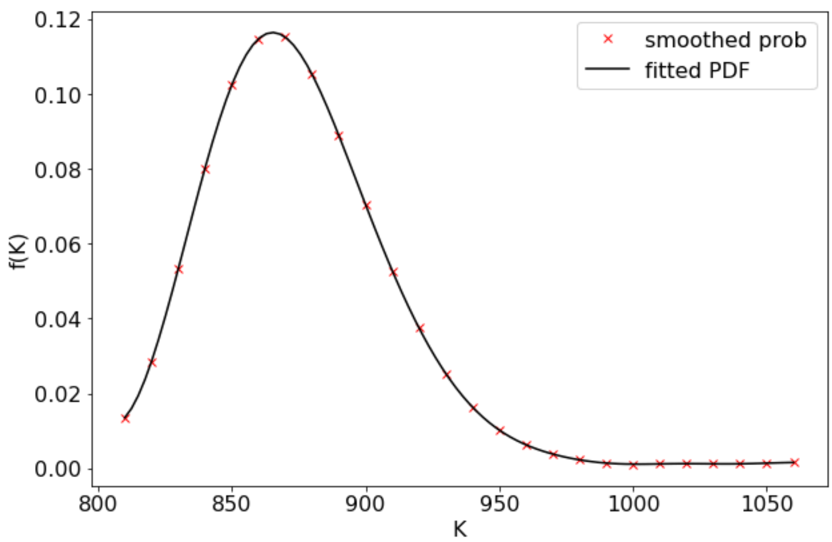



```
from scipy.ndimage import gaussian_filter1d

smoothed_prob = gaussian_filter1d(bflys.prob, 2)

plt.figure(figsize=(9,6))
```

```
plt.plot(bflys.strike, bflys.prob, "o", bflys.strike, smoothed_prob, "rx")
plt.legend(["raw prob", "smoothed prob"], loc="best")
plt.xlabel("Strike")
plt.ylabel("Probability")
plt.show()
# plt.savefig("ICICIBANK_smooth_bfly_prob.png", dpi=300)
```



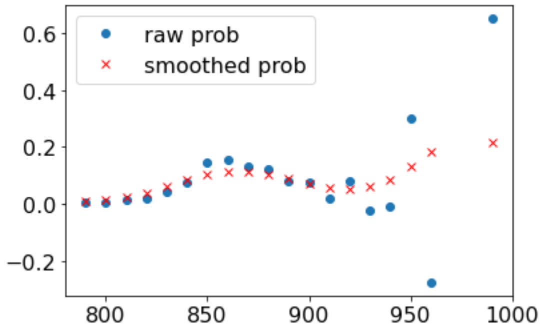

```
plt.figure(figsize=(9,6))
pdf = scipy.interpolate.interp1d(bflys.strike, smoothed_prob, kind="cubic",
                                 fill_value="extrapolate")
x_new = np.linspace(bflys.strike.min(), bflys.strike.max(), 100)
plt.plot(bflys.strike, smoothed_prob, "rx", x_new, pdf(x_new), "k-");
plt.legend(["smoothed prob", "fitted PDF"], loc="best")
plt.xlabel("K")
plt.ylabel("f(K)")
plt.tight_layout()
plt.show()
# plt.savefig("ICICIBANK_bfly_pdf.png", dpi=300)
```



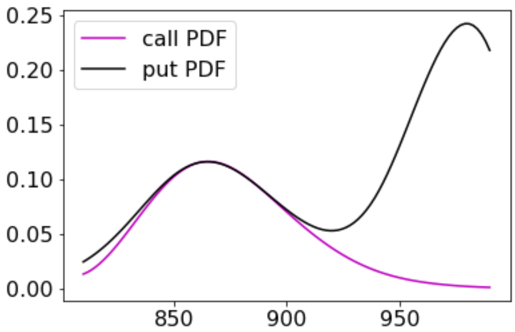

```
# Find area under curve
raw_total_prob = scipy.integrate.trapz(smoothed_prob, bflys.strike)
print(f"Raw total probability: {raw_total_prob}")
normalised_prob = smoothed_prob / raw_total_prob
total_prob = scipy.integrate.trapz(normalised_prob, bflys.strike)
print(f"Normalised total probability: {total_prob}")
# Don't need to normalise because there is mass in the left tail that we are ignoring

# # Normalise
# normalised_prob = smoothed_prob / raw_total_prob
# total_prob = scipy.integrate.trapz(normalised_prob, bflys.strike)
# print(f"Normalised total probability: {total_prob}")
# # should be less than 1
```

```
Raw total probability: 9.299505878658644
Normalised total probability: 1.0000000000000002
```

```
# Repeating the same with put butterflies
from scipy.ndimage import gaussian_filter1d

data = []

for (_, left) ,(_,centre), (_, right) in zip(Put.iterrows(), Put.iloc[1:].iterrows(), Put.iloc[2:].iterrows()):
    # Filter out all zero volume
    if not any(vol > 0 for vol in {left.VOLUME, centre.VOLUME, right.VOLUME}):
        continue
    # Filter out any zero open interest
    if not all(oi > 0 for oi in {left.OI, centre.OI, right.OI}):
        continue
    # Equidistant on either end
    if centre.STRIKE - left.STRIKE != right.STRIKE - centre.STRIKE:
        continue
    butterfly_price = left.midprice - 2* centre.midprice + right.midprice
    max_profit = centre.STRIKE - left.STRIKE
    data.append([centre.STRIKE, butterfly_price, max_profit])

put_bflys = pd.DataFrame(data, columns=["strike", "price", "max_profit"])
put_bflys["prob"] = put_bflys.price / put_bflys.max_profit
smoothed_prob_put = gaussian_filter1d(put_bflys.prob, 2)
plt.plot(put_bflys.strike, put_bflys.prob, "o", put_bflys.strike, smoothed_prob_put, "rx")
plt.legend(["raw prob", "smoothed prob"], loc="best")
plt.show()
```




```
put_pdf = scipy.interpolate.interp1d(put_bflys.strike, smoothed_prob_put, kind="cubic",
                                     fill_value=0.0)
x_new = np.linspace(bflys.strike.min(), put_bflys.strike.max(), 100)
plt.plot(x_new, pdf(x_new), "m-", x_new, put_pdf(x_new), "k-");
plt.legend(["call PDF", "put PDF"], loc="best")
plt.show()
```



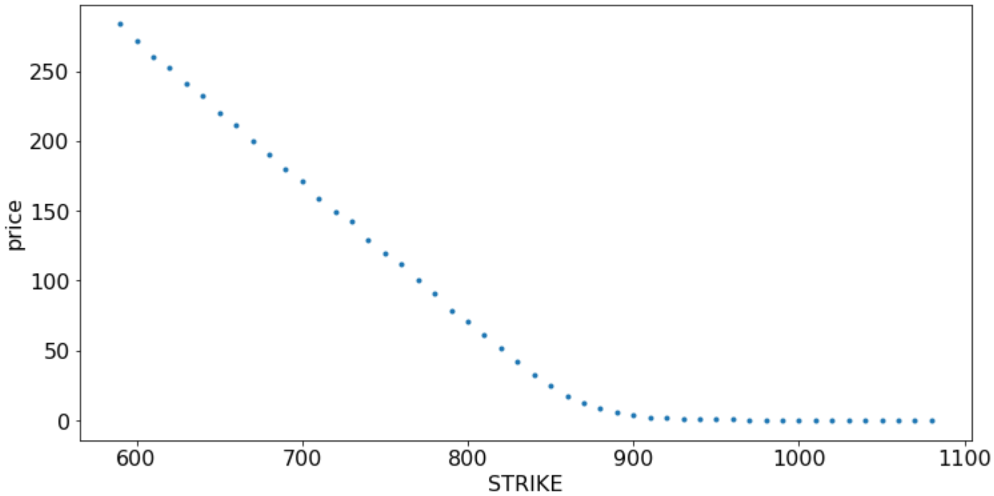

```
def construct_pdf(calls_df, make_plot=True, fill_value="extrapolate"):
    if "midprice" not in calls_df.columns:
        calls_df["midprice"] = (calls_df.bid + calls_df.ask) /2

    # Construct butterflies
    data = []

    for (_, left) ,(_,centre), (_, right) in zip(calls_df.iterrows(), calls_df.iloc[1:].iterrows(), calls_df.iloc[2:].iterrows()):
        # Filter out all zero volume
        if not any(vol > 0 for vol in {left.VOLUME, centre.VOLUME, right.VOLUME}):
            continue
        # Filter out any zero open interest
        if not all(oi > 0 for oi in {left.OI, centre.OI, right.OI}):
            continue
        # Equidistant on either end
        if centre.STRIKE - left.STRIKE != right.STRIKE - centre.STRIKE:
            continue
        butterfly_price = left.midprice - 2* centre.midprice + right.midprice
        max_profit = centre.STRIKE - left.STRIKE
```

```
            data.append([centre.STRIKE, butterfly_price, max_profit])

        bflys = pd.DataFrame(data, columns=["strike", "price", "max_profit"])
        bflys["prob"] = bflys.price / bflys.max_profit

        smoothed_prob = gaussian_filter1d(bflys.prob, 2)
        pdf = scipy.interpolate.interp1d(bflys.strike, smoothed_prob, kind="cubic",
                                         fill_value=fill_value)
        if not make_plot:
            return pdf

        plt.figure(figsize=(9,6))
        x_new = np.linspace(bflys.strike.min(), bflys.strike.max(), 100)
        plt.plot(bflys.strike, smoothed_prob, "rx", x_new, pdf(x_new), "k-");
        plt.legend(["smoothed prob", "fitted PDF"], loc="best")
        plt.xlabel("K")
        plt.ylabel("f(K)")


    #Implied PDF from Breeden-Litzenberger
    Call["midprice"] = (Call.BID + Call.ASK)/2
    Call = Call[Call.midprice > 0]
    Call
```

	STRIKE	LTP	CHNG	BID	ASK	VOLUME	OI	midprice
0	590	-	-	248.40	318.65	NaN	NaN	283.525
1	600	-	-	239.40	303.95	NaN	NaN	271.675
2	610	-	-	245.40	275.50	NaN	1.0	260.450
3	620	-	-	224.15	280.95	NaN	NaN	252.550
4	630	-	-	212.25	270.45	NaN	NaN	241.350
5	640	-	-	203.05	262.25	NaN	NaN	232.650
6	650	-	-	194.55	245.55	NaN	NaN	220.050
7	660	-	-	198.70	224.55	NaN	NaN	211.625
8	670	-	-	186.25	213.90	NaN	NaN	200.075
9	680	-	-	176.90	203.25	NaN	NaN	190.075
10	690	-	-	167.35	192.80	NaN	NaN	180.075
11	700	169.45	-20.55	167.95	173.60	1.0	127.0	170.775
12	710	-	-	147.35	170.70	NaN	NaN	159.025
13	720	-	-	137.40	160.50	NaN	NaN	148.950
14	730	-	-	134.55	149.90	NaN	10.0	142.225
15	740	-	-	117.35	141.15	NaN	NaN	129.250
16	750	-	-	110.05	129.90	NaN	3.0	119.975
17	760	-	-	102.50	120.60	NaN	2.0	111.550
18	770	-	-	90.60	110.00	NaN	NaN	100.300
19	780	-	-	82.45	99.60	NaN	6.0	91.025
20	790	-	-	67.40	89.80	NaN	NaN	78.600
21	800	69	1.3	70.15	71.30	124.0	200.0	70.725
22	810	-	-	58.95	63.80	NaN	18.0	61.375
23	820	52.5	3.25	50.65	51.80	20.0	37.0	51.225
24	830	40.5	2.9	41.55	42.25	6.0	86.0	41.900
25	840	31.6	1.8	32.50	33.15	43.0	274.0	32.825
26	850	24.6	1.85	24.45	24.70	1352.0	560.0	24.575
27	860	17.55	1.35	17.45	17.70	3012.0	1133.0	17.575
28	870	12.45	1.25	12.35	12.45	9861.0	5026.0	12.400
29	880	8.4	0.9	8.30	8.40	8534.0	4950.0	8.350
30	890	5.65	0.45	5.60	5.65	4680.0	4219.0	5.625
31	900	3.75	0.1	3.70	3.75	7286.0	9775.0	3.725
32	910	2.45	0.05	2.40	2.45	4831.0	5372.0	2.425
33	920	1.7	0.1	1.65	1.70	2889.0	4828.0	1.675
34	930	1.2	0.1	1.20	1.25	1863.0	2048.0	1.225
35	940	0.85	-	0.85	0.90	734.0	1191.0	0.875
36	950	0.75	0.1	0.65	0.75	549.0	1588.0	0.700
37	960	0.55	0.05	0.55	0.60	406.0	831.0	0.575
38	970	0.45	0.05	0.40	0.45	164.0	477.0	0.425
39	980	0.35	0.05	0.35	0.40	61.0	341.0	0.375
40	990	0.3	0.05	0.25	0.40	6.0	281.0	0.325
41	1000	0.3	0.05	0.25	0.30	170.0	2347.0	0.275
42	1010	0.2	0.05	0.15	0.20	13.0	114.0	0.175
43	1020	0.15	0.05	0.05	0.15	8.0	152.0	0.100
44	1030	0.15	-	0.10	0.25	10.0	39.0	0.175
45	1040	0.05	-0.05	0.05	0.10	2.0	32.0	0.075
46	1050	0.15	-	0.10	0.15	7.0	205.0	0.125
47	1060	-	-	0.00	0.15	NaN	6.0	0.075
48	1070	-	-	0.00	0.20	NaN	1.0	0.100

49	1080	-	-	0.05	0.20	NaN	1.0	0.125
50	1090	0.25	-	0.05	0.25	1.0	NaN	0.150

```
Call_sub = Call[(Call.STRIKE > 580) & (Call.STRIKE < 1090)]
plt.figure(figsize=(12,6))
plt.plot(Call_sub.STRIKE, Call_sub.midprice, ".");
plt.xlabel("STRIKE")
plt.ylabel("price")
plt.savefig("call_prices.png", dpi=400)
plt.show()
```




```
def call_value(S, K, sigma, t=0, r=0):
    # use np.multiply and divide to handle divide-by-zero
    with np.errstate(divide='ignore'):
        d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
        d2 = d1 - sigma * np.sqrt(t)
    return np.multiply(norm.cdf(d1),S) - np.multiply(norm.cdf(d2), K * np.exp(-r * t))

def call_vega(S, K, sigma, t=0, r=0):
    with np.errstate(divide='ignore'):
        d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
    return np.multiply(S, norm.pdf(d1)) * np.sqrt(t)

def bs_iv(price, S, K, t=0, r=0, precision=1e-4, initial_guess=0.2, max_iter=1000, verbose=False):
    iv = initial_guess
    for _ in range(max_iter):
        P = call_value(S, K, iv, t, r)
        diff = price - P
        if abs(diff) < precision:
            return iv
        grad = call_vega(S, K, iv, t, r)
        iv += diff/grad
    if verbose:
        print(f"Did not converge after {max_iter} iterations")
    return iv
```

```
c_test = call_value(871.65, 860, 0.2, t=1/52)
print(c_test)
```

```
16.51045069073359
```

```
bs_iv(c_test, 871.65,860, t=1/52)
```

```
0.2
```

```
S = 871.65
t = 1/52
Call["iv"] = Call.apply(lambda row: bs_iv(row.midprice, S, row.STRIKE, t, max_iter=500), axis=1)
```

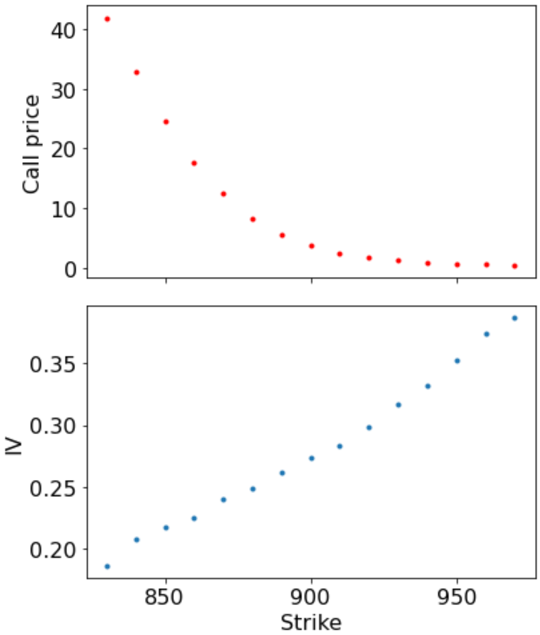
```
<ipython-input-43-08b1dc83cea8>:21: RuntimeWarning: divide by zero encountered in double_scalars
  iv += diff/grad
<ipython-input-43-08b1dc83cea8>:4: RuntimeWarning: invalid value encountered in double_scalars
  d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
<ipython-input-43-08b1dc83cea8>:10: RuntimeWarning: invalid value encountered in double_scalars
  d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
<ipython-input-43-08b1dc83cea8>:4: RuntimeWarning: overflow encountered in double_scalars
  d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
```

```
<ipython-input-43-08b1dc83cea8>:10: RuntimeWarning: overflow encountered in double_scalars
  d1 = np.divide(1, sigma * np.sqrt(t)) * (np.log(S/K) + (r+sigma**2 / 2) * t)
```

```
def plot_vol_smile(Call, savefig=False):
    plt.figure(figsize=(9,6))
    plt.plot(Call.STRIKE, Call.iv, ".")
    plt.xlabel("Strike")
    plt.ylabel("IV")
    if savefig:
        plt.savefig("vol_smile.png",dpi=300)
    plt.show()
```

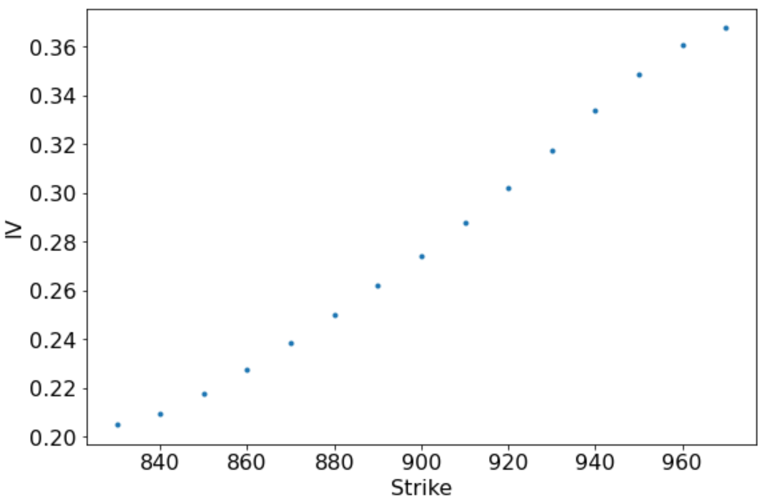
```
Call_no_na = Call.dropna()
```

```
fig, (ax1, ax2) = plt.subplots(2,1, figsize=(6,7), sharex=True)
ax1.plot(Call_no_na.STRIKE, Call_no_na.midprice, "r.")
ax1.set_ylabel("Call price")
ax2.plot(Call_no_na.STRIKE, Call_no_na.iv, ".")
ax2.set_ylabel("IV")
ax2.set_xlabel("Strike")
plt.tight_layout()
# plt.savefig("calls_to_iv.png", dpi=400)
plt.show()
```




```
Call_clean = Call.dropna().copy()
Call_clean["iv"] = gaussian_filter1d(Call_clean.iv, 2)
```

```
plot_vol_smile(Call_clean)
```




```
Call_clean = Call_clean[(Call_clean.STRIKE > 790) & (Call_clean.STRIKE < 1000)]
```