INVESTX-FAC

FINANCE AND ANALYTICS CLUB

Mentors: Shivansh Gupta | Aayush Gupta | Hamza

Technical Indicators

Technical indicators are heuristic or patterns based signals that are produced by the price , volatility, volume Used by traders who use technical indicators to predict future market movement

Traders and investors use these indicators to identify trends, generate buy or sell signals, and make informed decisions about entering or exiting trades.

We have learned about 15 technical indicators.
15 indicators were categorized into 3
Subgroups as volume,momentum,volatility.
And everyone was assigned one of the indicator in each group to code it in for a particular company .

- ## Accumulation and Distribution Line(A/D LIne)

The Accumulation/Distribution Line (A/D Line) is a technical indicator that combines price and volume data to assess the flow of money into or out of a security. It is used to determine whether accumulation or distribution is occurring in a particular asset.

The A/D Line is calculated based on the following formula:

- A/D Line = Previous A/D Line + Current Money Flow

The Current Money Flow is calculated by multiplying the volume of the period by the Money Flow Multiplier, which is determined by the relationship between the closing price and the range (high and low) of the period:

- Money Flow Multiplier = [(Close - Low) - (High - Close)] / (High - Low)

The A/D Line is typically plotted as a line on a price chart. If the A/D Line is rising, it suggests that accumulation is taking place, indicating buying

pressure and potentially signaling a bullish trend. Conversely, if the A/D Line is falling, it indicates distribution, suggesting selling pressure and potentially signaling a bearish trend.

```
################## CODE FOR A/D LINE
########################################

import pandas as pd
import numpy as np
import yfinance as yf
from ta import trend
import matplotlib.pyplot as plt
import mplcursors
data = yf.download("^NSEBANK", start="2023-01-01", end="2023-05-01")
df = data.reset_index()
#MF = [(Close - Low) - (High - Close)] / (High - Low)
#MFV = MF * Volume
ls=[]
Mfv=[]
Adl=0
adl=[]
for i in range(79):
    close=df['Close'][i]
    low=df['Low'][i]
    high=df['High'][i]
    k=((close-low)-(high-close))/(high-low)
    p=k*df['Volume'][i]
    adl.append(Adl)
    Adl+=p

    Mfv.append(p)
    ls.append(k)

df['MF']=ls
df['MFV']=Mfv
df['ADL']=adl
df.plot(x='Date',y='ADL')
plt.show()
```

- Implied volatility

Implied volatility is an indicator used in options trading to gauge the market's expectation of future price volatility of the underlying asset. It represents the market's perception of the potential magnitude of price swings and is a crucial component in pricing options.

```
##############CODE FOR IMPLIED
VOLATILITY#################################
import pandas as pd
import numpy as np
import yfinance as yf
from ta import trend
import matplotlib.pyplot as plt
import mplcursors
import math

data = yf.download("^NSEBANK", start="2023-01-01", end="2023-05-01")
df = data.reset_index()
lst=[0]

for i in range(79):
    if(i!=0):
        a=math.log(df['Close'][i]/df['Close'][i-1])
        lst.append(a)
df['LOG_R']=lst
iv=df['LOG_R'].tolist()
d=np.array(iv)
std=np.std(d)
print(df)
print("the Implied volatility is",std)
```

- Average Directional Index

The Average Directional Index (ADX) is a technical indicator used in financial markets to assess the strength and direction of a trend. It was developed by J. Welles Wilder and is commonly used by traders and analysts to make informed decisions about buying or selling assets.

The ADX is a non-directional indicator, meaning it does not provide information about the direction of the trend (whether it is upward or

downward), but rather focuses on the strength or intensity of the trend. It is represented as a single line with values ranging from 0 to 100.

A high ADX value suggests a strong trend, while a low value indicates a weak or absent trend. Traders often use a threshold, such as 25 or 30, to determine whether a trend is strong enough to be considered significant. Additionally, the ADX can be used in conjunction with other technical indicators to confirm the presence of a trend and to identify potential entry and exit points for trading positions.

Formulas Used:

$$+\text{DI} = \left(\frac{\text{Smoothed } +\text{DM}}{\text{ATR}} \right) \times 100$$

$$\text{-DI} = \left(\frac{\text{Smoothed -DM}}{\text{ATR}} \right) \times 100$$

$$\text{DX} = \left(\frac{|\ +\text{DI} - \text{-DI}\ |}{|\ +\text{DI} + \text{-DI}\ |} \right) \times 100$$

$$\text{ADX} = \frac{(\text{Prior ADX} \times 13) + \text{Current ADX}}{14}$$

where:

$$+\text{DM (Directional Movement)} = \text{Current High} - \text{PH}$$

$$\text{PH} = \text{Previous High}$$

$$\text{-DM} = \text{Previous Low} - \text{Current Low}$$

$$\text{Smoothed } +/\text{-DM} = \sum_{t=1}^{14} \text{DM} - \left(\frac{\sum_{t=1}^{14} \text{DM}}{14} \right) + \text{CDM}$$

$$\text{CDM} = \text{Current DM}$$

$$\text{ATR} = \text{Average True Range}$$

```
############## Here I have calculated the Average Directional Index
########################


import numpy as np
import pandas as pd
import yfinance as yf
import math

data = yf.download("AAPL", start="2017-01-01", end="2017-04-30")
data['tr1']=data['High']-data['Low']
data['tr2']=data['High']-data['Close'].shift(1)
data['tr3']=data['Low']-data['Close'].shift(1)
data['Positive DM'] = data['High']
data['Negative DM'] = data['Low']

data['True Range'] = data[['tr1', 'tr2', 'tr3']].apply(lambda x: max(x), axis=1)

######################################### Positive DM
###################################

for i in range(81):
  if(data.iloc[i,1]-data.iloc[i-1,1]>data.iloc[i-1,2]-data.iloc[i,2]):
    data.iloc[i,9] =  data.iloc[i,1]-data.iloc[i-1,1]
  else:
      data.iloc[i,9] = 0
######################################### Negative DM
###################################
for i in range(81):
  if(data.iloc[i-1,2]-data.iloc[i,2]>data.iloc[i,1]-data.iloc[i-1,1]):
    data.iloc[i,10] =  data.iloc[i-1,2]-data.iloc[i,2]
  else:
      data.iloc[i,10] = 0

period=17
data['ATR']=data['True Range'].rolling(period).mean()

############################## To get +DI and -DI
######################

data['Plus DI'] = 100 * (data['Positive DM'].ewm(alpha = 1/(period)).mean() /data['ATR'])
data['Minus DI'] = 100 * (data['Negative DM'].ewm(alpha = 1/(period)).mean() /data['ATR'])

############################## To get DI
##############################

data['DI'] = (abs(data['Plus DI'] - data['Minus DI']) / abs(data['Plus DI'] + data['Minus DI'])) * 100

############################ To get ADX and smoothed ADX
############################
```

```
data['ADX'] = ((data['DI'].shift(1) * (period - 1)) + data['DI']) / (period)
data['ADX Smooth'] = data['ADX'].ewm(alpha = 1/(period)).mean()

data

data['ADX Smooth']

data
```

- ## Stochastic Oscillator

A stochastic oscillator is a momentum indicator comparing a particular closing price of a security to a range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a moving average of the result.

TRADE

TABLE OF CONTENTS
TECHNICAL ANALYSIS TECHNICAL ANALYSIS BASIC EDUCATION
Stochastic Oscillator: What It Is, How It Works, How To Calculate
By ADAM HAYES Updated June 25, 2021
Reviewed by CHARLES POTTERS
Fact checked by AMANDA BELLUCCO-CHATHAM
Stochastic Oscillator
Investopedia / Jessica Olah

What Is a Stochastic Oscillator?
A stochastic oscillator is a momentum indicator comparing a particular closing price of a security to a range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a moving average of the result. It is used to generate overbought and oversold trading signals, utilizing a 0–100 bounded range of values.

A stochastic oscillator is a popular technical indicator for generating overbought and oversold signals.

It is a popular momentum indicator, first developed in the 1950s.

Stochastic oscillators tend to vary around some mean price level since they rely on an asset's price history.

Stochastic oscillators measure the momentum of an asset's price to determine trends and predict reversals.

Stochastic oscillators measure recent prices on a scale of 0 to 100, with measurements above 80 indicating that an asset is overbought and measurements below 20 indicating that it is oversold.

Stochastic oscillator charting generally consists of two lines: one reflecting the actual value of the oscillator for each session, and one reflecting its three-day simple moving average. Because price is thought to follow momentum, the intersection of these two lines is considered to be a signal that a reversal may be in the works, as it indicates a large shift in momentum from day to day.

Divergence between the stochastic oscillator and trending price action is also seen as an important reversal signal. For example, when a bearish trend reaches a new lower low, but the oscillator prints a higher low, it may be an indicator that bears are exhausting their momentum and a bullish reversal is brewing.

Formula used

\\
%K=((C−L14)/(H14−L14C))×100
where:

C = The most recent closing price

L14 = The lowest price traded of the 14 previous trading sessions

H14 = The highest price traded during the same 14-day period

%K = The current value of the stochastic indicator.))).
////////////////////////////////////

##############################
Hear I have calculated stochastic indicator
##############################

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt


ticker = "MSFT"
start_date = "2022-05-23"
end_date = "2023-05-23"


data = yf.download(ticker, start=start_date, end=end_date)


def calculate_stochastic_oscillator(data, window=14):
```

```
    data["Lowest Low"] = data["Low"].rolling(window=window).min()
    data["Highest High"] = data["High"].rolling(window=window).max()
    data["%K"] = (data["Close"] - data["Lowest Low"]) / (data["Highest High"] - data["Lowest
Low"]) * 100
    data["%D"] = data["%K"].rolling(window=3).mean()
    return data


data1 = calculate_stochastic_oscillator(data)
data1["%K"] = data1["%K"].fillna(0)  # jaha data nhi h 0 bhar do
data1["%D"] = data1["%D"].fillna(0)

print()
plt.figure(figsize=(12, 6))
plt.plot(data.index, data["%K"], label="%K")
plt.plot(data.index, data["%D"], label="%D")
plt.title("Stochastic Oscillator")
plt.xlabel("Date")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.show()
```

- ## Ichimoku Cloud Lines

The Ichimoku Cloud lines is a collection of technical indicators that show support and resistance levels, as well as momentum and trend direction. It does this by taking multiple averages and plotting them on a chart. It also uses these figures to compute a "cloud" that attempts to forecast where the price may find support or resistance in the future.

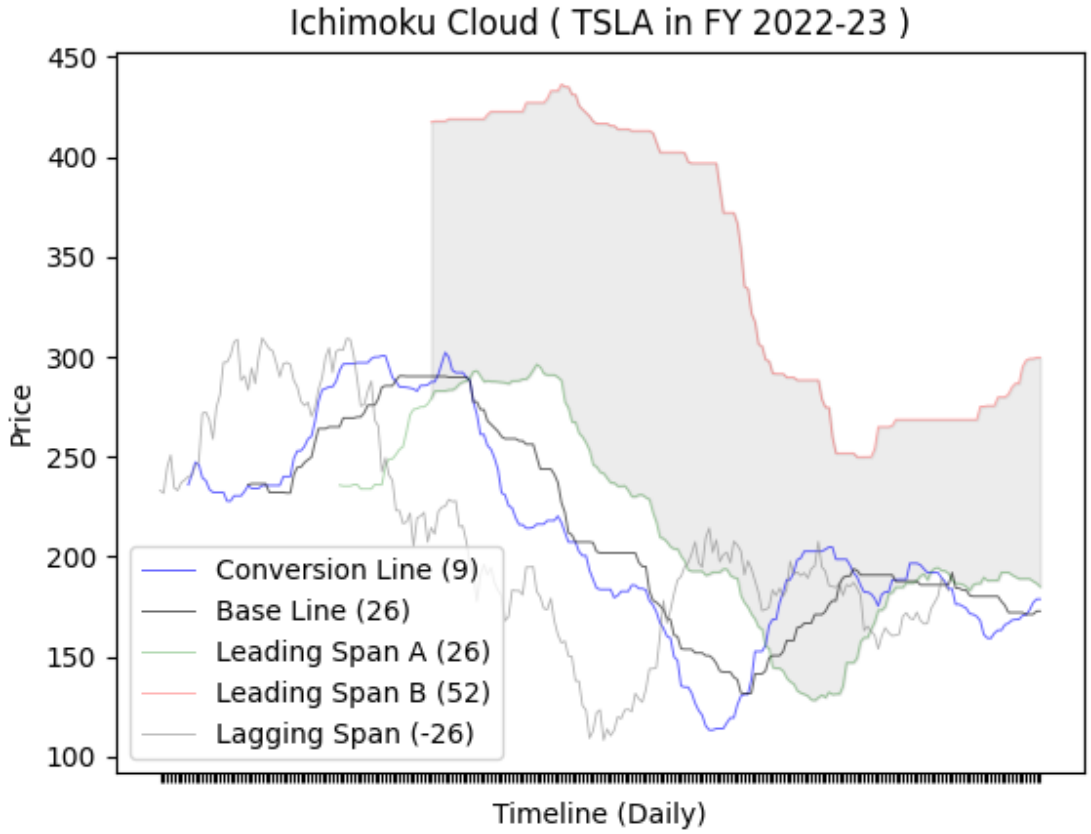
Formulas Used:

1. *Tenkan-sen (Conversion Line): (9-period high + 9-period low) / 2*
2. *Kijun-sen (Base Line): (26-period high + 26-period low) / 2*
3. *Senkou Span A (Leading Span A): (Conversion Line + Base Line) / 2, plotted 26 periods ahead*
4. *Senkou Span B (Leading Span B): (52-period high + 52-period low) / 2, plotted 26 periods ahead*
5. *Chikou Span (Lagging Span): Current closing price, plotted 26 periods behind*

```
####### Here is the code of Ichimoku Cloud Lines ###########
# Ichimoku Cloud Lines
```

```
import pandas as pd
import matplotlib.pyplot as plt

def calculate_tenkan_sen(high, low):
    # Conversion line
    return (high.rolling(window=9).max() + low.rolling(window=9).min()) / 2

def calculate_kijun_sen(high, low):
    # Base line
    return (high.rolling(window=26).max() + low.rolling(window=26).min()) / 2

def calculate_senkou_span_a(tenkan_sen, kijun_sen):
    # Leading Span A
    return ((tenkan_sen + kijun_sen) / 2).shift(26)

def calculate_senkou_span_b(high, low):
    # Leading Span B
    return ((high.rolling(window=52).max() + low.rolling(window=52).min()) / 2)

def calculate_chikou_span(close):
    # Lagging Span
    return close.shift(-26)

# Usage
data = pd.read_csv('TSLA.csv')
high = data['High']
low = data['Low']
close = data['Close']
date = data['Date']

conv_line = calculate_tenkan_sen(high, low)
base_line = calculate_kijun_sen(high, low)
leadS_a = calculate_senkou_span_a(conv_line, base_line)
leadS_b = calculate_senkou_span_b(high, low)
lagS = calculate_chikou_span(close)

plt.plot(date, conv_line, label='Conversion Line (9)', color='blue', alpha=0.6, linewidth=0.7)
plt.plot(date, base_line, label='Base Line (26)', color='black', alpha=0.6, linewidth=0.7)
plt.plot(date, leadS_a, label='Leading Span A (26)', color='green', alpha=0.3, linewidth=0.7)
plt.plot(date, leadS_b, label='Leading Span B (52)', color='red', alpha=0.3, linewidth=0.7)
plt.fill_between(date, leadS_a, leadS_b, color='grey', alpha=0.15)
plt.plot(date, lagS, label='Lagging Span (-26)', color='grey', alpha=0.6, linewidth=0.6)

plt.title('Ichimoku Cloud ( TSLA in FY 2022-23 )')
plt.ylabel('Price')
plt.xlabel('Timeline (Daily)')
plt.legend(loc='best')

ax = plt.gca()
ax.axes.xaxis.set_ticklabels([])
```

plt.show()



- Volume Price Indicator

The volume price trend indicator is used to determine the balance between a security's demand and supply. The percentage change in the share price trend shows the relative supply or demand of a particular security, while volume indicates the force behind the trend.

Possible trading signals provided by the VPT indicator include the following:
• Upward price trend is confirmed by an increasing VPT and increasing price.

• Downward price trend is confirmed by a decreasing VPT and decreasing price.
• Upward price momentum is interpreted as waning and bearish divergence is indicated when unchanged or falling VPT values occur with rising prices.

Formulas Used:

Daily Returns = ((Close-Open)/Open)*100
VPT = Cumulative Sum of (Volume*Daily Returns)

####### Here I have calculated the Volume Price Indicator ###########

```
import numpy as np
import pandas as pd
import yfinance as yf
import math
import matplotlib.pyplot as plt

data = yf.download("AAPL", start="2017-01-01", end="2017-04-30")

data['Price Change'] = data['Close'].pct_change()
data['Price Change'].fillna(0, inplace=True)
# Calculate Volume Price Trend
data['VPT'] = (data['Volume'] * data['Price Change']).cumsum()
```

Data

- # Donchian Channel

Donchian Channels are three lines generated by moving average calculations that comprise an indicator formed by upper and lower bands around a midrange or median band. The upper band marks the highest price of a security over N periods while the lower band marks the lowest price of a security over N periods. The area between the upper and lower bands represents the Donchian Channel.

Formulas Used:

UC = Highest High in Last *N* Periods
Middle Channel=((*UC*+*LC*)/2)
LC = Lowest Low in Last *N* periods
where:
UC=Upper channel

N=Number of minutes, hours, days,
weeks, months,Minutes, hours, days, weeks
LC=Lower channel
########################## Here I have calculated the Donchian Channels ################

```
import numpy as np
import pandas as pd
import yfinance as yf
import math
import matplotlib.pyplot as plt

data = yf.download("AAPL", start="2017-01-01", end="2017-04-30")

period=24
data['Upper Donchian'] = data['High'].rolling(period).max()
data['Lower Donchian'] = data['Low'].rolling(period).min()
data['Middle Donchian'] = (data['Upper Donchian'] + data['Lower Donchian']) / 2

data

plt.plot(data.index, data['Close'], color='black', label='Close')
plt.title('Price and Donchian Channels vs Time')
plt.plot(data.index, data['Upper Donchian'], color='red', label='Upper Donchian')
plt.plot(data.index, data['Middle Donchian'], color='green', label='Middle Donchian')
plt.plot(data.index, data['Lower Donchian'], color='blue', label='Lower Donchian')
plt.xlabel=('Date')
plt.ylabel('Price')
plt.show()
```

- ## Ease Of Movement Indicator

The Ease of Movement (EOM) indicator is a technical analysis tool used to assess the relationship between price and volume in financial markets. It provides insights into the ease or difficulty with which prices are moving based on the volume of trades.

The EOM indicator is calculated by taking into account both the price change and the volume traded during a specific period. It is represented as a line chart that fluctuates above and below a zero line.

The EOM indicator helps traders and analysts identify the strength of price movements. When the EOM line moves above the zero line, it suggests that prices are rising with relatively low volume, indicating ease of

movement. Conversely, when the EOM line falls below the zero line, it indicates that prices are declining with relatively low volume, implying ease of downward movement.

Traders often look for divergences between the EOM indicator and the price chart to identify potential reversal or continuation signals. For example, if prices are rising while the EOM line is falling, it may suggest that the upward movement is losing momentum and a potential reversal could occur.

The EOM indicator is commonly used in conjunction with other technical analysis tools to make more informed trading decisions. It can help traders assess the underlying strength or weakness of price movements and potentially anticipate future price trends.

Formulas Used:

$$\text{Distance Moved} = \left(\frac{\text{High} + \text{Low}}{2} - \frac{\text{PH} + \text{PL}}{2} \right)$$

$$\text{Box Ratio} = \frac{\left(\frac{\text{Volume}}{\text{Scale}} \right)}{\text{High} - \text{Low}}$$

$$\text{1-Period EMV} = \frac{\left(\frac{\text{High} + \text{Low}}{2} - \frac{\text{PH} + \text{PL}}{2} \right)}{\left(\frac{\frac{\text{Volume}}{\text{Scale}}}{\text{High} - \text{Low}} \right)}$$

$$\text{14-Period Ease of Movement} = \text{14-Period Simple Moving Average of 1-Period EMV}$$

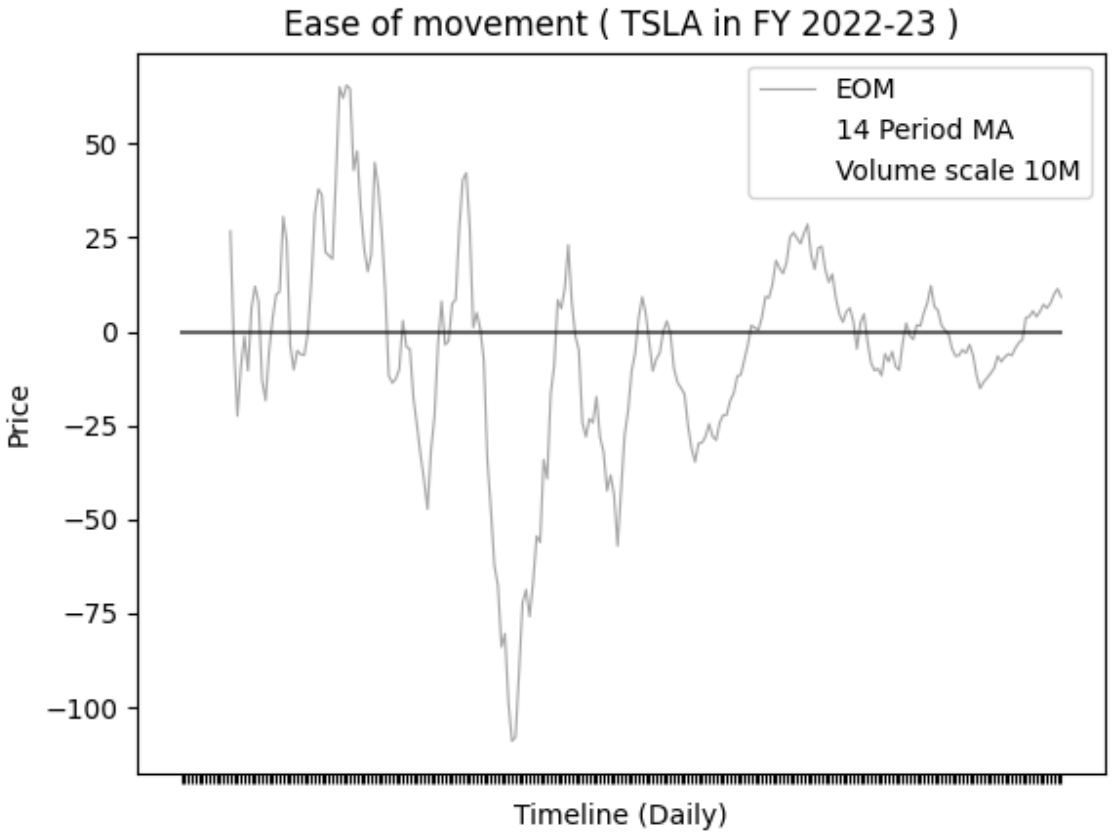
where :

$$\text{PH} = \text{Prior High}$$

$$\text{PL} = \text{Prior Low}$$

########################## Here is the code of Ease of Movement Indicator ################

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def calculate_eom(data):

    high = data['High']
    low = data['Low']
    vol = data['Volume']
    dm = ((high + low) / 2) - ((high.shift(1) + low.shift(1)) / 2)
    br = (vol / 100000000) / (high - low)

    eom = dm / br
    eom_ma = eom.rolling(14).mean()

    return eom_ma


data = pd.read_csv('TSLA.csv')
date = data['Date']

eom_i = calculate_eom(data)

plt.plot(date, eom_i, label='EOM', color='grey', alpha=0.7, linewidth=0.8)
zeros = np.zeros(len(data))
plt.plot(date, zeros, color='black', alpha=0.6)

plt.title('Ease of movement ( TSLA in FY 2022-23 )')
plt.ylabel('Price')
plt.xlabel('Timeline (Daily)')
plt.plot([], [], ' ', label='14 Period MA')
plt.plot([], [], ' ', label='Volume scale 10M')
plt.legend(loc='best')

ax = plt.gca()
ax.axes.xaxis.set_ticklabels([])

plt.show()
```



- Keltner Channels

The Keltner Channels indicator is a technical analysis tool used by traders to identify potential price breakouts, overbought or oversold conditions, and to determine the overall volatility of a financial instrument. It was developed by Chester W. Keltner and is based on the concept of moving average envelopes.

The Keltner Channels consist of three lines plotted on a price chart: a middle line, an upper channel line, and a lower channel line. The middle line is typically a moving average of the price, such as a simple moving average or an exponential moving average. The upper and lower channel

lines are calculated by adding and subtracting a multiple of the Average True Range (ATR) from the middle line.

The ATR is a measure of volatility and represents the average price range over a specified period. By using the ATR as a basis for the channel lines, the Keltner Channels adjust dynamically to changes in market volatility.

Traders interpret the Keltner Channels in several ways. When the price moves above the upper channel line, it is considered a potential bullish signal, indicating that the price may be overextended. Conversely, when the price drops below the lower channel line, it is seen as a potential bearish signal, suggesting that the price may be oversold.

Additionally, traders observe the width of the channel to gauge market volatility. A wider channel suggests higher volatility, while a narrower channel indicates lower volatility.

The Keltner Channels are often used in conjunction with other technical indicators to confirm trading signals or identify potential reversals. Traders may also use the Keltner Channels to set stop-loss orders or profit targets based on the channel boundaries.

Formulas Used:

$$\text{Keltner Channel Middle Line} = EMA$$

$$\text{Keltner Channel Upper Band} = EMA + 2 * ATR$$

$$\text{Keltner Channel Lower Band} = EMA - 2 * ATR$$

where:

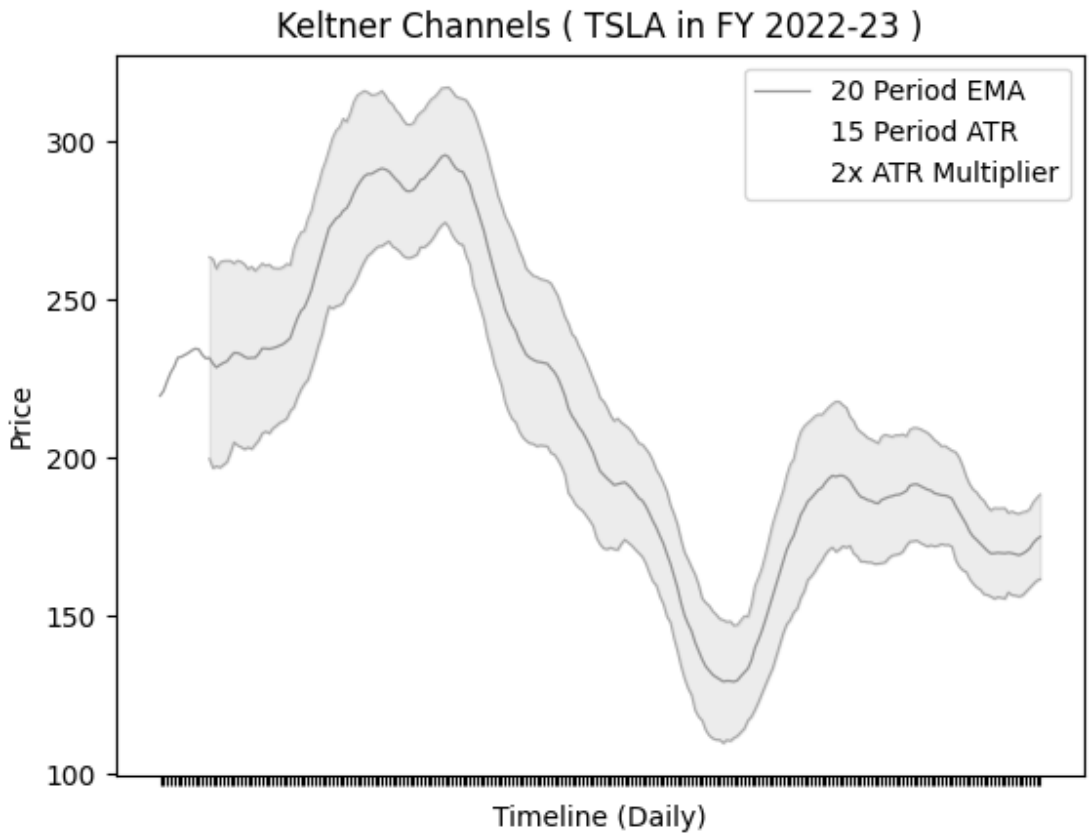
$EMA = \text{Exponential moving average (typically over 20 perio}$

$ATR = \text{Average True Range (typically over 10 or 20 periods)}$

########################## Here is the code of Keltner Channels#################
import pandas as pd
import yfinance as yf

import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt

```
def calculate_ema(dataframe):
    ema = dataframe['Close'].ewm(span=20, adjust=False).mean()
    return ema

def calculate_atr(data):
    # Calculate the True Range (TR)
    data['HL'] = data['High'] - data['Low']
    data['HC'] = abs(data['High'] - data['Close'].shift(1))
    data['LC'] = abs(data['Low'] - data['Close'].shift(1))
    data['TR'] = data[['HL', 'HC', 'LC']].max(axis=1)

    # Calculate the Average True Range (ATR)
    # I choose 15 period
    data['ATR'] = data['TR'].rolling(15).mean()

    return data['ATR']


data = pd.read_csv('TSLA.csv')

date = data['Date']
ema = calculate_ema(data)
atr = calculate_atr(data)
# print(ema)
plt.plot(date, ema, color='grey', linewidth=0.9, alpha=0.8, label='20 Period EMA')
plt.plot(date, ema-(2*atr), color='grey', linewidth=0.8, alpha=0.6)
plt.plot(date, ema+(2*atr), color='grey', linewidth=0.8, alpha=0.6)
plt.fill_between(date, ema-(2*atr), ema+(2*atr), color='grey', alpha=0.15)

plt.title('Keltner Channels ( TSLA in FY 2022-23 )')
plt.ylabel('Price')
plt.xlabel('Timeline (Daily)')
plt.plot([], [], ' ', label='15 Period ATR')
plt.plot([], [], ' ', label='2x ATR Multiplier')
plt.legend(loc='best')

ax = plt.gca()
ax.axes.xaxis.set_ticklabels([])

plt.show()
```



LSTM

LSTM stands for Long Short-Term Memory, which is a type of recurrent neural network (RNN) architecture. LSTM networks are designed to handle and process sequential data, such as time series data or natural language sequences, by retaining and utilizing information from previous time steps.

Reasons why LSTM models are favored for stock market prediction:

1) Handling Long-Term Dependencies: LSTM models are specifically designed to overcome the limitations of traditional recurrent neural networks (RNNs) in capturing long-term dependencies. They have a

memory cell and various gates that allow them to retain information over long periods. In stock market prediction, where trends and patterns may take time to develop, LSTM models can effectively capture these dependencies.

2) Capturing Temporal Patterns: Stock market data often exhibits complex temporal patterns, such as trends, seasonality, and cyclical behavior. LSTM models can learn and model these patterns by analyzing the historical data. The memory cells in LSTM models enable them to retain and use information from past time steps, making them suitable for capturing the dynamics of stock market data.

3) Dealing with Noisy Data: Stock market data is often noisy and contains irregularities due to various factors like market sentiment, news events, and economic indicators. LSTM models are robust to noise and can handle missing or incomplete data. They can learn to smooth out the noise and identify meaningful patterns in the data.

4) Handling Multiple Inputs: Stock market prediction usually involves analyzing multiple factors and indicators, such as historical prices, trading volumes, and technical indicators. LSTM models can easily handle multiple input features and learn their relationships to make predictions. They can effectively extract relevant features and identify complex interactions among them.

5) Adaptability to Variable Time Horizons: LSTM models are flexible in predicting stock market movements over different time horizons. By adjusting the model's architecture and training process, they can be trained to make short-term or long-term predictions, depending on the specific requirements of the task.

Making a LSTM Model

Code

```
model = Sequential()
model.add(LSTM(units = 50, return_sequences = True,
input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences=True))
```

```
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss='mean_squared_error")
model.fit(X_train, y_train, epochs = 100, batch_size=32)
```

Explaining code

The provided code snippet creates a Sequential model using Keras, with three LSTM layers followed by dropout layers. The LSTM layers are recurrent neural network layers that can process sequential data. The dropout layers help prevent overfitting. The model is compiled with the Adam optimizer and mean squared error loss. It is then trained on the input data for a specified number of epochs and batch size. The goal is to predict a continuous value based on the input sequence.

Explaining Model

MODEL ARCHITECTURE

1) The model uses two LSTM layers. LSTMs (Long Short-Term Memory) are a type of recurrent neural network (RNN) that are commonly used for sequence data. The first LSTM layer has 64 units, and the second LSTM layer has 32 units.

2) The ReLU (Rectified Linear Unit) activation function is used in both LSTM layers. ReLU applies the activation function f(x) = max(0, x), which introduces non-linearity to the model and helps in capturing complex patterns in the data.

3) The first LSTM layer is set to return sequences (return_sequences=True), meaning it will output a sequence of vectors rather than a single vector. This is necessary because the second LSTM layer expects sequences as input.

4) A dropout layer is added after the LSTM layers with a dropout rate of 0.2. Dropout randomly sets a fraction of input units to 0 during training, which

helps prevent overfitting by reducing the reliance on specific input features.

5) The final layer is a dense layer, which is a fully connected layer where each unit is connected to every unit in the previous layer. The number of units in this layer is equal to the number of features in the target variable (trainY.shape[1]), which means it will output predictions for each feature.

6) The model is compiled with the Adam optimizer, which is an adaptive learning rate optimization algorithm. It adjusts the learning rate during training based on the estimated gradients for each parameter.

7) The loss function used is mean squared error (MSE), which is commonly used for regression problems. It calculates the mean squared difference between the predicted values and the actual values of the target variable. Minimizing the MSE loss function helps the model to learn and improve its predictions.

Example of LSTM model-

Dataset from: https://www.kaggle.com/rakannimer/air-passengers

International Airline Passengers prediction problem
This is a problem where, given a year and a month, the task is to predict the number of international airline passengers in units of 1,000.
The data ranges from January 1949 to December 1960, or 12 years

```
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, Flatten
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

#from #Ex keras.callbacks import EarlyStopping
from keras.layers import ConvLSTM2D
```

```
# Load the dataset

dataframe = read_csv('data/AirPassengers.csv', usecols=[1])
plt.plot(dataframe)

#Convert pandas dataframe to numpy array

dataset dataframe.values
dataset = dataset.astype('float32') #Convert values to float

#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to
# normalize the dataset

scaler = MinMaxScaler (feature_range=(0, 1)) #Also try QuantileTransformer
dataset= scaler.fit_transform(dataset)

#We cannot use random way of splitting dataset into train and test as
#The sequence of events is important for time series.
# So Let us take first 60% values for train and the remaining 1/3 for testing
# split into train and test sets

train_size = int(len(dataset) * 0.66)
test_size = len(dataset) - train_size
train, test = dataset [0:train_size,:], dataset[train_size:len(dataset),:]

#X and Y. We need to transform our data into something that Looks Like X and Y
# This way it can be trained on a sequence rather than individual data points.
# Let us convert into n number of columns for X where we feed sequence of number
#then the final column as Y where we provide the next number in the sequence as
# So Let us convert an array of values into a dataset matrix
#seq_size is the number of previous time steps to use as
#input variables to predict the next time period.
#creates a dataset where X is the number of passengers at a given time (t, t-1
#and Y is the number of passengers at the next time (t + 1).
```

```
def to_sequences (dataset, seq_size=1):
x = [ ]
y = [ ]
for i in range (len (dataset)-seq_size-1):
#print(i)

window= dataset[i: (i+seq_size), 0]
x.append(window)
y.append(dataset[i+seq_size, 0])
return np.array(x), np.array(y)
seq_size = 5 # Number of time steps to Look back

#Larger sequences (Look further back) may improve forecasting.

trainx, trainY=to_sequences (train, seq_size)
testx, testY = to_sequences (test, seq_size)
print("Shape of training set: {}".format(trainx.shape))
print("Shape of test set: {}".format(testx.shape))

# Reshape input to be [samples, time steps, features]

trainX = np.reshape (trainx, (trainx.shape [0], 1, trainx.shape [1]))
testX = np.reshape (testx, (testx.shape [0], 1, testx.shape [1]))
OOT print('Single LSTM with hidden Dense...')

model Sequential()
model.add(LSTM (64, input_shape=(None, seq_size)))
model.add(Dense(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

#monitor = EarlyStopping (monitor='val_Loss', min_delta=1e-3, patience=20,
# verbose=1, mode= 'auto', restore_best_weights=True)

model.summary()
print('Train...')

####################
#Stacked LSTM with 1 hidden dense Layer
# reshape input to be [samples, time steps, features]
#trainX = np.reshape (trainx, (trainx.shape [0], 1, trainx.shape[1]))
#testX = np.reshape (testx, (testx.shape [0], 1, testx.shape[1]))
```

```
#model = Sequential()
#model.add(LSTM (50, activation='relu', return_sequences=True,
input_shape=(None
#model.add(LSTM (50, activation='relu'))
#model.add(Dense(32))
#model.add(Dense(1))
#model.compile (optimizer= 'adam', Loss='mean_squared_error')

model. summary()

model.fit(trainx, trainy, validation data=(testx, testy),
verbose=2, epochs=100)

#moke predictions
trainPredict model.predict(traink)
testPredict model.predict(testx)

#invert predictions back to prescoled values
#This is to compare with original input values
#Since we used winmaxscaler we can now use scaler.inverse_transform
to invert the transformation.

trainPredict scaler.inverse transform(trainPredict)
trainy scaler.inverse_transform( [trainy])
testPredict scaler.inverse transform(testPredict)
testy scaler.inverse transform( [testy])

#calculate root mean squared error

trainscore= math. sqrt(mean squared_error(trainy[@], trainPredict[:,0]))
print("Train Score: %.2f RMSE % (trainScore))
testScore= math.sqrt(mean_squared_error(testy[e], testPredict[:,e]))
print("Test Score: %.2f RMSE % (testScore))

# shift train predictions for plotting
testscore math.sqrt(mean squared_error(testy[e), testPredict[:,]))

print('Test Score: %.2f RMSE % (testScore))

#shift train predictions for plotting
we must shift the predictions so that they align on the x-axis with the origin
trainPredictPlot np.empty like(dataset)
```

```
trainPredictPleot[:] np. nan
trainPredictPlot [seq_size:len(trainPredict) +seq_size, :) trainPredict

#shift test predictions for plotting

testPredictPlot = np.empty like(dataset)
testPredictPlot[:,] np. nan
testPredictPlot[len(trainPredict)+(seq_size*2)+1:len (dataset)-1, :] testPredic

# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
seplt.plot(testPredictPlot)
plt.show()
```

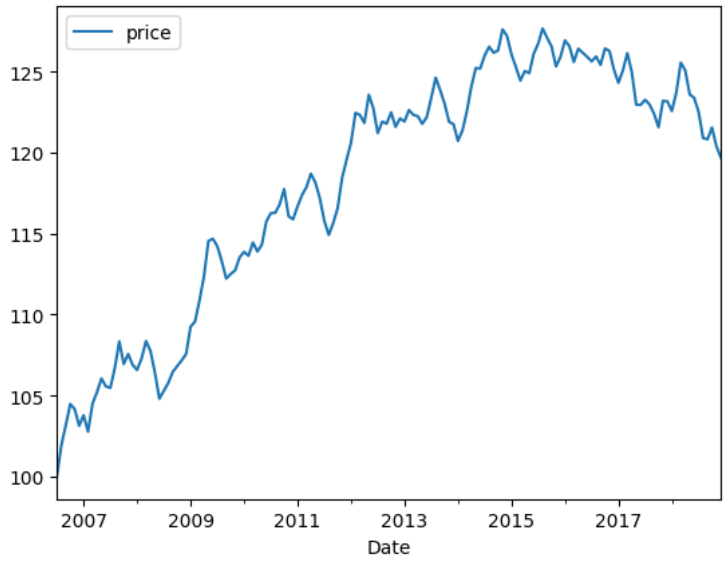
Train.csv file:

Final Code for LSTM Model

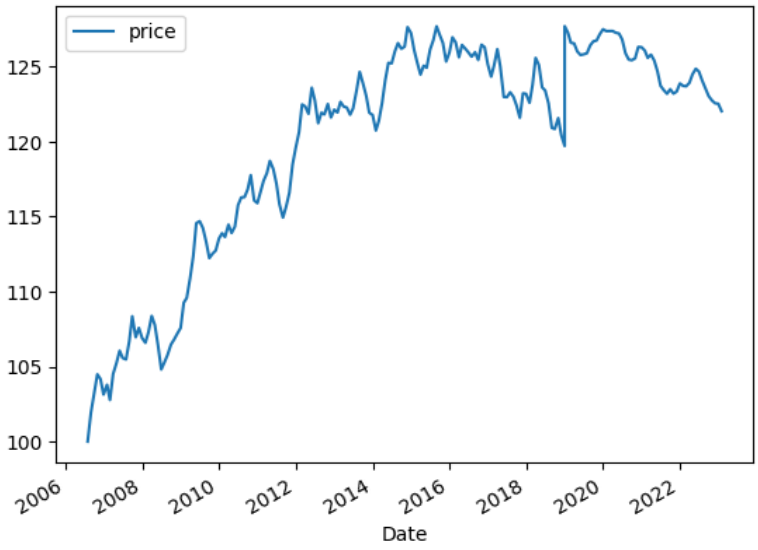
```
df = pd.read_csv('Train.csv')
print(df.head())
df=df.head(150)
cols = list(df)[2:12]
print(cols)
df_for_training = df[cols].astype(float)
scaler = StandardScaler()
scaler = scaler.fit(df_for_training)
df_for_training_scaled = scaler.transform(df_for_training)
trainX = []
trainY = []
n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14 # Number of past days we want to use to predict the future.
for i in range(n_past, len(df_for_training_scaled) - n_future +1):
    trainX.append(df_for_training_scaled[i - n_past:i, 0:df_for_training.shape[1]])
    trainY.append(df_for_training_scaled[i + n_future - 1:i + n_future, 9])

trainX, trainY = np.array(trainX), np.array(trainY)
```

```
print('trainX shape == {}.'.format(trainX.shape))
print('trainY shape == {}.'.format(trainY.shape))
df_for_training_scaled
trainY
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]),
return_sequences=True))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse')
model.summary()
3history = model.fit(trainX, trainY, epochs=5, batch_size=3, validation_split=0.1,
verbose=1)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
from pandas.tseries.holiday import USFederalHolidayCalendar
from pandas.tseries.offsets import CustomBusinessDay
us_bd = CustomBusinessDay(calendar=USFederalHolidayCalendar())
# df2= df.head(150)
train_dates = pd.to_datetime(df2['Date'])
print(train_dates.tail(15))
n_past = 1
n_days_for_prediction=50  #let us predict past 15 days

predict_period_dates = pd.date_range(list(train_dates)[-n_past],
periods=n_days_for_prediction, freq='M').tolist()
print(predict_period_dates)
prediction = model.predict(trainX[-n_days_for_prediction:])
prediction_copies = np.repeat(prediction, df_for_training.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(prediction_copies)[:,9]
forecast_dates = []
for time_i in predict_period_dates:
    forecast_dates.append(time_i.date())

df_forecast = pd.DataFrame({'Date':np.array(forecast_dates), 'price':y_pred_future})
df_forecast['Date']=pd.to_datetime(df_forecast['Date'])
original = df[['Date', 'price']]
```

```
original['Date']=pd.to_datetime(original['Date'])
original
df_forecast
original.plot(x='Date',y='price')
original= pd.concat([original,df_forecast],axis=0)
original.plot(x='Date',y='price')
#f_forecast.plot(x='Date',y='price')


def directional_accuracy(y_true, y_pred):
    y_true_direction = np.sign(np.diff(y_true))
    y_pred_direction = np.sign(np.diff(y_pred))
    da = np.mean(y_true_direction ==  y_pred_direction)
    return da
     directional_accuracy(y_true ,y_pred_future )
     Out[99]: 0.6842105263157895
```



True value



Predicted value

Predicting stock price using technical indicators

We have used the LSTM model shown above involving technical indicators to predict price. Our aim is to develop a best fit model using a combination of indicators.

Options Basics

Options are contracts that give the bearer the right but not the obligation to either buy or sell an amount of some underlying asset at a predetermined price at or before the contract expires.

Options are widely used in financial markets for various purposes, including speculation, hedging, and generating income. They provide traders with flexibility and risk management opportunities due to their unique characteristics and potential for leverage. However, options trading can be complex and involves significant risks, requiring careful consideration and understanding of the underlying assets and market conditions.

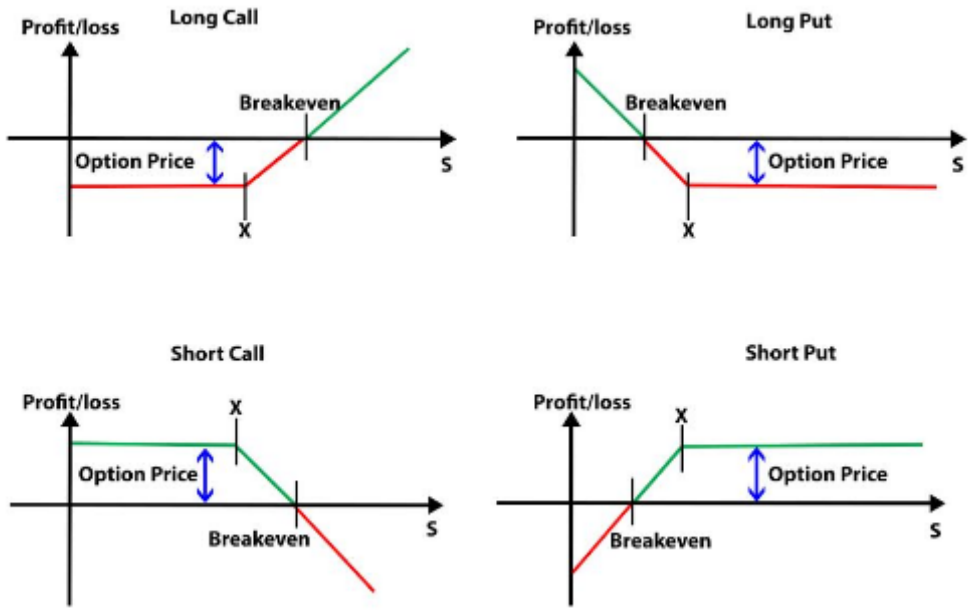
Key points about options:

1. **Call Option**: This gives the buyer the right to purchase the underlying asset at the agreed-upon price (strike price) before or on the expiration date.
2. **Put Option**: This gives the buyer the right to sell the underlying asset at the strike price before or on the expiration date.
3. **Strike Price**: The price at which the option holder can buy or sell the underlying asset.
4. **Expiration Date**: The date until which the option contract remains valid. Once the option expires, it becomes worthless.
5. **Premium**: The price paid by the option buyer to the seller for acquiring the option contract. The premium is the seller's income and the buyer's cost.

In-the-Money (ITM), At-the-Money (ATM), Out-of-the-Money (OTM): These terms describe the relationship between the current price of the underlying asset and the strike price of the option.

1. **ITM**: For a call option, when the asset price is higher than the strike price. For a put option, when the asset price is lower than the strike price.

2. **ATM**: When the asset price is approximately equal to the strike price.
3. **OTM**: For a call option, when the asset price is lower than the strike price. For a put option, when the asset price is higher than the strike price.



Options Trading Strategies

There are many options trading strategies, a few of them are explained below.

1. Butterfly Spread-A butterfly spread combines both a bull and bear spread. This is a neutral strategy that uses four options contracts with the same expiration but three different strike prices:A higher strike price ,an at-the-money strike price, a lower strike price.

2. Straddle-A straddle is a strategy accomplished by holding an equal number of puts and calls with the same strike price and expiration dates. The following are the two types of straddle positions.

- **Long Straddle**: The long straddle is designed around the purchase of a put and a call at the exact same strike price and expiration date. The

long straddle is meant to take advantage of the market price change by exploiting increased volatility. Regardless of which direction the market's price moves, a long straddle position will have you positioned to take advantage of it.

- **Short Straddle**: The short straddle requires the trader to sell both a put and a call option at the same strike price and expiration date. By selling the options, a trader is able to collect the premium as a profit. A trader only thrives when a short straddle is in a market with little or no volatility. The opportunity to profit will be based 100% on the market's lack of ability to move up or down. If the market develops a bias either way, then the total premium collected is in jeopardy.

Black Scholes Merton Model

Developed in 1973 by Fischer Black, Robert Merton, and Myron Scholes, the Black-Scholes model was the first widely used mathematical method to calculate the theoretical value of an option contract, using current stock prices, expected dividends, the option's strike price, expected interest rates, time to expiration, and expected volatility.

Black-Scholes Assumptions

The Black-Scholes model makes certain assumptions:

- No dividends are paid out during the life of the option.
- Markets are random (i.e., market movements cannot be predicted).
- There are no transaction costs in buying the option.
- The risk-free rate and volatility of the underlying asset are known and constant.
- The returns of the underlying asset are normally distributed.
- The option is European and can only be exercised at expiration

$$C = SN(d_1) - Ke^{-rt}N(d_2)$$

where:

$$d_1 = \frac{ln\frac{S}{K} + (r + \frac{\sigma_v^2}{2})t}{\sigma_s\sqrt{t}}$$

and

$$d_2 = d_1 - \sigma_s\sqrt{t}$$

and where:

$C = \text{Call option price}$

$S = \text{Current stock (or other underlying) price}$

$K = \text{Strike price}$

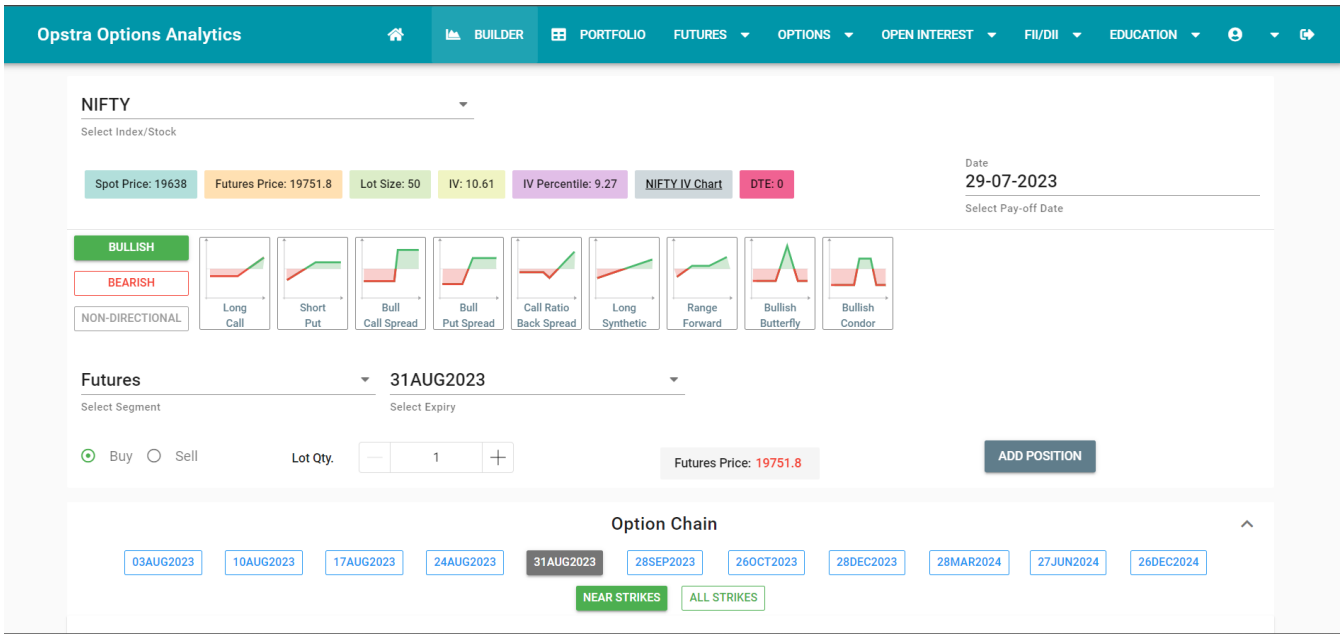
$r = \text{Risk-free interest rate}$

$t = \text{Time to maturity}$

$N = \text{A normal distribution}$

Web Scraping for extracting data from Opstra

OPSTRA is a one-stop solution for all the F&O trading needs. With its advanced features, we can perform a thorough analysis to make smart trade decisions, every time. Trade in the F&O, currency, and commodities segments with real-time price updates.
We have used python and selenium for extracting the data from the Opstra website.



Selenium is an open-source automation framework used to automate web browsers. It lets developers interact with web applications programmatically, mimicking user actions like clicking buttons and filling forms. Selenium supports multiple languages and browsers, making it versatile for web testing, scraping, and automation tasks.
We have used selenium to automate web browsing then we have used it to extract the data for 10 dates given on the Opstra Options Website.

The Code is attached below:-

```
import numpy as np
import pandas as pd
import time from selenium
import webdriver from selenium.webdriver.common.keys
import Keys from selenium.webdriver.common.by
import By from selenium.webdriver.chrome.options
import Options from selenium.webdriver.chrome.service
import ChromeDriverManager
options = webdriver.ChromeOptions()
```

```
options.add_experimental_option("prefs", { "download.default_directory":
r"C:\Users\guddu\Desktop\Hello", "download.prompt_for_download": False,
"download.directory_upgrade": True, "plugins.always_open_pdf_externally": True
})
driver = webdriver.Chrome(options=options)
driver.get('https://opstra.definedge.com')
time.sleep(2)
search_element = driver.find_element(By.XPATH,
'//*[@id="app"]/div[2]/nav/div/div[4]/button/div')
search_element.click()
search_element1 = driver.find_element(By.XPATH, '//*[@id="username"]')
search_element1.send_keys("your email")
search_element2 = driver.find_element(By.XPATH, '//*[@id="password"]')
search_element2.send_keys("password")
search_element3 = driver.find_element(By.XPATH, '//*[@id="kc-login"]')
search_element3.click()
driver.get('https://opstra.definedge.com/strategy-builder')
time.sleep(2)
search_element4 = driver.find_element(By.XPATH,
'//*[@id="app"]/div[62]/main/div/div/div/div/div[3]/div[2]/div[3]/ul/li/div[1]/
div[1]')
time.sleep(2)
search_element4.click()
for j in range(1,11):
add='//*[@id="app"]/div[62]/main/div/div/div/div/div[3]/div[2]/div[3]/ul/li/div
[2]/div[1]/div['+str(j)+']/button/div'
search_element5 = driver.find_element(By.XPATH, add )
search_element5.click()
CallLTP = driver.find_elements(By.XPATH, '//tbody/tr/td[1]')
itmprob = driver.find_elements(By.XPATH, '//tbody/tr/td[2]')
 CallIv = driver.find_elements(By.XPATH, '//tbody/tr/td[3]')
CallDelta = driver.find_elements(By.XPATH, '//tbody/tr/td[4]')
 StrikePrice = driver.find_elements(By.XPATH, '//tbody/tr/td[5]')
PutDelta = driver.find_elements(By.XPATH, '//tbody/tr/td[6]')
PutIV = driver.find_elements(By.XPATH, '//tbody/tr/td[7]')
ITMProb = driver.find_elements(By.XPATH, '//tbody/tr/td[8]')
PutLTP = driver.find_elements(By.XPATH, '//tbody/tr/td[9]')
optiondata = []
for i in range(len(CallLTP)):
     tempdata={'CallLTP': CallLTP[i].text,
     'itmprob': itmprob[i].text,
     'CallIv': CallIv[i].text,
     'CallDelta': CallDelta[i].text,
     'StrikePrice': StrikePrice[i].text,
     'PutDelta': PutDelta[i].text,
     'PutIV': PutIV[i].text,
     'ITMProb': ITMProb[i].text,
     'PutLTP': PutLTP[i].text}
     optiondata.append(tempdata)
```

```
df = pd.DataFrame(optiondata) df output_file = "output"+str(j)+".csv"
df.to_csv(output_file, index=False)
```

In this code we have first imported all the essential libraries and then we have started our web automation. Then the code opens the opstra website and then we have iterated over all the 10 dates which provide us the data about the available options contracts on those dates. We have primarily used **XPath** of the buttons present on the website to click on them via Selenium and the code we have written. We have then extracted the data which is present in the form of **HTML Table** and them we have converted it into a **Pandas** data frame. In the end we have converted it into a **CSV** file so that we can share it easily.

Here are the screenshots of the obtained **CSV** files:-

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
973.95	99.9	0	0	18800	-1.8	23.53	0	3.15
948.45	99.8	32.45	92.64	18850	-1.94	22.63	0.1	3.3
877.9	99.7	19.28	98.89	18900	-2.23	21.99	0.2	3.75
845.1	99.5	28.53	92.85	18950	-2.49	21.21	0.4	4.1
791.4	99.2	25.81	93.54	19000	-2.89	20.57	0.7	4.7
743.7	98.8	25.32	92.59	19050	-3.26	19.78	1.1	5.2
685.5	98.3	20.75	94.91	19100	-3.76	19.07	1.6	5.9
627.9	97.5	14.33	98.55	19150	-4.35	18.33	2.4	6.7
586.4	96.4	18.61	93.94	19200	-4.9	17.42	3.5	7.3
532.7	95	15.71	95.28	19250	-5.72	16.65	4.9	8.35
486.95	93.2	16.21	92.86	19300	-6.89	16.01	6.7	10
436.55	90.8	14.74	92.51	19350	-8.33	15.36	9.1	12
390.1	88	14.47	90.22	19400	-10.49	14.94	11.9	15.4
334.3	84.5	11.3	92.41	19450	-13.23	14.56	15.4	19.9
299.15	80.5	13.58	84.4	19500	-16.83	14.3	19.4	26.35
259.3	75.9	13.76	79.34	19550	-21.17	14.06	24	34.7
217.2	70.8	13.15	74.76	19600	-26.47	13.97	29.1	46.2
180.85	65.2	13.04	68.55	19650	-32.1	13.57	34.7	58.3
149.65	59.3	13.19	61.46	19700	-38.97	13.74	40.6	77.6
121.25	53.2	13.22	54.17	19750	-45.95	13.69	46.7	98.7
97.75	47	13.4	46.84	19800	-53.17	13.39	52.9	121.25
76.7	41	13.44	39.7	19850	-60.32	13.41	58.9	150.1
59.5	35.1	13.53	33.02	19900	-66.77	13.7	64.8	184.4
45.05	29.6	13.56	26.85	19950	-73.92	13.08	70.3	215.4
34.4	24.5	13.75	21.62	20000	-78.99	13.4	75.4	255.9
24.95	20	13.74	16.81	20050	-84.46	13.05	79.9	295.1
18.9	16	14.01	13.23	20100	-87.74	13.46	83.9	340.15
13.65	12.6	14.11	10.06	20150	-93.33	12.03	87.3	380.75
10.1	9.7	14.34	7.71	20200	-94.63	12.7	90.2	429.45
7.65	7.3	14.66	5.98	20250	-92.1	16.15	92.6	485.3
6	5.4	15.08	4.73	20300	0	0	94.5	515.9
4.9	4	15.6	3.86	20350	0	0	95.9	544.05
4.1	2.8	16.17	3.2	20400	-96.13	16.96	97.1	628.95
3.65	2	16.88	2.78	20450	0	0	97.9	659.45
3.45	1.3	17.74	2.53	20500	0	0	98.6	719.25
3.05	0.9	18.38	2.2	20550	0	0	99	745.85
3	0.6	19.31	2.08	20600	0	0	99.3	797
2.8	0.4	20.07	1.89	20650	0	0	99.5	769.85
2.75	0.2	20.96	1.79	20700	0	0	99.7	892.6
2.6	0.1	21.72	1.65	20750	0	0	99.8	952.3

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
1002.7	99.3	14.41	98.19	18800	-3.27	16.4	0.6	7.25
400.65	99.1	0	0	18850	-3.69	16.04	0.8	8.15
950	98.7	23.7	87.62	18900	-4.13	15.63	1.2	9.05
1029.5	98.2	36.92	76.41	18950	-4.79	15.37	1.7	10.55
847.35	97.6	21.43	87.12	19000	-5.38	14.95	2.3	11.75
938.65	96.8	35.3	74.64	19050	-6.13	14.59	3.1	13.35
755.85	95.8	20.6	84.82	19100	-6.77	14.06	4.1	14.45
662.35	94.6	12.93	93.39	19150	-8.13	13.95	5.3	17.8
652	93.1	18.12	84.09	19200	-8.97	13.38	6.8	19.2
601.3	91.2	17.03	83.44	19250	-10.08	12.88	8.7	21.25
558.9	89.1	16.85	81.39	19300	-11.48	12.43	10.8	24
519.7	86.6	16.9	78.86	19350	-13.73	12.29	13.3	29.55
469	83.7	15.71	77.78	19400	-16.11	12.05	16.2	35.3
430	80.5	15.59	75.01	19450	-18.59	11.68	19.4	41
389.8	76.9	15.25	72.3	19500	-21.82	11.48	23	49.5
344.2	72.9	14.36	69.98	19550	-25.47	11.29	27	59.6
303.7	68.7	13.8	66.88	19600	-29.46	11.05	31.2	70.9
273.7	64.2	13.96	62.79	19650	-34.1	10.96	35.7	85.95
240.15	59.6	13.69	58.89	19700	-38.89	10.68	40.3	101
207.95	54.8	13.37	54.77	19750	-44.07	10.33	45.1	117.35
176.65	49.9	12.97	50.4	19800	-49.72	10.1	50	138
150.65	45	12.8	45.84	19850	-55.45	10.13	54.9	165.1
128.65	40.3	12.75	41.33	19900	-61.65	9.63	59.6	188
104.55	35.7	12.36	36.54	19950	-67.48	9.51	64.2	219.1
88	31.3	12.38	32.25	20000	-73.69	9.09	68.6	249.8
72.2	27.1	12.29	28.03	20050	-80.56	8.38	72.8	281.2
57.95	23.3	12.15	23.95	20100	-86.03	8.02	76.6	319.95
46.95	19.8	12.12	20.39	20150	-90.08	7.86	80.1	363.1
38.6	16.6	12.21	17.37	20200	0	0	83.3	396.85
31.95	13.8	12.34	14.8	20250	0	0	86.1	441.95
25	11.3	12.28	12.17	20300	0	0	88.6	487.9
20.65	9.2	12.44	10.28	20350	0	0	90.7	533.4
16.6	7.4	12.52	8.52	20400	0	0	92.5	579.95
14.4	5.9	12.84	7.4	20450	0	0	94	600
11.9	4.6	13.01	6.23	20500	0	0	95.3	673.8
9.75	3.6	13.16	5.2	20550	0	0	96.3	685.2
7.8	2.7	13.25	4.26	20600	0	0	97.2	769.55
6.8	2.1	13.55	3.71	20650	0	0	97.8	765
5.55	1.5	13.69	3.08	20700	0	0	98.4	835
5.45	1.1	14.26	2.92	20750	0	0	98.8	900.6

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
-	-	-	-	18700	-4.87	15.89	1.3	14.3
-	-	-	-	18750	-5.12	15.39	1.7	14.7
-	-	-	-	18800	-5.76	15.18	2.2	16.6
-	-	-	-	18900	-6.99	14.56	3.5	20
820.85	95.5	0	0	18950	-	-	-	-
826.5	94.5	12.46	93.7	19000	-7.81	13.46	5.4	21.05
619.75	93.3	0	0	19050	-	-	-	-
790.7	91.9	18.35	82.13	19100	-10.24	13.18	8	28.5
722.95	88.5	19.23	77.56	19200	-12.37	12.39	11.4	33.7
752	86.4	23.88	71.57	19250	-26.99	22.05	13.5	163.8
612.8	84.1	16.65	76.66	19300	-15.94	12.01	15.8	44.7
607.4	81.6	18.86	72.08	19350	-18.26	11.93	18.3	52.65
518.05	78.9	15.17	73.94	19400	-20.44	11.66	21	59.45
447.9	75.9	12.8	74.65	19450	-23.38	11.66	24	70.75
434.65	72.8	14.32	69.69	19500	-25.86	11.26	27.1	78.05
400	69.4	14.25	66.89	19550	-28.99	11.08	30.5	89.45
367.6	65.9	14.23	63.91	19600	-32.22	10.77	34	100.5
328.9	62.3	13.75	61.18	19650	-35.9	10.57	37.6	114.65
296.3	58.5	13.55	58.04	19700	-39.85	10.36	41.4	130.45
260.25	54.7	13.06	54.84	19750	-44.15	10.32	45.2	151.35
234	50.8	13.05	51.37	19800	-48.52	9.98	49.1	169
204.4	47	12.74	47.81	19850	-53.41	9.19	52.9	181.05
171.75	43.2	12.15	43.94	19900	-58.55	8.93	56.7	204.9
158.5	39.4	12.58	40.66	19950	-62.49	9.57	60.5	246.45
132.75	35.8	12.17	36.76	20000	-67.94	8.99	64.1	270
117.8	32.3	12.3	33.5	20050	-71.17	9.5	67.6	313
98.3	28.9	12.03	29.82	20100	-74.53	9.74	71	353.25
72.7	22.8	12.05	23.74	20200	-89.35	7	77.1	404.2
50.15	17.5	11.85	18	20300	-85.97	10.1	82.4	520
36	13	11.95	13.71	20400	-	-	-	-
32	11.1	12.19	12.25	20450	-	-	-	-
26	9.5	12.11	10.37	20500	-	-	-	-
32.35	8	13.52	11.36	20550	-	-	-	-
14.7	4.6	12.73	6.15	20700	-	-	-	-
11.5	3	13.14	4.85	20800	-	-	-	-
12.2	2.5	13.81	4.89	20850	-	-	-	-
9.4	2	13.63	3.95	20900	-	-	-	-
12.95	1.6	15.01	4.8	20950	-	-	-	-
7.85	1.2	14.15	3.26	21000	0	0	98.7	1049.95
6.15	0.2	16.29	2.33	21300	-	-	-	-

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
-	-	-	-	18300	-5.36	19.31	0.3	23.1
-	-	-	-	18350	-4.96	18.2	0.5	19.85
-	-	-	-	18550	-6.23	16.79	1.3	23.85
-	-	-	-	18600	-6.9	16.69	1.7	26.75
-	-	-	-	18700	-8.18	16.29	2.6	31.95
-	-	-	-	18750	-6.78	14.46	3.2	22.65
-	-	-	-	18800	-9.98	16.08	3.9	40
-	-	-	-	18850	-11.56	16.4	4.7	48.8
-	-	-	-	18900	-8.16	13.26	5.6	25.85
-	-	-	-	19000	-9.84	12.75	7.9	31.05
-	-	-	-	19050	-18.64	17.55	9.2	94.7
674.75	89.1	0	0	19100	-	-	-	-
-	-	-	-	19200	-15.01	11.99	14.4	48.95
574.95	76.3	15.65	70.84	19400	-	-	-	-
458.55	70.7	13	69.19	19500	-27.56	10.82	29.2	96.55
648.1	67.7	24.36	59.87	19550	-31.55	11.43	32.2	122.6
418.75	64.6	14.3	62.63	19600	-33.4	10.45	35.3	121.15
374.25	61.4	13.57	60.47	19650	-36.73	10.33	38.5	136.85
341.45	58.2	13.38	57.77	19700	-40.22	10.19	41.7	153.9
309.6	54.9	13.17	54.97	19750	-43.55	9.45	45	160.3
277.65	51.5	12.88	52.07	19800	-47.63	9.57	48.4	186.15
259.05	48.2	13.17	49.13	19850	-51.72	9.32	51.7	206.25
223.45	44.8	12.57	45.97	19900	-55.83	9.31	55.1	233.5
188.15	41.6	11.89	42.48	19950	-59.14	9.99	58.3	276.55
173	38.4	12.13	39.57	20000	-64.62	8.79	61.5	284.2
153.1	35.2	12.05	36.48	20050	-64.68	11.13	64.7	362
127.95	32.2	11.61	32.93	20100	0	0	67.7	275.75
122.4	29.3	12.15	30.87	20150	-	-	-	-
99.9	26.5	11.67	27.34	20200	0	0	73.4	330.75
76.5	21.4	11.69	22.28	20300	-	-	-	-
44	13.2	11.84	14.23	20500	-	-	-	-
118.4	11.5	17.83	22.67	20550	-	-	-	-
29.7	10	11.59	10.48	20600	-	-	-	-
67.8	7.5	16.16	15.76	20700	-	-	-	-
17.95	5.4	12.11	6.62	20800	-	-	-	-
19.4	4.6	12.79	6.75	20850	-	-	-	-
16	3.9	12.72	5.75	20900	-	-	-	-
12.55	1.9	13.82	4.35	21100	-	-	-	-
10	1	14.46	3.43	21250	-	-	-	-
10	0.8	14.86	3.35	21300	-	-	-	-

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
1029.05	91.1	12.2	90.4	18900	-11.03	13.03	8.8	45.9
1025.3	89.9	15.13	84.32	18950	-11.99	12.88	10	50.25
952.45	88.6	13.08	86.37	19000	-13.18	12.82	11.3	56.15
900	87.2	12.39	86.23	19050	-13.93	12.48	12.7	58.5
852.7	85.7	12.06	85.38	19100	-15.22	12.37	14.2	64.75
832.9	84.1	13.38	81.41	19150	-16.46	12.2	15.8	70.35
765.5	82.4	11.8	82.64	19200	-18.06	12.16	17.5	78.85
761.2	80.6	13.73	77.5	19250	-19.61	12.04	19.3	86.7
700.95	78.7	12.59	77.53	19300	-21.29	11.93	21.2	95.5
668.4	76.6	12.84	75.2	19350	-22.94	11.74	23.3	103.55
615.25	74.5	12.05	74.45	19400	-24.81	11.62	25.4	113.55
597.9	72.3	12.9	71.08	19450	-26.89	11.54	27.6	125.55
541.55	70	11.93	70.25	19500	-29.17	11.54	29.9	140.05
510.95	67.6	12.07	67.76	19550	-31.42	11.44	32.3	153.75
468	65.1	11.63	65.89	19600	-33.84	11.41	34.8	169.9
427.55	62.6	11.27	63.76	19650	-36.11	11.15	37.3	181.9
397.45	60.1	11.29	61.13	19700	-38.97	11.4	39.8	207.55
371.15	57.5	11.4	58.4	19750	-41.4	11.08	42.4	220.3
342.8	54.8	11.38	55.74	19800	-44.12	11.01	45.1	240.45
311.65	52.2	11.2	53.07	19850	-46.88	10.91	47.7	261.4
283.95	49.6	11.1	50.32	19900	-49.7	10.88	50.3	285.5
252.15	46.9	10.79	47.45	19950	-52.59	10.69	53	306.8
235.4	44.3	11	44.76	20000	-55.47	10.64	55.6	333.3
210.6	41.7	10.85	41.9	20050	-58.4	10.52	58.2	359.45
188.2	39.2	10.74	39.05	20100	-61.49	10.28	60.7	383.9
166.85	36.7	10.61	36.18	20150	-65.05	9.77	63.2	403.85
152.05	34.3	10.7	33.67	20200	-67.31	10.1	65.6	445.2
130.9	31.9	10.45	30.69	20250	-69.91	10.13	68	480.8
117.5	29.6	10.49	28.26	20300	-72.52	10.1	70.3	516.25
103.45	27.4	10.43	25.78	20350	-73.32	10.87	72.5	570
91.65	25.3	10.43	23.51	20400	-76.93	10.24	74.6	595.05
81.45	23.3	10.46	21.42	20450	-81.55	9.26	76.6	617
72.2	21.4	10.49	19.45	20500	-83.1	9.46	78.5	661.9
-	-	-	-	20550	-86.16	9.04	80.3	698
53	17.8	10.33	15.39	20600	-86.12	9.71	82.1	751.15
46.95	16.2	10.39	13.89	20650	0	0	83.7	719.75
40.7	14.7	10.38	12.37	20700	-96.04	6.9	85.2	814.35
-	-	-	-	20750	0	0	86.6	802.45
31.05	12	10.45	9.84	20800	0	0	87.9	904.55
-	-	-	-	20850	0	0	89.1	828.75

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
1105.6	83.2	12.32	84.52	19000	-17.36	13.38	16.7	109.85
1034.7	82	10.86	86.22	19050	-18.34	13.25	17.9	116.55
1162.35	80.7	17.78	74.38	19100	-19.26	13.05	19.2	122.2
978.85	79.3	12.11	81.09	19150	-20.08	12.77	20.6	126.05
936	77.9	11.97	79.95	19200	-21.43	12.75	22	136.9
881.65	76.5	11.34	79.68	19250	-22.17	12.35	23.4	138.5
864.25	75	12.17	76.65	19300	-23.82	12.45	24.9	153.4
800	73.4	11.15	76.88	19350	-25.34	12.45	26.5	166.45
799.9	71.8	12.49	73.02	19400	-26.86	12.41	28.1	179.25
819.25	70.2	14.33	69.15	19450	-28.27	12.26	29.7	189.75
717.8	68.6	12.07	70.31	19500	-29.83	12.17	31.3	202.55
708.8	66.9	12.88	67.61	19550	-31.18	11.88	33	210
647.35	65.1	11.95	66.94	19600	-33.19	12.07	34.8	232.65
695	63.4	14.44	62.98	19650	-35.12	12.19	36.5	254.45
579.95	61.6	11.82	63.38	19700	-36.71	11.92	38.3	264.85
539.95	59.8	11.52	61.72	19750	-38.79	12.18	40.1	293.3
514.45	58	11.64	59.67	19800	-40.41	11.77	41.9	300.45
511.2	56.1	12.39	57.36	19850	-42.37	11.78	43.8	322.55
449.4	54.3	11.37	55.81	19900	-44.25	11.53	45.6	336.85
420	52.5	11.28	53.78	19950	-46.2	11.16	47.4	347.65
398.85	50.6	11.4	51.72	20000	-48.29	11.17	49.3	372.55
350	48.8	10.69	49.53	20050	-50.39	11.05	51.1	394.15
342.75	46.9	11.16	47.55	20100	-52.45	11.16	53	424.45
326.55	45.1	11.34	45.59	20150	-55.81	9.23	54.8	386.8
295.35	43.3	11.05	43.35	20200	-56.57	11.16	56.6	480.85
260.65	41.5	10.61	40.88	20250	-60	9.8	58.4	465.55
249.25	39.8	10.84	39.06	20300	-60.47	11.27	60.1	545
250.05	38	11.43	37.77	20350	-67.49	8.15	61.9	477.5
210	36.3	10.72	34.88	20400	-64.74	10.98	63.6	600
-	-	-	-	20450	-68.1	10.13	65.3	607.45
176.7	33	10.65	30.9	20500	-70.12	10.11	66.9	642.15
-	-	-	-	20550	-82.28	6.51	68.5	584.5
142.25	29.8	10.41	26.74	20600	-	-	-	-
179.15	28.2	12.15	28.39	20650	-	-	-	-
118.1	26.7	10.39	23.24	20700	-	-	-	-
110.9	25.3	10.52	21.93	20750	-	-	-	-
98.75	23.9	10.43	20.17	20800	-	-	-	-
82.45	21.2	10.49	17.42	20900	-	-	-	-
69.9	18.8	10.61	15.12	21000	-89.23	8.89	81.1	1020.6
49.1	14.4	10.8	11.17	21200	-	-	-	-

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
8800	100	0	0	11000	-0.16	32.1	0	2.05
-	-	-	-	12000	-0.25	28.85	0	3.05
7164	99.9	42.15	95.56	13000	-0.71	27.91	0	9.05
-	-	-	-	14000	-1.45	26.03	0	18.6
-	-	-	-	14500	-0.41	19.11	0	3.35
5140	99.8	28.76	94.83	15000	-2.18	22.71	0.1	25.55
4190.9	99.1	26.31	91.79	16000	-3.32	19.37	0.8	34.95
3850	98	29.78	86.29	16500	-	-	-	-
3269.3	96	23.86	87.09	17000	-6.13	16.87	3.9	61.6
2830.7	92.5	22.85	83.67	17500	-8.55	15.71	7.4	84.9
2374.25	87.4	20.93	80.31	18000	-12.01	14.57	12.5	118.45
1909.25	80.3	18.53	76.48	18500	-	-	-	-
1542.85	71.6	18.04	69.61	19000	-24.4	12.55	28.3	249.3
837.45	51	15.45	53.03	20000	-47.14	10.16	48.9	508.65
339.45	31.1	13.3	31.4	21000	-93.03	4.71	68.8	967.45
107.8	16.2	12.35	13.55	22000	0	0	83.7	1706.2
54	7.2	13.71	6.99	23000	0	0	92.7	2555
33.15	2.8	15.35	4.18	24000	-	-	-	-
25.1	0.9	17.26	2.97	25000	-	-	-	-

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
5000	99.6	0	0	15000	-4.62	22.72	0.3	77.7
-	-	-	-	16000	-3.54	16.26	1.7	40
-	-	-	-	17000	-8.49	16.03	5.8	110
-	-	-	-	18000	-14.35	13.92	14.6	180
1860.95	71.2	17.94	68.81	19000	-25.41	12.22	28.7	327.3
1168	53.7	16.02	55.75	20000	-43.42	10.18	46.2	574.35
601.2	36.3	13.92	39.11	21000	-73.84	6.97	63.6	970
-	-	-	-	22000	0	0	77.9	1080.05
-	-	-	-	24000	-90.39	15.23	94	3900

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
-	-	-	-	14000	-1.67	19.1	1.3	24
-	-	-	-	15000	-4.6	19.95	3.5	80
-	-	-	-	16000	-7.23	18.25	7.7	124
-	-	-	-	17000	-10.75	16.12	14.2	175
-	-	-	-	18000	-17.02	14.5	23.2	276.95
2550	71.4	19.1	73.04	18500	-	-	-	-
-	-	-	-	19000	-27.05	13.15	34	457.3
1500	54.2	16.85	57.67	20000	-41.44	11.88	45.7	750
1439.1	42.8	21.85	48.43	21000	-	-	-	-

CallLTP	itmprob	CallIv	CallDelta	StrikePrice	PutDelta	PutIV	ITMProb	PutLTP
8200	97.4	49.63	83.02	14000	-2.2	17.08	2.5	37.05
-	-	-	-	15000	-4.68	17.08	5.3	88.5
4995	90	19.86	88.51	16000	-8.07	16.52	9.9	162.5
4161	83.6	19.28	83.33	17000	-11.54	14.8	16.3	222.7
3440	75.6	19.38	76.32	18000	-18	14	24.3	366.25
3080	71.1	19.05	72.74	18500	-	-	-	-
2732.1	66.4	18.64	68.92	19000	-26.89	13.27	33.5	586
2050	56.7	17.37	60.53	20000	-38.05	12.29	43.2	878.15
1461	47	16.23	50.81	21000	-51.92	10.97	52.9	1252
990	37.9	15.33	40.39	22000	-68.83	9.39	62	1750
675	29.8	15.07	30.94	23000	-83.46	8.65	70.1	2490