Configuration Manual

MSc Research Project
FinTech

Trinh Anh Tuan
Student ID: X22117865

School of Computing
National College of Ireland

Supervisor: Brian Byrne

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Trinh Anh Tuan

Student ID: X22117865

Programme: FinTech **Year:** 2022 - 2023

Module: Research Project

Supervisor: Brian Byrne

Submission Due Date: 14 Aug 2023

Project Title: Efficient Calibration of Implied Volatility

Word Count: **5010 Page Count: 17**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Tuan

Date: 13 Aug 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	Tuan
Date:	13 Aug 2023
Penalty Applied (if applicable):	

Configuration Manual

Trinh Anh Tuan
Student ID: X2217865

1 Introduction

This paper implements the system setup and other technique setup used in Research Project: "Efficient Calibration of Implied Volatility".

Google Colab is the virtual environment that the Research Project developed on. All the data source and the results are then uploaded to Google Drive: https://drive.google.com/drive/folders/1E0gkCl3Sd6GKDxdxaA360EmQ2zhSHnxd?usp=drive_link

2 Configuration

2.1 Devices specification

Model: Surface laptop 3
Processor: Intel(R) Core(TM) i5-1035G7 CPU @ 1.20GHz 1.50 GHz
Ram: 8 GB
System type: 64-bit operating system, x64-based processor
Edition: Windows 10 Home

2.2 2.2 Software specification

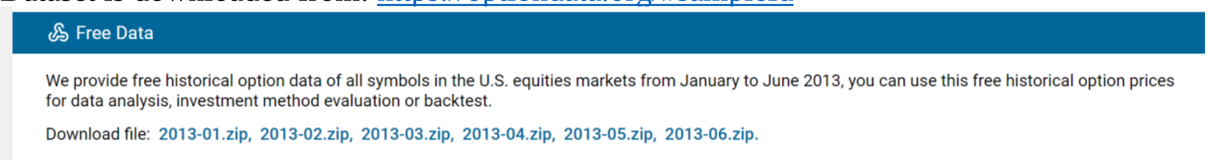
Gmail account with the accession to Google Drive and Google Colab

3 Research Project

This project is setup in Google Colab. This project is divided into three mains parts including: download data and data preprocessing, compute and visualize implied volatility surface, predict and visualize predicted implied volatility surface.

3.1 Download and Preprocess Options Data

Dataset is downloaded from: https://optiondata.org/#sampleId



The data is free and includes 6 months daily option trading.

After downloading the data, the dataset is about 7 GBs, and it is uploaded to Google Drive. Then in Colab, first I import the libraries needed:

Import Libaries

```
[ ] import pandas as pd
    import numpy as np
    import os

    import warnings
    warnings.filterwarnings('ignore')
```

```
[ ] # mount Google Colab to Google Drive -> this is to save data to Google Drive later
    from google.colab import drive as colab_drive
    colab_drive.mount('/content/drive')
```

```
# import data from Google Drive

# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

After that, I setup code for downloading dataset from shared links and choose 18 stocks that I need to analyse:

Download & Proprocess data

```
# download data from shared links
def download_data_from_link(link):
  # get file id
  fluff, _id = link.split('/d/')
  id, _ = _id.split('/')
  # print("File id = ", id)

  # download file from the id
  downloaded = drive.CreateFile({'id':id})
  downloaded.GetContentFile('Filename.csv')
  _df = pd.read_csv('Filename.csv')
  return _df
```

```
[ ] STOCKS_LIST = ['MSFT','AAPL','V','UNH','JPM','JNJ','WMT','PG','HD','CVX','KO','MRK','CRM','MCD','CSCO','NKE','DIS','INTC']
```

Dividend yield is one important variable needed so I calculate it by import dividend cash flow and divided to stock price. In order to do this, I use yfinance package. After calculating the dividend yield, I save it to Google Drive

▾ Dividend data

```
# get dividend info from yahoo finance
import yfinance as yf

# compute annual dividend yield
dividend_yield = []
for ticker in STOCKS_LIST:
  stock = yf.Ticker(ticker)
  stock_historical = stock.history(start="2013-01-01", end="2013-12-31", interval="1d")
  div = np.sum(stock_historical.Dividends)/stock_historical.Close[-1]
  dividend_yield.append(div)

dividends = pd.DataFrame(list(zip(STOCKS_LIST, dividend_yield)), columns=['ticker', 'dividend_yield'])
dividends
```

```
# save data to Drive

# make a new folder in drive
if os.path.exists('/content/drive/My Drive/data_implied_volatility/preprocessed') == False:
  os.makedirs('/content/drive/My Drive/data_implied_volatility/preprocessed', exist_ok=True)

dividends.to_csv('/content/drive/My Drive/data_implied_volatility/preprocessed/dividends_2013.csv', index=False)
```

In the next step, I import dataset from Google Drive link: https://drive.google.com/file/d/1bY01zq5Wnm0FZOyDhVjQLzEtQQHKF8uW/view
This link includes all the path link to all files that I download from Website.

```
# get data from list of google drive link
gdrive_links_list = "https://drive.google.com/file/d/1bY01zq5Wnm0FZOyDhVjQLzEtQQHKF8uW/view?usp=share_link"
links_to_data = download_data_from_link(gdrive_links_list)
links_to_data["date"] = pd.to_datetime(links_to_data["date"])
links_to_data.sort_values(by="date", inplace=True)
links_to_data
```

Then I run code to get and merge all the options files into one files and just take the columns that I need for my calculation. After running all the code, I save the result to Drive for later import needed:

```
## run time: 102 days, 18 stocks => 9 mins
# download and preprocess options data
option_links = links_to_data[links_to_data["data"]=="option"]["URL"].to_list()
option_dates = links_to_data[links_to_data["data"]=="option"]["date"].to_list()

# for count, link in tqdm(links_to_data.index(), desc='Getting Documents from {} Fillings'.format(cik), unit='filling'):
for count, link in enumerate(option_links):
  _df = download_data_from_link(link)
  print("Processing Option data on: ", option_dates[count])
  # preprocess stocks
  # get only stocks with dividend yield
  _df = _df[_df["underlying"].isin(STOCKS_LIST)]
  # calculate important info
  _df["option_price"] = (_df["bid"] + _df["ask"]) / 2
  _df["expiration"] = pd.to_datetime(_df["expiration"])
  _df["quote_date"] = pd.to_datetime(_df["quote_date"])
  _df["time2mature_days"] = (_df["expiration"] - _df["quote_date"])/ pd.to_timedelta(1, unit='D')
  _df["time2mature"] = _df["time2mature_days"]/365


  # get only relevant data
  _df = _df[["underlying", "type", "expiration", "quote_date", "strike", "option_price", "time2mature_days", "time2mature", "implied_volatility"]]
  _df.rename(columns={"underlying": "ticker", "type": "option_type"}, inplace=True)

  # merge data from all download stock into one file
  if count > 0:
    options_df = pd.concat([options_df, _df], ignore_index=True)
  else:
    options_df = _df

options_df
```

```
# save data to Drive
options_df.to_csv('/content/drive/My Drive/data_implied_volatility/preprocessed/options_2013.csv', index=False)
```

I repeat the same process with stocks information in order to get the stock price come with the option information in 2013:

Stocks Data

```
[ ] ## run time: 102 days, 18 stocks => 1.5 mins
    # download and preprocess stock data
    stock_links = links_to_data[links_to_data["data"]=="stock"]["URL"].to_list()
    stock_dates = links_to_data[links_to_data["data"]=="stock"]["date"].to_list()

    for count, link in enumerate(stock_links):
      _df = download_data_from_link(link)
      print("Processing Stocks data on: ", stock_dates[count])
      # preprocess stocks
      # get only stocks with dividend yield
      _df = _df.merge(dividends, left_on=["symbol"], right_on=["ticker"], how="right")
      # get only symbol and close price
      _df = _df[["symbol", "close"]]
      _df["quote_date"] = stock_dates[count]
      _df["quote_date"] = pd.to_datetime(_df["quote_date"])

      # rename data
      _df.rename(columns={"symbol": "ticker", "close": "underlying_price"}, inplace=True)

      # merge data from all download stock into one file
      if count > 0:
        stocks_df = pd.concat([stocks_df, _df], ignore_index=True)
      else:
        stocks_df = _df

    stocks_df
```

```
[ ] # save data to Drive
    stocks_df.to_csv('/content/drive/My Drive/data_implied_volatility/preprocessed/stocks_2013.csv', index=False)
```

Risk free rate is another factor that needs to be imported and I use pandas to import and then I save the results to Google Drive:

Risk-free rate

```
[ ] import pandas_datareader.data as web
    from datetime import datetime
```

```
[ ] ### Py ###
    # us treasury bond yield yahoo finance symbols: ['DGS10', 'DGS5', 'DGS2', 'DGS1', 'DGS1MO', 'DGS3MO']
    symbol = 'DGS1' # 1-Year Treasury Constant Maturity Rate
    us_yield = web.DataReader(symbol, 'fred', start=datetime(2013,1,1), end=datetime(2013,12,31))

    us_yield
```

```
[ ] rf_rates = us_yield.reset_index()
    rf_rates.rename(columns={"DATE": "quote_date", symbol: "rf_rate"}, inplace=True)
    rf_rates["quote_date"] = pd.to_datetime(rf_rates["quote_date"])
    rf_rates["rf_rate"] = rf_rates["rf_rate"]/100
    rf_rates
```

```
[ ] # save data to Drive
    rf_rates.to_csv('/content/drive/My Drive/data_implied_volatility/preprocessed/rf_rates_2013.csv', index=False)
```

After getting all the information, I merge data from all sources and then save it to Google Drive. The link that contain all results file for this step is in this link: https://drive.google.com/drive/folders/1-24VZyWLK2JQVkpSyUz70FvOpX9JyrtL

Merge data from all sources

```
[ ] # merge option and stock data
    df = options_df.merge(stocks_df, on=["ticker", "quote_date"])
    df
```

```
[ ] # merge risk-free data
    df = df.merge(rf_rates, on=["quote_date"])
    df
```

```
[ ] # merge dividend yield
    df = df.merge(dividends, on='ticker')
    df
```

```
[ ] # save data to Drive
    df.to_csv('/content/drive/My Drive/data_implied_volatility/preprocessed/all_data_2013.csv', index=False)
```

3.2 Compute and Visualize Implied Volatility

3.2.1 Compute and Visualize Implied Volatility for European option

First I import libraries needed:

Import Libaries

```
[ ] ### Py ###
    import pandas as pd
    import numpy as np
    import scipy.stats as si
    from scipy.optimize import fmin  #optimisation function using Nelder-Mead simplex algorithm


    import warnings
    warnings.filterwarnings('ignore')
```

```
[ ] # mount Google Colab to Google Drive -> this is to save data to Google Drive later
    from google.colab import drive as colab_drive
    colab_drive.mount('/content/drive')

    Mounted at /content/drive
```

After that, I import the data that was uploaded to Google Drive in the step 1:

Import data

```
[ ] # import data from Google Drive

    # Code to read csv file into Colaboratory:
    !pip install -U -q PyDrive
    from pydrive.auth import GoogleAuth
    from pydrive.drive import GoogleDrive
    from google.colab import auth
    from oauth2client.client import GoogleCredentials
    # Authenticate and create the PyDrive client.
    auth.authenticate_user()
    gauth = GoogleAuth()
    gauth.credentials = GoogleCredentials.get_application_default()
    drive = GoogleDrive(gauth)
```

```
[ ]
    # download data from shared links
    def download_data_from_link(link):
      # get file id
      fluff, _id = link.split('/d/')
      id, _ = _id.split('/')
      # print("File id = ", id)

      # download file from the id
      downloaded = drive.CreateFile({'id':id})
      downloaded.GetContentFile('Filename.csv')
      _df = pd.read_csv('Filename.csv')
      return _df
```

```
[ ] # import data from a preprocessed, saved file
    link = "https://drive.google.com/file/d/1-TTT80ApS1WOCAOooWOH1BWRqICWM0Op/view?usp=share_link"
    df = download_data_from_link(link)
    df
```

Next, I setup formula function for Black-Scholes model:

```
[ ] # cumulative normal distribution function
    def NORMSDIST(x):
        NORMSDIST = si.norm.cdf(x,0.0,1.0)
        return(NORMSDIST)

    # Black Scholes model for call option
    def BlackScholes(option_type, S,K,r,T,sigma,q):
        d1 = ( (np.log(S/K)+(r+0.5*sigma**2)*T) / (sigma*np.sqrt(T)) )
        d2 = ( (np.log(S/K)+(r-0.5*sigma**2)*T) / (sigma*np.sqrt(T)) )
        if option_type == "call":
          option_price = S*np.exp(-q*T)*NORMSDIST(d1) - K*np.exp(-r*T)*NORMSDIST(d2)
        elif option_type == "put":
          option_price = K*np.exp(-r*T)*NORMSDIST(-d2) - S*np.exp(-q*T)*NORMSDIST(-d1)
        else:
          print("Call or put not defined")
          option_price = np.nan
        return option_price
```

Then I create a function that makes use of Nelder-Mead simplex algorithm for implied volatility solver.

```
from scipy.optimize import fminbound   # Nelder-Mead simplex algorithm
import scipy as sq

def compute_iv_eu(option_type, S, K, T, r, q, option_price):
    # This function is the implied volatility solver which makes use of Nelder-Mead simplex algorithm
    # The paramaters are the same as described as in LeisenReimerBinomial function, except for Option_Value
    # option_price is the parameter of the option value (for which you want to calculate the Implied Volatility)

    def objective_function(iv):
      # This is the objection function for Nelder-Mead simplex algorithm
      result = (option_price - BlackScholes(option_type, S,K,r,T,iv,q))**2
      return result

    # initialize starting point for faster conversion
    start_point = np.sqrt(2*abs(((np.log(S/K)+r*T)/T)))
    res = sq.optimize.fmin(objective_function, start_point, full_output=True, ftol=0.001, maxiter=50, disp=0)
    iv_result = res[0][0]
    error = 100*(np.sqrt(res[1])/option_price)
    if (iv_result<=0) or (iv_result>1):
      res = sq.optimize.fminbound(objective_function, 0.01, 1, full_output=True, xtol = 0.001, maxfun=50, disp=0)
      iv_result = res[0]
      error = 100*(np.sqrt(res[1])/option_price)

    return [iv_result, error]
```

After this step, I need to check if the this solver works well or not:

```
[ ] # Test to show that the IV solver works
    _option_type = 'put'
    _underlying_price = 100
    _strike = 100
    _ttm = 1
    _r = 0.01
    _q = 0.1
    _option_price = 0.5

    optionvalue = BlackScholes(option_type=_option_type, S=_underlying_price,
                               K=_strike, r=_r, T=_ttm, sigma=0.5, q=_q)
    print('The value of an American put option, with IV of 50% is equal to ', str(optionvalue))
    [IV_value, error] = compute_iv_eu(_option_type, _underlying_price, _strike, _ttm, _r, _q, optionvalue)
    print('The implied volatility is ', str(IV_value))
    error

    The value of an American put option, with IV of 50% is equal to  22.894799772397825
    The implied volatility is  0.5000018341030819
    6.746272927671271e-05
```

After this step, I start to estimate implied volatility using time to maturity and strike price:

```
[ ] # estimate implied volatility using time2mat and strike
    def estimate_iv_eu(ticker, time2mature_days, strike, option_type="call", df=df):
        # The loop below will use ttm and K inputed to find other information includes in bsmvol:
        # market price = (bid+ask)/2, stockprice = price of underlying asset,
        # rr = 10 years treasury rate extracted from DGS10, ti2ma = time to maturity/365, ty = type call option
        matched_df = df[(df['ticker']==ticker) & (df['time2mature_days']==time2mature_days) & (df["strike"]==strike) & (df["option_type"]==option_type)]
        # If cannot find the match ttm and K inputed, the fuction return NA, if not, the information found will be used to calcualted implied volatility
        if len(matched_df)>0:
            res = matched_df.apply(lambda x: compute_iv_eu(option_type=x["option_type"], S=x["underlying_price"],
                                            K=x["strike"], T=x["time2mature"], r=x["rf_rate"], q=x["dividend_yield"], option_price=x["option_price"]), axis=1)
            # return iv where error is smallest
            res = [v for v in res]
            res = pd.DataFrame(res, columns=['iv', 'error'])
            res.dropna(inplace=True)
            iv = res.loc[res['error'].idxmin()]['iv']
        else:
            iv = np.nan

        return iv
```

Then I test if the results is accurate or not and the results are good when it has high accuracy with the real data:

```python
# test
def _print_test_result(test_df):
  _ticker = test_df["ticker"].to_list()[0]
  _option_type = test_df["option_type"].to_list()[0]
  _time2mature_days = test_df["time2mature_days"].to_list()[0];
  _strike = test_df["strike"].to_list()[0];
  _option_price = test_df["option_price"].to_list()[0]
  _iv = estimate_iv_eu(ticker=_ticker, time2mature_days=_time2mature_days, strike=_strike, option_type=_option_type)
  # recalculate option price using BSM model and calculated iv
  _underlying_price = test_df["underlying_price"].to_list()[0]
  _r = test_df["rf_rate"].to_list()[0]
  _q = test_df["dividend_yield"].to_list()[0]
  _time2mature = test_df["time2mature"].to_list()[0];
  _bs_price = BlackScholes(option_type=_option_type, S=_underlying_price, K=_strike, r=_r, T=_time2mature, sigma=_iv, q=_q)
  print(f"   {_ticker}, {_option_type},\tTTM={_time2mature_days},\tStrike={_strike},\t==> iv={_iv}, \t==> BS option price = {_bs_price}, \tMarket price = {_option_price}")

# test for AAPL
# test function for call option
print("EU option implied volatility: ")
_ticker = "AAPL"; _option_type = "call"
test1 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].head(1)
_print_test_result(test1)
test2 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].tail(1)
_print_test_result(test2)

# test function for put option
_ticker = "AAPL"; _option_type = "put"
test3 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].head(1)
_print_test_result(test3)
test4 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].tail(1)
_print_test_result(test4)
```

```
EU option implied volatility:
   AAPL, call,  TTM=2.0,     Strike=400.0,   ==> iv=0.5456510797044793,   ==> BS option price = 148.96044108975934,    Market price = 148.75
   AAPL, call,  TTM=585.0,   Strike=1050.0,  ==> iv=0.33544215316947845,  ==> BS option price = 1.5194792303293276,    Market price = 1.52
   AAPL, put,   TTM=2.0,     Strike=400.0,   ==> iv=0.5784755972513936,   ==> BS option price = 1.8847652390696487e-13, Market price = 0.005
   AAPL, put,   TTM=585.0,   Strike=1050.0,  ==> iv=0.14534807366140126,  ==> BS option price = 626.7000511081371,     Market price = 623.8
```

Second test with other stock and the results still good:

```python
# test for MSFT
# test function for call option
print("EU option implied volatility: ")
_ticker = "MSFT"; _option_type = "call"
test1 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].head(1)
_print_test_result(test1)
test2 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].tail(1)
_print_test_result(test2)

# test function for put option
_ticker = "MSFT"; _option_type = "put"
test3 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].head(1)
_print_test_result(test3)
test4 = df[(df["ticker"]==_ticker) & (df["option_type"]==_option_type)].tail(1)
_print_test_result(test4)
```

```
EU option implied volatility:
   MSFT, call,  TTM=2.0,     Strike=19.0,   ==> iv=0.9111242148970856,   ==> BS option price = 8.61550170036777,      Market price = 8.55
   MSFT, call,  TTM=585.0,   Strike=50.0,   ==> iv=0.24021270954598467,  ==> BS option price = 0.47988918916959644,   Market price = 0.48
   MSFT, put,   TTM=2.0,     Strike=19.0,   ==> iv=0.999607542093466,    ==> BS option price = 6.782222250344414e-08, Market price = 0.01
   MSFT, put,   TTM=585.0,   Strike=50.0,   ==> iv=0.1890980595147025,   ==> BS option price = 16.87485500709671,     Market price = 16.875
```

In the next code, because the dataset is large so I setup the time step to divide the dataset into month to process easier:

```python
time_steps = ["2013-01-31", "2013-02-28", "2013-03-31", "2013-04-30", "2013-05-31", "2013-06-30"]
time_steps = pd.to_datetime(time_steps)
time_steps
```

```
DatetimeIndex(['2013-01-31', '2013-02-28', '2013-03-31', '2013-04-30',
               '2013-05-31', '2013-06-30'],
              dtype='datetime64[ns]', freq=None)
```

In the next step, I run the code to compute the implied volatility and all the results will be saved to Google Drive automatically when it finished running:

```python
# compute implied volatility for all row in the dataset
## execution time:
##    5 days, 18 stocks ->  43,707 rows ->    11mins
##  102 days, 18 stocks -> 881,253 rows -> 2h 30mins

import time
import os

_STOCKS_LIST = STOCKS_LIST
n_stocks = len(_STOCKS_LIST)

_time_steps = time_steps
n_time = len(_time_steps)
_input = ["option_type", 'underlying_price', 'strike', 'time2mature', 'rf_rate', 'dividend_yield', 'option_price', 'quote_date']

for _c, stock in enumerate(_STOCKS_LIST):
  if stock != "AAPL":
    _time_steps = [pd.to_datetime('2013-06-30')]
    n_time = len(_time_steps)

  for c, _time in enumerate(_time_steps):
    start = time.time()
    if c == 0:
      sub_df = df.loc[(df['ticker']==stock) & (df["quote_date"] <= _time)][_input]
    else:
      sub_df = df.loc[(df['ticker']==stock) & (df["quote_date"] > _time_steps[c-1]) & (df["quote_date"] <= _time)][_input]

    n_options = len(sub_df)
    n_days = len(sub_df["quote_date"].unique())
    start_date = sub_df["quote_date"].min().date()
    end_date = sub_df["quote_date"].max().date()
    print(f"{_c+1}/{n_stocks}: {stock} - {c+1}/{n_time} {start_date} to {end_date}: {n_days} days - {n_options} options")


    # estimate iv using all info
    res = sub_df.apply(lambda x: compute_iv_eu(option_type=x['option_type'], S=x['underlying_price'], K=x['strike'], T=x['time2mature'], r=x['rf_rate'], q=x['dividend_yield'], option_price=x['option_price']), axis=1)

    res = [v for v in res]
    res = pd.DataFrame(res, columns=['iv', 'error'])
    sub_df['iv_eu'] = res['iv'].values
    sub_df['error_eu'] = res['error'].values

    run_time = round((time.time()-start)/60, 1)
    print(f"=> {run_time} mins for computing iv of {n_options} EU options")
    # save data to drive
    sub_df.to_csv(f'/content/drive/My Drive/data_implied_volatility/results/EU_v2/EUoptions_{stock}_{_time.year}-{_time.month}.csv', index=False)
```

```
1/18: AAPL - 1/6 2013-01-02 to 2013-01-31: 21 days - 59178 options
=> 9.9 mins for computing iv of 59178 EU options
1/18: AAPL - 2/6 2013-02-01 to 2013-02-28: 19 days - 49320 options
=> 8.4 mins for computing iv of 49320 EU options
1/18: AAPL - 3/6 2013-03-01 to 2013-03-28: 20 days - 48242 options
=> 8.2 mins for computing iv of 48242 EU options
1/18: AAPL - 4/6 2013-04-01 to 2013-04-30: 17 days - 39342 options
=> 6.6 mins for computing iv of 39342 EU options
1/18: AAPL - 5/6 2013-05-01 to 2013-05-31: 22 days - 53358 options
=> 8.9 mins for computing iv of 53358 EU options
1/18: AAPL - 6/6 2013-06-03 to 2013-06-11: 6 days - 14064 options
=> 2.3 mins for computing iv of 14064 EU options
2/18: CRM - 1/1 2013-01-02 to 2013-06-11: 105 days - 58086 options
=> 10.5 mins for computing iv of 58086 EU options
3/18: CSCO - 1/1 2013-01-02 to 2013-06-11: 105 days - 34792 options
=> 6.0 mins for computing iv of 34792 EU options
4/18: CVX - 1/1 2013-01-02 to 2013-06-11: 105 days - 32526 options
=> 5.3 mins for computing iv of 32526 EU options
5/18: DIS - 1/1 2013-01-02 to 2013-06-11: 105 days - 30224 options
=> 5.1 mins for computing iv of 30224 EU options
```

Then I load all the results:

```
[ ] # ## load the saved calculated data from drive
    # link = "https://drive.google.com/file/d/1Kj91npjrfTTbyJQIvQg8A_HYyrckrXSZ/view?usp=share_link"
    # result_links = download_data_from_link(link)
    # result_links
```

	name	URL	stock
0	EUoptions_WMT_2013-6.csv	https://drive.google.com/file/d/10JWliImWhvGvI...	WMT
1	EUoptions_V_2013-6.csv	https://drive.google.com/file/d/10HrTaSmw-Imcr...	V
2	EUoptions_UNH_2013-6.csv	https://drive.google.com/file/d/10Exdb664e3aE_...	UNH
3	EUoptions_PG_2013-6.csv	https://drive.google.com/file/d/10BK7BkYgEDi-v...	PG
4	EUoptions_NKE_2013-6.csv	https://drive.google.com/file/d/107DJwVP83N2G5...	NKE
5	EUoptions_MSFT_2013-6.csv	https://drive.google.com/file/d/103a_Hf3DbdDAA...	MSFT
6	EUoptions_MRK_2013-6.csv	https://drive.google.com/file/d/101yUB4vCKH-H8...	MRK
7	EUoptions_MCD_2013-6.csv	https://drive.google.com/file/d/101dxMiS_v9XXZ...	MCD
8	EUoptions_KO_2013-6.csv	https://drive.google.com/file/d/1-suAldmx9LKd4...	KO
9	EUoptions_JPM_2013-6.csv	https://drive.google.com/file/d/1-rizuZhDq5wTK...	JPM
10	EUoptions_JNJ_2013-6.csv	https://drive.google.com/file/d/1-r4nRg2RccBYx...	JNJ
11	EUoptions_INTC_2013-6.csv	https://drive.google.com/file/d/1-qOetOc4yRDH_...	INTC
12	EUoptions_HD_2013-6.csv	https://drive.google.com/file/d/1-grqZn6fWvC0C...	HD
13	EUoptions_DIS_2013-6.csv	https://drive.google.com/file/d/1-a_XxUWEJf_pO...	DIS
14	EUoptions_CVX_2013-6.csv	https://drive.google.com/file/d/1-_Ea9XSq0DktS...	CVX
15	EUoptions_CSCO_2013-6.csv	https://drive.google.com/file/d/1-YJcao5flUave...	CSCO
16	EUoptions_CRM_2013-6.csv	https://drive.google.com/file/d/1-IrNsAzRuKPt6...	CRM
17	EUoptions_AAPL_2013-6.csv	https://drive.google.com/file/d/1-HXiQ-9JcG5JK...	AAPL
18	EUoptions_AAPL_2013-5.csv	https://drive.google.com/file/d/1-DdC-ZThYQfCD...	AAPL
19	EUoptions_AAPL_2013-4.csv	https://drive.google.com/file/d/1-5_CesngMcNtJ...	AAPL
20	EUoptions_AAPL_2013-3.csv	https://drive.google.com/file/d/1-5LJrIECbmW-R...	AAPL
21	EUoptions_AAPL_2013-2.csv	https://drive.google.com/file/d/1-5-HtfXN0yI5G...	AAPL
22	EUoptions_AAPL_2013-1.csv	https://drive.google.com/file/d/1--obgS5FHsF0x...	AAPL

After that, I filter 18 stocks that I need and merge into one file:

```
[ ] STOCKS_LIST = ['AAPL', 'CRM', 'CSCO', 'CVX', 'DIS', 'HD', 'INTC', 'JNJ', 'JPM',
                   'KO', 'MCD', 'MRK', 'MSFT', 'NKE', 'PG', 'UNH', 'V', 'WMT']
```

```
for _c, stock in enumerate(STOCKS_LIST):
  print(f"{_c+1}/{len(STOCKS_LIST)} {stock}")
  _result_links = result_links.loc[result_links['stock']==stock]['URL'].values
  for c, _link in enumerate(_result_links):
    __df = download_data_from_link(_link)
    __df['ticker'] = stock
    if c == 0:
      _df = __df.copy()
    else:
      _df = pd.concat([_df, __df], ignore_index=True)
  if _c == 0:
    df = _df
  else:
    df = pd.concat([df, _df], ignore_index=True)

df
```

```
1/18 AAPL
2/18 CRM
3/18 CSCO
4/18 CVX
5/18 DIS
6/18 HD
7/18 INTC
8/18 JNJ
9/18 JPM
10/18 KO
11/18 MCD
12/18 MRK
13/18 MSFT
14/18 NKE
15/18 PG
16/18 UNH
17/18 V
18/18 WMT
```

	option_type	underlying_price	strike	time2mature	rf_rate	dividend_yield	option_price	quote_date	iv_eu	error_eu	ticker
0	call	450.720009	330.0	0.010959	0.0014	0.024208	119.825	2013-06-03	3.712603e-01	0.780514	AAPL
1	put	450.720009	330.0	0.010959	0.0014	0.024208	0.005	2013-06-03	9.068806e-01	0.000002	AAPL
2	call	450.720009	340.0	0.010959	0.0014	0.024208	109.850	2013-06-03	6.217249e-15	0.755668	AAPL
3	put	450.720009	340.0	0.010959	0.0014	0.024208	0.005	2013-06-03	8.252447e-01	0.000002	AAPL
4	call	450.720009	350.0	0.010959	0.0014	0.024208	100.475	2013-06-03	1.698519e-01	0.130821	AAPL
...	...	...	...	...	...	...	...	...	...	...	...
881248	put	75.250000	105.0	1.602740	0.0014	0.029368	32.125	2013-06-11	9.858447e-02	0.846645	WMT
881249	call	75.250000	110.0	1.602740	0.0014	0.029368	0.195	2013-06-11	1.784709e-01	0.000024	WMT
881250	put	75.250000	110.0	1.602740	0.0014	0.029368	36.800	2013-06-11	1.052865e-01	1.161779	WMT
881251	call	75.250000	115.0	1.602740	0.0014	0.029368	0.125	2013-06-11	1.806817e-01	0.000135	WMT
881252	put	75.250000	115.0	1.602740	0.0014	0.029368	41.750	2013-06-11	1.113141e-01	1.201262	WMT

881253 rows × 11 columns

Then I remove duplicate:

```
[ ] # remove duplicate
    df.drop_duplicates(inplace=True)
    df
```

	option_type	underlying_price	strike	time2mature	rf_rate	dividend_yield	option_price	quote_date	iv_eu	error_eu	ticker
0	call	450.720009	330.0	0.010959	0.0014	0.024208	119.825	2013-06-03	3.712603e-01	0.780514	AAPL
1	put	450.720009	330.0	0.010959	0.0014	0.024208	0.005	2013-06-03	9.068806e-01	0.000002	AAPL
2	call	450.720009	340.0	0.010959	0.0014	0.024208	109.850	2013-06-03	6.217249e-15	0.755668	AAPL
3	put	450.720009	340.0	0.010959	0.0014	0.024208	0.005	2013-06-03	8.252447e-01	0.000002	AAPL
4	call	450.720009	350.0	0.010959	0.0014	0.024208	100.475	2013-06-03	1.698519e-01	0.130821	AAPL
...	...	...	...	...	...	...	...	...	...	...	...
881248	put	75.250000	105.0	1.602740	0.0014	0.029368	32.125	2013-06-11	9.858447e-02	0.846645	WMT
881249	call	75.250000	110.0	1.602740	0.0014	0.029368	0.195	2013-06-11	1.784709e-01	0.000024	WMT
881250	put	75.250000	110.0	1.602740	0.0014	0.029368	36.800	2013-06-11	1.052865e-01	1.161779	WMT
881251	call	75.250000	115.0	1.602740	0.0014	0.029368	0.125	2013-06-11	1.806817e-01	0.000135	WMT
881252	put	75.250000	115.0	1.602740	0.0014	0.029368	41.750	2013-06-11	1.113141e-01	1.201262	WMT

881253 rows × 11 columns

```
[ ] df.columns
```

```
Index(['option_type', 'underlying_price', 'strike', 'time2mature', 'rf_rate',
       'dividend_yield', 'option_price', 'quote_date', 'iv_eu', 'error_eu',
       'ticker'],
      dtype='object')
```

I reorder the file and save the data into Google Drive to use in the next step:

```
# reorder columns
df = df.reindex(columns=['ticker', 'option_type', 'underlying_price', 'strike', 'time2mature', 'rf_rate',
       'dividend_yield', 'option_price', 'quote_date', 'iv_eu', 'error_eu'])
df
```

	ticker	option_type	underlying_price	strike	time2mature	rf_rate	dividend_yield	option_price	quote_date	iv_eu	error_eu
0	AAPL	call	450.720009	330.0	0.010959	0.0014	0.024208	119.825	2013-06-03	3.712603e-01	0.780514
1	AAPL	put	450.720009	330.0	0.010959	0.0014	0.024208	0.005	2013-06-03	9.068806e-01	0.000002
2	AAPL	call	450.720009	340.0	0.010959	0.0014	0.024208	109.850	2013-06-03	6.217249e-15	0.755668
3	AAPL	put	450.720009	340.0	0.010959	0.0014	0.024208	0.005	2013-06-03	8.252447e-01	0.000002
4	AAPL	call	450.720009	350.0	0.010959	0.0014	0.024208	100.475	2013-06-03	1.698519e-01	0.130821
...	...	...	...	...	...	...	...	...	...	...	...
881248	WMT	put	75.250000	105.0	1.602740	0.0014	0.029368	32.125	2013-06-11	9.858447e-02	0.846645
881249	WMT	call	75.250000	110.0	1.602740	0.0014	0.029368	0.195	2013-06-11	1.784709e-01	0.000024
881250	WMT	put	75.250000	110.0	1.602740	0.0014	0.029368	36.800	2013-06-11	1.052865e-01	1.161779
881251	WMT	call	75.250000	115.0	1.602740	0.0014	0.029368	0.125	2013-06-11	1.806817e-01	0.000135
881252	WMT	put	75.250000	115.0	1.602740	0.0014	0.029368	41.750	2013-06-11	1.113141e-01	1.201262

881253 rows × 11 columns

```
# # save data
# df.to_csv(f'/content/drive/My Drive/data_implied_volatility/results/EU_v2/result_EUoptions.csv', index=False)
```

To run the calculation of implied volatility, I import the results that I just saved:

```
# load saved data
link = "https://drive.google.com/file/d/1wI2WD6fmAkUE0hI4cSFk5kY5l_bLSxCr/view?usp=share_link"
df = download_data_from_link(link)
df['quote_date'] = pd.to_datetime(df['quote_date'])
STOCKS_LIST = list(df['ticker'].unique())
df
```

	ticker	option_type	underlying_price	strike	time2mature	rf_rate	dividend_yield	option_price	quote_date	iv_eu	error_eu
0	AAPL	call	450.720009	330.0	0.010959	0.0014	0.024208	119.825	2013-06-03	3.712603e-01	0.780514
1	AAPL	put	450.720009	330.0	0.010959	0.0014	0.024208	0.005	2013-06-03	9.068806e-01	0.000002
2	AAPL	call	450.720009	340.0	0.010959	0.0014	0.024208	109.850	2013-06-03	6.217249e-15	0.755668
3	AAPL	put	450.720009	340.0	0.010959	0.0014	0.024208	0.005	2013-06-03	8.252447e-01	0.000002
4	AAPL	call	450.720009	350.0	0.010959	0.0014	0.024208	100.475	2013-06-03	1.698519e-01	0.130821
...	...	...	...	...	...	...	...	...	...	...	...
881248	WMT	put	75.250000	105.0	1.602740	0.0014	0.029368	32.125	2013-06-11	9.858447e-02	0.846645
881249	WMT	call	75.250000	110.0	1.602740	0.0014	0.029368	0.195	2013-06-11	1.784709e-01	0.000024
881250	WMT	put	75.250000	110.0	1.602740	0.0014	0.029368	36.800	2013-06-11	1.052865e-01	1.161779
881251	WMT	call	75.250000	115.0	1.602740	0.0014	0.029368	0.125	2013-06-11	1.806817e-01	0.000135
881252	WMT	put	75.250000	115.0	1.602740	0.0014	0.029368	41.750	2013-06-11	1.113141e-01	1.201262

881253 rows × 11 columns

In order to increase the accuracy of training model, I remove the predictions that have high error rates:

```
# remove all iv values with error_rate > 5%
df["error_eu"] = 100*(df["error_eu"]/df["option_price"])
df['iv_eu_adj'] = df.apply(lambda x: x["iv_eu"] if x["error_eu"]<5 else np.nan, axis=1)
# remove all iv with value < 0.10
df.loc[(df["option_type"]=="put"),'iv_eu_adj'] = df.loc[(df["option_type"]=="put"), "iv_eu_adj"].apply(lambda x: x if x>0.20 else np.nan)
# df.loc[(df["option_type"]=="put") & (df["time2mature"]<0.2) & (df["strike"]>500), "iv_eu_adj"] = df.loc[(df["option_type"]=="put") & (df["time2mature"]<0.2) & (df["strike"]>500)].apply(lambda x: x["iv_eu_adj"] if (x["iv_eu_adj"])>=0.45)  else np.nan, axis=1)
df
```

	ticker	option_type	underlying_price	strike	time2mature	rf_rate	dividend_yield	option_price	quote_date	iv_eu	error_eu	iv_eu_adj
0	AAPL	call	450.720009	330.0	0.010959	0.0014	0.024208	119.825	2013-06-03	3.712603e-01	0.651378	3.712603e-01
1	AAPL	put	450.720009	330.0	0.010959	0.0014	0.024208	0.005	2013-06-03	9.068806e-01	0.030680	9.068806e-01
2	AAPL	call	450.720009	340.0	0.010959	0.0014	0.024208	109.850	2013-06-03	6.217249e-15	0.687909	6.217249e-15
3	AAPL	put	450.720009	340.0	0.010959	0.0014	0.024208	0.005	2013-06-03	8.252447e-01	0.043152	8.252447e-01
4	AAPL	call	450.720009	350.0	0.010959	0.0014	0.024208	100.475	2013-06-03	1.698519e-01	0.130203	1.698519e-01
...	...	...	...	...	...	...	...	...	...	...	...	...
881248	WMT	put	75.250000	105.0	1.602740	0.0014	0.029368	32.125	2013-06-11	9.858447e-02	2.635471	NaN
881249	WMT	call	75.250000	110.0	1.602740	0.0014	0.029368	0.195	2013-06-11	1.784709e-01	0.012285	1.784709e-01
881250	WMT	put	75.250000	110.0	1.602740	0.0014	0.029368	36.800	2013-06-11	1.052865e-01	3.157008	NaN
881251	WMT	call	75.250000	115.0	1.602740	0.0014	0.029368	0.125	2013-06-11	1.806817e-01	0.107659	1.806817e-01
881252	WMT	put	75.250000	115.0	1.602740	0.0014	0.029368	41.750	2013-06-11	1.113141e-01	2.877275	NaN

881253 rows × 12 columns

Then move to the visualization part, I import the data save in above part:

Vizualize Implied Volatility

```
[ ] # # load the saved data
    # import data from a preprocessed, saved file
    link = "https://drive.google.com/file/d/11R-0-URJZDCH5GscFjs6CWyGsbwxxpJj/view?usp=share_link"
    df = download_data_from_link(link)
    df['quote_date'] = pd.to_datetime(df['quote_date'])
    STOCKS_LIST = list(df['ticker'].unique())
    df
```

	ticker	option_type	underlying_price	strike	time2mature	rf_rate	dividend_yield	option_price	quote_date	iv_eu	error_eu	iv_eu_adj
0	AAPL	call	450.720009	330.0	0.010959	0.0014	0.024208	119.825	2013-06-03	3.712603e-01	0.651378	3.712603e-01
1	AAPL	call	450.720009	340.0	0.010959	0.0014	0.024208	109.850	2013-06-03	6.217249e-15	0.687909	6.217249e-15
2	AAPL	call	450.720009	350.0	0.010959	0.0014	0.024208	100.475	2013-06-03	1.698519e-01	0.130203	1.698519e-01
3	AAPL	call	450.720009	355.0	0.010959	0.0014	0.024208	94.850	2013-06-03	1.953993e-14	0.796940	1.953993e-14
4	AAPL	call	450.720009	360.0	0.010959	0.0014	0.024208	90.300	2013-06-03	1.601138e-01	0.338842	1.601138e-01
...	...	...	...	...	...	...	...	...	...	...	...	...
881248	WMT	put	75.250000	95.0	1.602740	0.0014	0.029368	22.725	2013-06-11	8.233676e-02	1.145395	NaN
881249	WMT	put	75.250000	100.0	1.602740	0.0014	0.029368	27.325	2013-06-11	9.102681e-02	2.397453	NaN
881250	WMT	put	75.250000	105.0	1.602740	0.0014	0.029368	32.125	2013-06-11	9.858447e-02	2.635471	NaN
881251	WMT	put	75.250000	110.0	1.602740	0.0014	0.029368	36.800	2013-06-11	1.052865e-01	3.157008	NaN
881252	WMT	put	75.250000	115.0	1.602740	0.0014	0.029368	41.750	2013-06-11	1.113141e-01	2.877275	NaN

881253 rows × 12 columns

Then import Plotly package and setup for the visualization:

```
[ ] from plotly.subplots import make_subplots
    import plotly.graph_objects as go

    def _visualize_two_iv_surfaces(iv_matrix1, iv_matrix2, ticker):
        fig = make_subplots(
            rows=1, cols=2,
            column_widths=[0.5, 0.5],
           specs=[[{"type": "surface"}, {"type": "surface"}]],
           subplot_titles=("EU Call", "EU Put"))
        z1 = iv_matrix1.values
        x1, y1 = iv_matrix1.index.values, iv_matrix1.columns.values

        z2 = iv_matrix2.values
        x2, y2 = iv_matrix2.index.values, iv_matrix2.columns.values

        fig.add_trace(
            go.Surface(z=z1, x=x1, y=y1),
            row=1, col=1
        )

        fig.add_trace(
            go.Surface(z=z2, x=x2, y=y2),
            row=1, col=2
        )

        # Update xaxis properties
        fig.update_scenes(row=1, col=1, xaxis = dict(title='Time to Maturity', titlefont=dict(size=9), tickfont=dict(size=9)))
        fig.update_scenes(row=1, col=2, xaxis = dict(title='Time to Maturity', titlefont=dict(size=9), tickfont=dict(size=9)))

        # Update yaxis properties
        fig.update_scenes(row=1, col=1, yaxis = dict(title='Strike', titlefont=dict(size=9), tickfont=dict(size=9)))
        fig.update_scenes(row=1, col=2, yaxis = dict(title='Strike', titlefont=dict(size=9), tickfont=dict(size=9)))

        # Update zaxis properties
        fig.update_scenes(row=1, col=1, zaxis = dict(title='Implied Volatility', titlefont=dict(size=9), tickfont=dict(size=9)))
        fig.update_scenes(row=1, col=2, zaxis = dict(title='Implied Volatility', titlefont=dict(size=9), tickfont=dict(size=9)))

        fig.update_layout(height=550, width=1100, title_text=f"{ticker} - Implied Volatility Surface: Black-Scholes-Merton Model")
        fig.show()

    def _visualize_iv_surface(iv_matrix, ticker, option_type="EU Call"):
        z = iv_matrix.values
        x, y = iv_matrix.index.values, iv_matrix.columns.values
        # x, y = iv_matrix_call.columns, iv_matrix_call.index
        fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])

        # update axis properties
        fig.update_scenes(xaxis = dict(title='Time to Maturity', titlefont=dict(size=10), tickfont=dict(size=10)))
        fig.update_scenes(yaxis = dict(title='Strike', titlefont=dict(size=10), tickfont=dict(size=10)))
        fig.update_scenes(zaxis = dict(title='Implied Volatility', titlefont=dict(size=10), tickfont=dict(size=10)))
        # fig.update_scenes(xaxis_title_text='Time to Maturity (days/365)',  yaxis_title_text='Strike', zaxis_title_text='Implied Volatility')
        fig.update_layout(height=600, width=800, title_text=f'{ticker} - Implied Volatility Surface {option_type}: Black Scholes-Merton Model')
        fig.show()
```

I also need to create a function to calculate and visualize implied volatility in a period of time:

```
[ ] def visualize_iv_surface(ticker, option_type="call,put", start_date="2013-01-02", end_date="2013-06-11", iv_type='iv_eu', iv_df=df, drop_na=True):
      # # create implied volatily matrix
      start_date = pd.to_datetime(start_date)
      end_date = pd.to_datetime(end_date)

      if option_type == "call,put":
        iv_call_df = iv_df.loc[(iv_df["option_type"]=="call")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]
        iv_put_df = iv_df.loc[(iv_df["option_type"]=="put")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]

        iv_matrix_call = iv_call_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        iv_matrix_put = iv_put_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)

        # drop duplicate if needed
        if drop_na:
          iv_matrix_call = iv_matrix_call.loc[:, iv_matrix_call.isna().mean() <= 0.5]
          iv_matrix_put = iv_matrix_put.loc[:, iv_matrix_put.isna().mean() <= 0.5]
        # visualise 3d iv surface
        _visualize_two_iv_surfaces(iv_matrix1=iv_matrix_call, iv_matrix2=iv_matrix_put, ticker=ticker)
        # _visualize_iv_surface(iv_matrix_call, ticker=ticker, option_type="EU Call")
        # _visualize_iv_surface(iv_matrix_put, ticker=ticker, option_type="EU Put")

      elif option_type == "call":
        iv_call_df = iv_df.loc[(iv_df["option_type"]=="call")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]
        iv_matrix_call = iv_call_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        # visualize 3d iv surface
        _visualize_iv_surface(iv_matrix_call, ticker=ticker, option_type="EU Call")
      elif option_type == "put":
        iv_put_df = iv_df.loc[(iv_df["option_type"]=="put")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]
        iv_matrix_put = iv_put_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        # visualize 3d iv surface
        _visualize_iv_surface(iv_matrix_put, ticker=ticker, option_type="EU Put")
      else:
        print("Call or Put not specified")
        return 0
```

```
[ ] import ipywidgets as widgets
    from ipywidgets import interact, interact_manual
```

```
[ ] iv_type = 'iv_eu'
    iv_df = df
    drop_na = True

    def _visualize_iv_surface(iv_matrix, ticker, option_type="EU Call"):
        z = iv_matrix.values
        x, y = iv_matrix.index.values, iv_matrix.columns.values
        # x, y = iv_matrix_call.columns, iv_matrix_call.index
        fig = go.FigureWidget(data=[go.Surface(z=z, x=x, y=y)])

        # update axis properties
        fig.update_scenes(xaxis = dict(title='Time to Maturity', titlefont=dict(size=10), tickfont=dict(size=10)))
        fig.update_scenes(yaxis = dict(title='Strike', titlefont=dict(size=10), tickfont=dict(size=10)))
        fig.update_scenes(zaxis = dict(title='Implied Volatility', titlefont=dict(size=10), tickfont=dict(size=10)))
        # fig.update_scenes(xaxis_title_text='Time to Maturity (days/365)',  yaxis_title_text='Strike', zaxis_title_text='Implied Volatility')
        fig.update_layout(height=600, width=800, title_text=f'{ticker} - Implied Volatility Surface {option_type}: Black Scholes-Merton Model')
        fig.show()

    def visualize_iv_surface(ticker, option_type="call,put", start_date="2013-01-02", end_date="2013-06-11"):
      # # create implied volatily matrix
      start_date = pd.to_datetime(start_date)
      end_date = pd.to_datetime(end_date)

      if option_type == "call,put":
        iv_call_df = iv_df.loc[(iv_df["option_type"]=="call")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]
        iv_put_df = iv_df.loc[(iv_df["option_type"]=="put")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]

        iv_matrix_call = iv_call_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        iv_matrix_put = iv_put_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)

        # drop duplicate if needed
        if drop_na:
          iv_matrix_call = iv_matrix_call.loc[:, iv_matrix_call.isna().mean() <= 0.5]
          iv_matrix_put = iv_matrix_put.loc[:, iv_matrix_put.isna().mean() <= 0.5]
        # visualise 3d iv surface
        _visualize_two_iv_surfaces(iv_matrix1=iv_matrix_call, iv_matrix2=iv_matrix_put, ticker=ticker)
        # _visualize_iv_surface(iv_matrix_call, ticker=ticker, option_type="EU Call")
        # _visualize_iv_surface(iv_matrix_put, ticker=ticker, option_type="EU Put")

      elif option_type == "call":
        iv_call_df = iv_df.loc[(iv_df["option_type"]=="call")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]
        iv_matrix_call = iv_call_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        # visualize 3d iv surface
        _visualize_iv_surface(iv_matrix_call, ticker=ticker, option_type="EU Call")
      elif option_type == "put":
        iv_put_df = iv_df.loc[(iv_df["option_type"]=="put")&(iv_df["ticker"]==ticker)&(iv_df["quote_date"]>=start_date)&(iv_df["quote_date"]<=end_date)][["time2mature", "strike", iv_type]]
        iv_matrix_put = iv_put_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        # visualize 3d iv surface
        _visualize_iv_surface(iv_matrix_put, ticker=ticker, option_type="EU Put")
      else:
        print("Call or Put not specified")
        return 0


    _ = interact(visualize_iv_surface,
             ticker=widgets.Dropdown(options=STOCKS_LIST,
                                     value='AAPL', description='Ticker:', disabled=False),
             option_type=widgets.Dropdown(options=["call,put", "call", "put"],
                                     value='call', description='Option to Display:', disabled=False),
             start_date=widgets.DatePicker(value=pd.to_datetime('2013-01-02'), description='Start Date:'),
             end_date=widgets.DatePicker(value=pd.to_datetime('2013-06-11'), description='End Date:'))
```

After that, I draw the implied volatility for one day of AAPL:

```
# visualize one day AAPL
visualize_iv_surface(ticker="AAPL", option_type="call,put", start_date="2013-01-02", end_date="2013-01-02", iv_type='iv_eu_adj', iv_df=df, drop_na=True)
```

Then the visualize for 1 week:

```
# visualize one week AAPL
visualize_iv_surface(ticker="AAPL", option_type="call,put", start_date="2013-01-02", end_date="2013-01-09", iv_type='iv_eu_adj', iv_df=df, drop_na=True)
```

For 1 month:

```
# visualize one month AAPL
visualize_iv_surface(ticker="AAPL", option_type="call,put", start_date="2013-01-02", end_date="2013-02-02", iv_type='iv_eu_adj', iv_df=df, drop_na=True)
```

For 6 months:

```
[ ] # visualize 6-month AAPL
    visualize_iv_surface(ticker="AAPL", option_type="call,put", start_date="2013-01-02", end_date="2013-06-30", iv_type='iv_eu_adj', iv_df=df, drop_na=True)
```

3.2.2 Compute and Visualize Implied Volatility for American option

The process for American option is the same as European option. The only difference is the formula used in here is the Binominal function:

```
# Import packages
import numpy as np
import scipy as sq


# Defined functions
def LeisenReimerBinomial(option_type, S, K, T, r, q, sigma, n=30):
    # This functions calculates the implied volatility of American options
    # This code is based on "The complete guide to Option Pricing Formulas" by Espen Gaarder Haug (2007)
    # option_type:
    # "call" Returns the call value
    # "put" Returns the put value
    # S is the share price at time t
    # K is the strike price
    # T is the time to maturity in years (days/365)
    # r is the risk-free interest rate
    # q is the dividend yield
    # sigma is the volatility
    # n determines the step size

    # Start of the code
    # rounds n up tot the nearest odd integer (the function is displayed below the LeisenReimerBinomial function in line x)
    n = round_up_to_odd(n)

    # Creates a list with values from 0 up to n (which will be used to determine to exercise or not)
    n_list = np.arange(0, (n + 1), 1)

    # Checks if the input option is a put or a call, if not it returns an error
    if option_type == 'call':
        z = 1
    elif option_type == 'put':
        z = -1
    else:
        return 'Call or put not defined'

    # d1 and d2 formulas of the Black-Scholes formula for European options
    d1 = (np.log(S / K) + (sigma ** 2 / 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # The Preizer-Pratt inversion method 1
    hd1 = 0.5 + np.sign(d1) * (0.25 - 0.25 * np.exp(-(d1 /
                                                    (n + 1 / 3 + 0.1 / (n + 1))) ** 2 * (n + 1 / 6))) ** 0.5

    # The Preizer-Prat inversion method 2
    hd2 = 0.5 + np.sign(d2) * (0.25 - 0.25 * np.exp(-(d2 /
                                                    (n + 1 / 3 + 0.1 / (n + 1))) ** 2 * (n + 1 / 6))) ** 0.5

    # Calculates the stepsize in years
    dt = T / n
```

```python
    # The Preizer-Pratt inversion method 1
    hd1 = 0.5 + np.sign(d1) * (0.25 - 0.25 * np.exp(-(d1 /
                                                   (n + 1 / 3 + 0.1 / (n + 1))) ** 2 * (n + 1 / 6))) ** 0.5

    # The Preizer-Prat inversion method 2
    hd2 = 0.5 + np.sign(d2) * (0.25 - 0.25 * np.exp(-(d2 /
                                                   (n + 1 / 3 + 0.1 / (n + 1))) ** 2 * (n + 1 / 6))) ** 0.5

    # Calculates the stepsize in years
    dt = T / n

    # The up and down factors
    u = hd1 / hd2
    d = 1/u
    df = np.exp((q-r) * dt)
    # The probability of going up in a risk-neutral world
    # p = hd2
    p = (np.exp((r-q)*dt) - d)/(u-d)

    # Creates the most right column of the three
    max_pay_off_list = []
    for i in n_list:
        i = i.astype('int')
        T = (n - i)
        max_pay_off = np.maximum(0, z * (S * u ** i * d ** T - K))
        max_pay_off_list.append(max_pay_off)

    # The binominal tree
    for j in np.arange(n - 1, 0 - 1, -1):
        for i in np.arange(0, j + 1, 1):
            i = i.astype(int)  # Need to be converted to a integer
            t = (j - i)
            max_pay_off_list[i] = np.maximum((z * (S * u ** i * d ** t - K)),
                                             (p * max_pay_off_list[i + 1] + (1 - p) * max_pay_off_list[i]) * df)
    price = max_pay_off_list[0]

    return price


def round_up_to_odd(n):
    # This function returns a number rounded up to the nearest odd integer
    # For example when n = 100, the function returns 101
    return np.ceil(n) // 2 * 2 + 1


def compute_iv_am(option_type, S, K, T, r, q, option_price):
    # This function is the implied volatility solver which makes use of Nelder-Mead simplex algorithm
    # The paramaters are the same as described as in LeisenReimerBinomial function, except for Option_Value
    # option_price is the parameter of the option value (for which you want to calculate the Implied Volatility)

    def objective_function(iv):
      # This is the objection function for Nelder-Mead simplex algorithm
      result = (option_price - LeisenReimerBinomial(option_type, S, K, T, r, q, iv))**2
      return result
    # initialize starting point for faster conversion
    start_point = np.sqrt(2*abs(((np.log(S/K)+r*T)/T)))
    res = sq.optimize.fmin(objective_function, start_point, full_output=True, ftol=0.01, maxiter=30, disp=0)
    iv_result = res[0][0]
    error = 100*(np.sqrt(res[1])/option_price)
    if (iv_result<=0) or (iv_result>1):
      res = sq.optimize.fminbound(objective_function, 0.01, 1, full_output=True, xtol = 0.01, maxfun=30, disp=0)
      iv_result = res[0]
      error = 100*(np.sqrt(res[1])/option_price)

    return [iv_result, error]
```

3.2.3 Predict Implied Volatility using Machine Learning for European option

The step of import package and import dataset will be the same as the above parts. Then I need to prepare necessary variables for Dumas fomular:

```python
## preparing data for Dumas
# df_dm_call = pd.DataFrame()

# ticker = "INTC"
iv_type = 'iv_eu_adj'

all_df_dm = pd.DataFrame()

option_types = ["call", "put"]
# option_types = ["call"]
for option_type in option_types:
  df_dm = df.loc[df['option_type']==option_type][["ticker", iv_type, "strike", "time2mature"]]
  all_df_dm_cp = pd.DataFrame()
  for ticker in STOCKS_LIST:
    _strike = np.sort(df_dm.loc[df_dm["ticker"]==ticker]["strike"].unique())
    _ttm = np.sort(df_dm.loc[df_dm["ticker"]==ticker]["time2mature"].unique())

    df_dm_cp = pd.DataFrame()
    for k in _strike:
      if len(df_dm_cp) == 0:
        df_dm_cp["T"] = _ttm
        df_dm_cp["X"] = k
      else:
        _df_dm_cp = pd.DataFrame()
        _df_dm_cp["T"] = _ttm
        _df_dm_cp["X"] = k

        df_dm_cp = pd.concat([df_dm_cp, _df_dm_cp], axis=0, ignore_index=True)

    df_dm_cp['ticker'] = ticker

    if len(all_df_dm_cp) == 0:
      all_df_dm_cp = df_dm_cp.copy()
    else:
      all_df_dm_cp = pd.concat([all_df_dm_cp, df_dm_cp], axis=0, ignore_index=True)

  # add implied volatility
  _iv_eu = df.loc[df['option_type']==option_type][["ticker", "strike", "time2mature", iv_type]]
  # print(_iv_eu[_iv_eu['ticker']=="AAPL"])
  # print(all_df_dm_cp[all_df_dm_cp['ticker']=="AAPL"])
  all_df_dm_cp = all_df_dm_cp.merge(_iv_eu, left_on=['ticker', 'X', 'T'], right_on=['ticker', 'strike', 'time2mature'], how='left')
  all_df_dm_cp.drop(columns=["strike", 'time2mature'], inplace=True)

  # compute others variables
  all_df_dm_cp["X^2"] = all_df_dm_cp["X"]**2
  all_df_dm_cp["T^2"] = all_df_dm_cp["T"]**2
  all_df_dm_cp["XT"] = all_df_dm_cp["X"] * all_df_dm_cp["T"]

  # add option type
  if option_type == "call":
    all_df_dm_cp['option_type'] = 1
  else:
    all_df_dm_cp['option_type'] = 0

  # concat call and put df
  if len(all_df_dm) == 0:
    all_df_dm = all_df_dm_cp.copy()
  else:
    all_df_dm = pd.concat([all_df_dm, all_df_dm_cp], axis=0, ignore_index=True)

  # move column ticker to the end
  all_df_dm = all_df_dm.reindex(columns=[iv_type, 'T', 'X', 'X^2', 'T^2', 'XT', 'option_type', 'ticker'])
```

Then in order to run model, I will need to change sticker to one_hot binary variables:

```python
# convert ticker to one_hot binary variable using one_hot encoding
dm_encoder=ce.OneHotEncoder(cols='ticker',handle_unknown='return_nan',return_df=True,use_cat_names=True)
dm_encoded = dm_encoder.fit_transform(all_df_dm)
dm_encoded
```

	iv_eu_adj	T	X	X^2	T^2	XT	option_type	ticker_AAPL	ticker_CRM	ticker_CSCO	...	ticker_JPM	ticker_KO	ticker_MCD	ticker_MRK	ticker_MSFT	ticker_NKE	ticker_PG	ticker_UNH	ticker_V	ticker_WMT
0	NaN	0.000000	135.0	18225.0	0.000000	0.000000	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.999608	0.002740	135.0	18225.0	0.000008	0.369863	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.999608	0.005479	135.0	18225.0	0.000030	0.739726	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.999608	0.008219	135.0	18225.0	0.000068	1.109589	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.999608	0.010959	135.0	18225.0	0.000120	1.479452	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1268020	NaN	2.024658	115.0	13225.0	4.099238	232.835616	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1268021	NaN	2.027397	115.0	13225.0	4.110340	233.150685	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1268022	NaN	2.035616	115.0	13225.0	4.143734	234.095890	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1268023	NaN	2.038356	115.0	13225.0	4.154896	234.410959	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1268024	NaN	2.041096	115.0	13225.0	4.166072	234.726027	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

1268025 rows × 25 columns

Next step, I use Linear Regression and I use Sklearn package to run it:

```
[ ] ## Fit linear regression model
    from sklearn import linear_model
    from sklearn.metrics import mean_squared_error, r2_score
    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split
```

```
# drop nan
_dm_encoded = dm_encoded.dropna()

# run Regression using sklearn
X = _dm_encoded.drop(columns=[iv_type])
y = _dm_encoded[iv_type]
# X = df_dm_scaled.drop(columns=['iv'])
# y = df_dm_scaled['iv']

# split train/test set with .75/.25 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# # drop the last column of ticker to prevent co-linearity
# last_col_name = X.iloc[:,-1].name
# last_col_val_Xtrain = X_train.iloc[:,-1].values
# last_col_val_Xtest = X_test.iloc[:,-1].values
# X_train.drop(columns=[last_col_name], inplace=True)
# X_test.drop(columns=[last_col_name], inplace=True)

lr = linear_model.LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

coef = [i for i in lr.coef_]
intercept = lr.intercept_

lr_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
lr_r2 = r2_score(y_test, y_pred)

# The intercept
print("Intercept: ", intercept)
# The coefficients[]
print("Coefficients: \n", coef)

# The mean squared error
print("Root Mean squared error: RMSE = %.4f" % lr_rmse)
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: R2 = %.2f" % lr_r2)
```

```
Intercept:  0.624158311218606
Coefficients:
 [-0.36628424485681, -0.0022635509632729983, 1.9551074095740078e-06, 0.15738882617429803, -8.651835486358119e-05, 0.006279190703969242, 0.46826701002651216, 0.12584529792940066, -0.0884184638230587, 0.07009002953291585, -0.072537460737879
Root Mean squared error: RMSE = 0.1831
Coefficient of determination: R2 = 0.22
```

Then I also use Statmodel to run again:

```python
import statsmodels.api as sm

#define response variable
y = _dm_encoded[iv_type]

#define predictor variables
x = _dm_encoded.drop(columns=[iv_type])

#add constant to predictor variables
x = sm.add_constant(x)

#fit linear regression model
lr_model = sm.OLS(y, x).fit()

#view model summary
print(lr_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              iv_eu_adj   R-squared:                       0.220
Model:                            OLS   Adj. R-squared:                  0.220
Method:                 Least Squares   F-statistic:                     8723.
Date:                Mon, 14 Aug 2023   Prob (F-statistic):               0.00
Time:                        09:08:10   Log-Likelihood:             1.9799e+05
No. Observations:              713213   AIC:                        -3.959e+05
Df Residuals:                  713189   BIC:                        -3.957e+05
Df Model:                          23
Covariance Type:            nonrobust
================================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          0.5912      0.001    828.496      0.000       0.590       0.593
T             -0.3671      0.001   -266.326      0.000      -0.370      -0.364
X             -0.0023      8e-06   -282.675      0.000      -0.002      -0.002
X^2         1.952e-06   6.76e-09    288.581      0.000    1.94e-06    1.97e-06
T^2            0.1576      0.001    223.378      0.000       0.156       0.159
XT         -8.569e-05   1.63e-06    -52.712      0.000   -8.89e-05   -8.25e-05
option_type    0.0064      0.000     14.436      0.000       0.006       0.007
ticker_AAPL    0.5003      0.002    285.908      0.000       0.497       0.504
ticker_CRM     0.1582      0.001    173.638      0.000       0.156       0.160
ticker_CSCO   -0.0553      0.001    -50.423      0.000      -0.057      -0.053
ticker_CVX     0.1023      0.001     88.469      0.000       0.100       0.105
ticker_DIS    -0.0387      0.001    -34.973      0.000      -0.041      -0.037
ticker_HD      0.0096      0.001      8.588      0.000       0.007       0.012
ticker_INTC   -0.0489      0.001    -44.075      0.000      -0.051      -0.047
ticker_JNJ     0.0019      0.001      1.505      0.132      -0.001       0.004
ticker_JPM    -0.0043      0.001     -5.118      0.000      -0.006      -0.003
ticker_KO     -0.0653      0.001    -58.010      0.000      -0.068      -0.063
ticker_MCD     0.0388      0.001     32.558      0.000       0.036       0.041
ticker_MRK    -0.0376      0.001    -35.069      0.000      -0.040      -0.036
ticker_MSFT   -0.0767      0.001    -75.893      0.000      -0.079      -0.075
ticker_NKE    -0.0132      0.001    -12.396      0.000      -0.015      -0.011
ticker_PG      0.0172      0.001     14.317      0.000       0.015       0.020
ticker_UNH    -0.0281      0.001    -22.484      0.000      -0.031      -0.026
ticker_V       0.1351      0.001    124.204      0.000       0.133       0.137
ticker_WMT    -0.0041      0.001     -3.456      0.001      -0.006      -0.002
==============================================================================
Omnibus:                   103230.277   Durbin-Watson:                   0.800
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           186234.428
Skew:                           0.940   Prob(JB):                         0.00
Kurtosis:                       4.652   Cond. No.                     7.91e+19
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.05e-24. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Then, I use model to predict:

```
def predict_iv_dm(ticker, option_type, strike, ttm, model=lr, model_name = 'lr'):
  if option_type == 'call':
    option_type = 1
  else:
    option_type = 0

  iv = np.nan
  X = pd.DataFrame(np.array([[iv, ttm, strike, strike**2, ttm**2, strike*ttm, option_type, ticker]]), columns=[iv_type, 'T', 'X', 'X^2', 'T^2', 'XT', 'option_type', 'ticker'])
  X_transformed = dm_encoder.transform(X)
  X_transformed.drop(columns=[iv_type], inplace=True)

  # if model_name == 'lr':
  #   # drop the last column
  #   X_transformed.drop(columns=[last_col_name], inplace=True)

  iv_pred = model.predict(X_transformed)
  return max(0.01, min(1, iv_pred[0]))

print("Predicted value:")
print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 200, ttm = 0.2, model=lr))
print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 1050, ttm = 0.2, model=lr))

print("\nActual value:")
print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==200)&(all_df_dm['T']==0.2)][iv_type].values[0])
print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==1050)&(all_df_dm['T']==0.2)][iv_type].values[0])
```

```
Predicted value:
0.6537765855585147
0.7923517691220407

Actual value:
0.999607542093466
0.5715448994044049
```

```
df_dm = dm_encoder.inverse_transform(dm_encoded)

# add predicted values
# X_transformed = dm_encoded.drop(columns=[last_col_name])
# iv_eu_pred_lr = lr.predict(X_transformed.drop(columns=[iv_type]).values)
iv_eu_pred_lr = lr.predict(dm_encoded.drop(columns=[iv_type]).values)
df_dm.insert(loc=1, column='iv_eu_pred_lr', value=iv_eu_pred_lr)

# rename column
df_dm.rename(columns={"T": "time2mature", "X": "strike"}, inplace=True)

# remap option_type
df_dm["option_type"] = df_dm["option_type"].apply(lambda x: "call" if x==1 else "put")

df_dm
```

	iv_eu_adj	iv_eu_pred_lr	time2mature	strike	X^2	T^2	XT	option_type	ticker
0	NaN	0.828757	0.000000	135.0	18225.0	0.000000	0.000000	call	AAPL
1	0.999608	0.827723	0.002740	135.0	18225.0	0.000008	0.369863	call	AAPL
2	0.999608	0.826691	0.005479	135.0	18225.0	0.000030	0.739726	call	AAPL
3	0.999608	0.825661	0.008219	135.0	18225.0	0.000068	1.109589	call	AAPL
4	0.999608	0.824634	0.010959	135.0	18225.0	0.000120	1.479452	call	AAPL
...	...	...	...	...	...	...	...	...	...
1268020	NaN	0.234809	2.024658	115.0	13225.0	4.099238	232.835616	put	WMT
1268021	NaN	0.235526	2.027397	115.0	13225.0	4.110340	233.150685	put	WMT
1268022	NaN	0.237689	2.035616	115.0	13225.0	4.143734	234.095890	put	WMT
1268023	NaN	0.238415	2.038356	115.0	13225.0	4.154896	234.410959	put	WMT
1268024	NaN	0.239144	2.041096	115.0	13225.0	4.166072	234.726027	put	WMT

1268025 rows × 9 columns

Then I save the results above to Google Drive:

```
[ ] import os
    # # make a new folder in drive
    if os.path.exists('/content/drive/My Drive/data_implied_volatility/models') == False:
      os.makedirs('/content/drive/My Drive/data_implied_volatility/models', exist_ok=True)
```

Then I start to create a matrix and in that matrix, divide Strike and Time2Mature into equal interval:

```python
def divide_space(df):
  # divide an input df into equal interval of Strike and Time2Mature
  df_dm_space = pd.DataFrame()

  option_types = ["call", "put"]
  # option_types = ["call"]
  for option_type in option_types:
    _df_dm = df.loc[df['option_type']==option_type][["ticker", iv_type, "strike", "time2mature"]]
    df_dm_cp = pd.DataFrame()
    for ticker in STOCKS_LIST:
      _strike = np.sort(df_dm.loc[df_dm["ticker"]==ticker]["strike"].unique())
      _ttm = np.sort(df_dm.loc[df_dm["ticker"]==ticker]["time2mature"].unique())

      n_strike = len(_strike)
      step_strike = (_strike[-1] - _strike[0])/n_strike
      _strike_equal = [(_strike[0] + c*step_strike) for c in range(n_strike) if (_strike[0] + c*step_strike)<= _strike[-1]]

      n_ttm = len(_ttm)
      step_ttm = (_ttm[-1] - _ttm[0])/n_ttm
      _ttm_equal = [(_ttm[0] + c*step_ttm) for c in range(n_ttm) if (_ttm[0] + c*step_ttm)<= _ttm[-1]]

      _df_dm_cp = pd.DataFrame()
      for k in _strike_equal:
        if len(_df_dm_cp) == 0:
          _df_dm_cp["T"] = _ttm_equal
          _df_dm_cp["X"] = k
        else:
          __df_dm_cp = pd.DataFrame()
          __df_dm_cp["T"] = _ttm_equal
          __df_dm_cp["X"] = k

          _df_dm_cp = pd.concat([_df_dm_cp, __df_dm_cp], axis=0, ignore_index=True)

      _df_dm_cp['ticker'] = ticker

      if len(df_dm_cp) == 0:
        df_dm_cp = _df_dm_cp.copy()
      else:
        df_dm_cp = pd.concat([df_dm_cp, _df_dm_cp], axis=0, ignore_index=True)

    # compute others variables
    df_dm_cp["X^2"] = df_dm_cp["X"]**2
    df_dm_cp["T^2"] = df_dm_cp["T"]**2
    df_dm_cp["XT"] = df_dm_cp["X"] * df_dm_cp["T"]

    # add option type
    if option_type == "call":
      df_dm_cp['option_type'] = 1
    else:
      df_dm_cp['option_type'] = 0


    # concat call and put df
    if len(df_dm_space) == 0:
      df_dm_space = df_dm_cp.copy()
    else:
      df_dm_space = pd.concat([df_dm_space, df_dm_cp], axis=0, ignore_index=True)

  # move column ticker to the end
  df_dm_space = df_dm_space.reindex(columns=['T', 'X', 'X^2', 'T^2', 'XT', 'option_type', 'ticker'])

  return df_dm_space
```

Then I predict matrix use fitted model:

predict matrix using fitted model

```
[ ] ## Use fitted model to compute iv of df_dm_space matrix
    # ## decoding ticker
    df_viz = df_dm_space.copy()
    df_dm_space.insert(loc=0, column=iv_type, value=np.nan)
```

```
[ ] df_dm_space_transformed = dm_encoder.transform(df_dm_space)
    df_dm_space_transformed
```

	iv_eu_adj	T	X	X^2	T^2	XT	option_type	ticker_AAPL	ticker_CRM	ticker_CSCO	...	ticker_JPM	ticker_KO	ticker_MCD	ticker_MRK	ticker_MSFT	ticker_NKE	ticker_PG	ticker_UNH
0	NaN	0.000000	135.000000	18225.000000	0.000000	0.000000	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	NaN	0.005274	135.000000	18225.000000	0.000028	0.712010	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	NaN	0.010548	135.000000	18225.000000	0.000111	1.424020	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	NaN	0.015822	135.000000	18225.000000	0.000250	2.136031	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	NaN	0.021097	135.000000	18225.000000	0.000445	2.848041	1	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
827863	NaN	2.013662	112.265625	12603.570557	4.054834	226.065001	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
827864	NaN	2.019149	112.265625	12603.570557	4.076961	226.680982	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
827865	NaN	2.024635	112.265625	12603.570557	4.099149	227.296963	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
827866	NaN	2.030122	112.265625	12603.570557	4.121396	227.912944	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
827867	NaN	2.035609	112.265625	12603.570557	4.143704	228.528925	0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

827868 rows × 25 columns

```
[ ] # add predicted values
    iv_eu_pred_lr = lr.predict(df_dm_space_transformed.drop(columns=[iv_type]).values)
    df_viz.insert(loc=0, column='iv_eu_pred_lr', value=iv_eu_pred_lr)

    # remap option_type
    df_viz["option_type"] = df_viz["option_type"].apply(lambda x: "call" if x==1 else "put")

    # rename columns
    df_viz.rename(columns={"T": "time2mature", "X": "strike"}, inplace=True)
    df_viz
```

	iv_eu_pred_lr	time2mature	strike	X^2	T^2	XT	option_type	ticker
0	0.828757	0.000000	135.000000	18225.000000	0.000000	0.000000	call	AAPL
1	0.826768	0.005274	135.000000	18225.000000	0.000028	0.712010	call	AAPL
2	0.824788	0.010548	135.000000	18225.000000	0.000111	1.424020	call	AAPL
3	0.822816	0.015822	135.000000	18225.000000	0.000250	2.136031	call	AAPL
4	0.820853	0.021097	135.000000	18225.000000	0.000445	2.848041	call	AAPL
...	...	...	...	...	...	...	...	...
827863	0.237408	2.013662	112.265625	12603.570557	4.054834	226.065001	put	WMT

Then I visualize the matrix:

```
[ ] from plotly.subplots import make_subplots
    import plotly.graph_objects as go

    def _visualize_two_iv_surfaces(iv_matrix1, iv_matrix2, ticker, model="Dumas - Linear Regression"):
        fig = make_subplots(
            rows=1, cols=2,
            column_widths=[0.5, 0.5],
           specs=[[{"type": "surface"}, {"type": "surface"}]],
           subplot_titles=("EU Call", "EU Put"))
        z1 = iv_matrix1.values
        x1, y1 = iv_matrix1.index.values, iv_matrix1.columns.values

        z2 = iv_matrix2.values
        x2, y2 = iv_matrix2.index.values, iv_matrix2.columns.values

        fig.add_trace(
            go.Surface(z=z1, x=x1, y=y1),
            row=1, col=1
        )

        fig.add_trace(
            go.Surface(z=z2, x=x2, y=y2),
            row=1, col=2
        )

        # Update xaxis properties
        fig.update_scenes(row=1, col=1, xaxis = dict(title='Time to Maturity', titlefont=dict(size=9), tickfont=dict(size=9)))
        fig.update_scenes(row=1, col=2, xaxis = dict(title='Time to Maturity', titlefont=dict(size=9), tickfont=dict(size=9)))

        # Update yaxis properties
        fig.update_scenes(row=1, col=1, yaxis = dict(title='Strike', titlefont=dict(size=9), tickfont=dict(size=9)))
        fig.update_scenes(row=1, col=2, yaxis = dict(title='Strike', titlefont=dict(size=9), tickfont=dict(size=9)))

        # Update zaxis properties
        fig.update_scenes(row=1, col=1, zaxis = dict(title='Implied Volatility', titlefont=dict(size=9), tickfont=dict(size=9)))
        fig.update_scenes(row=1, col=2, zaxis = dict(title='Implied Volatility', titlefont=dict(size=9), tickfont=dict(size=9)))

        fig.update_layout(height=600, width=1200, title_text=f"{ticker} - Implied Volatility Surface: {model}")
        fig.show()

    def _visualize_iv_surface(iv_matrix, ticker, option_type="EU Call", model="Dumas - Linear Regression"):
        z = iv_matrix.values
        x, y = iv_matrix.index.values, iv_matrix.columns.values
        # x, y = iv_matrix_call.columns, iv_matrix_call.index
        fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])

        # update axis properties
        fig.update_scenes(xaxis = dict(title='Time to Maturity', titlefont=dict(size=10), tickfont=dict(size=10)))
        fig.update_scenes(yaxis = dict(title='Strike', titlefont=dict(size=10), tickfont=dict(size=10)))
        fig.update_scenes(zaxis = dict(title='Implied Volatility', titlefont=dict(size=10), tickfont=dict(size=10)))
        # fig.update_scenes(xaxis_title_text='Time to Maturity (days/365)',  yaxis_title_text='Strike', zaxis_title_text='Implied Volatility')
        fig.update_layout(height=600, width=800, title_text=f'{ticker} - IV Surface - {option_type}: {model}')
        fig.show()
```

```
[ ] def visualize_iv_surface(ticker, option_type="call,put", iv_type='iv_eu', iv_df=df_viz, model="Dumas - Linear Regression", drop_na=True):
      # # create implied volatly matrix
      # start_date = pd.to_datetime(start_date)
      # end_date = pd.to_datetime(end_date)

      if option_type == "call,put":
        iv_call_df = iv_df.loc[(iv_df["option_type"]=="call")&(iv_df["ticker"]==ticker)][["time2mature", "strike", iv_type]]
        iv_put_df = iv_df.loc[(iv_df["option_type"]=="put")&(iv_df["ticker"]==ticker)][["time2mature", "strike", iv_type]]

        iv_matrix_call = iv_call_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        iv_matrix_put = iv_put_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)

        # drop duplicate if needed
        if drop_na:
          iv_matrix_call = iv_matrix_call.loc[:, iv_matrix_call.isna().mean() <= 0.2]
          iv_matrix_put = iv_matrix_put.loc[:, iv_matrix_put.isna().mean() <= 0.2]
        # visualise 3d iv surface
        _visualize_two_iv_surfaces(iv_matrix1=iv_matrix_call, iv_matrix2=iv_matrix_put, ticker=ticker, model=model)
        # _visualize_iv_surface(iv_matrix_call, ticker=ticker, option_type="EU Call")
        # _visualize_iv_surface(iv_matrix_put, ticker=ticker, option_type="EU Put")

      elif option_type == "call":
        iv_call_df = iv_df.loc[(iv_df["option_type"]=="call")&(iv_df["ticker"]==ticker)][["time2mature", "strike", iv_type]]
        iv_matrix_call = iv_call_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        # visualize 3d iv surface
        _visualize_iv_surface(iv_matrix_call, ticker=ticker, option_type="EU Call", model=model)
      elif option_type == "put":
        iv_put_df = iv_df.loc[(iv_df["option_type"]=="put")&(iv_df["ticker"]==ticker)][["time2mature", "strike", iv_type]]
        iv_matrix_put = iv_put_df[["time2mature", "strike", iv_type]].drop_duplicates(subset=["time2mature", "strike"]).pivot(index="time2mature", columns="strike", values=iv_type)
        # visualize 3d iv surface
        _visualize_iv_surface(iv_matrix_put, ticker=ticker, option_type="EU Put", model=model)
      else:
        print("Call or Put not specified")
        return 0
```

```
[ ] # visualize_iv_surface_equal(ticker="AAPL", model=lr, iv_type='iv_eu_pred_lr', option_type="call,put", start_date="2013-01-02", end_date="2013-01-02", df=df, drop_na=False)
    visualize_iv_surface(ticker="AAPL", option_type="call,put", drop_na=True, iv_type='iv_eu_pred_lr', model="Dumas - Linear Regression", iv_df=df_viz)
```

Then, I move to second model, it is Decision Tree, the process similar to Linear Regression. I train the model

```
[ ]  from sklearn.tree import DecisionTreeRegressor
     import pickle
```

```
[ ]  # # drop nan
     # _dm_encoded = dm_encoded.dropna()

     # # run Regression using sklearn
     # X = _dm_encoded.drop(columns=[iv_type])
     # y = _dm_encoded[iv_type]
```

```
# training time = 5 secs

# # split train/test set with .75/.25 ratio
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)


dt = DecisionTreeRegressor(random_state = 0)
dt.fit(X_train, y_train)

# # load saved model
# model_save_name = 'eu_decision_tree.pickle'
# path = f"/content/drive/My Drive/data_implied_volatility/models/{model_save_name}"
# with open(path , 'rb') as f:
#     dt = pickle.load(f)

y_pred = dt.predict(X_test)


# The mean squared error
dt_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
# The R2
dt_r2 = r2_score(y_test, y_pred)
print("Decision Tree - Root Mean squared error: RMSE = %.6f" % dt_rmse)
print("Decision Tree - R2: R2 = %.6f" % dt_r2)
```

```
Decision Tree - Root Mean squared error: RMSE = 0.142390
Decision Tree - R2: R2 = 0.527823
```

Then use it to predict value:

```
[ ] ## Use fitted model to predict missing value
    def predict_iv_dm(ticker, option_type, strike, ttm, model=dt):
      if option_type == 'call':
        option_type = 1
      else:
        option_type = 0
      iv = np.nan
      X = pd.DataFrame(np.array([[iv, ttm, strike, strike**2, ttm**2, strike*ttm, option_type, ticker]]), columns=[iv_type, 'T', 'X', 'X^2', 'T^2', 'XT', 'option_type', 'ticker'])
      X_transformed = dm_encoder.transform(X)
      X_transformed.drop(columns=[iv_type], inplace=True)

      iv_pred = model.predict(X_transformed)
      return max(0.01, min(1, iv_pred[0]))

    print("Predicted value:")
    print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 200, ttm = 0.2, model=dt))
    print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 1050, ttm = 0.2, model=dt))

    print("\nActual value:")
    print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==200)&(all_df_dm['T']==0.2)][iv_type].values[0])
    print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==1050)&(all_df_dm['T']==0.2)][iv_type].values[0])
```

```
Predicted value:
0.999607542093466
0.5715448994044049

Actual value:
0.999607542093466
0.5715448994044049
```

```
[ ] ## Use fitted model to predict missing value
    # ## decoding ticker
    # df_dm = dm_encoder.inverse_transform(dm_encoded)

    # add predicted values
    iv_eu_pred_dt = dt.predict(dm_encoded.drop(columns=[iv_type]).values)
    df_dm.insert(loc=2, column='iv_eu_pred_dt', value=iv_eu_pred_dt)

    df_dm
```

	iv_eu_adj	iv_eu_pred_lr	iv_eu_pred_dt	time2mature	strike	X^2	T^2	XT	option_type	ticker
0	NaN	0.828757	0.999608	0.000000	135.0	18225.0	0.000000	0.000000	call	AAPL
1	0.999608	0.827723	0.999608	0.002740	135.0	18225.0	0.000008	0.369863	call	AAPL
2	0.999608	0.826691	0.999608	0.005479	135.0	18225.0	0.000030	0.739726	call	AAPL
3	0.999608	0.825661	0.999608	0.008219	135.0	18225.0	0.000068	1.109589	call	AAPL
4	0.999608	0.824634	0.999608	0.010959	135.0	18225.0	0.000120	1.479452	call	AAPL
...	...	...	...	...	...	...	...	...	...	...
1268020	NaN	0.234809	0.200164	2.024658	115.0	13225.0	4.099238	232.835616	put	WMT
1268021	NaN	0.235526	0.200164	2.027397	115.0	13225.0	4.110340	233.150685	put	WMT
1268022	NaN	0.237689	0.200164	2.035616	115.0	13225.0	4.143734	234.095890	put	WMT
1268023	NaN	0.238415	0.200164	2.038356	115.0	13225.0	4.154896	234.410959	put	WMT
1268024	NaN	0.239144	0.200164	2.041096	115.0	13225.0	4.166072	234.726027	put	WMT

1268025 rows × 10 columns

```
[ ] # add predicted values
    iv_eu_pred_dt = dt.predict(df_dm_space_transformed.drop(columns=[iv_type]).values)
    df_viz.insert(loc=0, column='iv_eu_pred_dt', value=iv_eu_pred_dt)

    df_viz
```

	iv_eu_pred_dt	iv_eu_pred_lr	time2mature	strike	X^2	T^2	XT	option_type	ticker
0	0.999608	0.828757	0.000000	135.000000	18225.000000	0.000000	0.000000	call	AAPL
1	0.999608	0.826768	0.005274	135.000000	18225.000000	0.000028	0.712010	call	AAPL
2	0.803750	0.824788	0.010548	135.000000	18225.000000	0.000111	1.424020	call	AAPL
3	0.999608	0.822816	0.015822	135.000000	18225.000000	0.000250	2.136031	call	AAPL
4	0.999608	0.820853	0.021097	135.000000	18225.000000	0.000445	2.848041	call	AAPL
...	...	...	...	...	...	...	...	...	...
827863	0.200164	0.237408	2.013662	112.265625	12603.570557	4.054834	226.065001	put	WMT
827864	0.200164	0.238828	2.019149	112.265625	12603.570557	4.076961	226.680982	put	WMT
827865	0.200164	0.240257	2.024635	112.265625	12603.570557	4.099149	227.296963	put	WMT
827866	0.200164	0.241695	2.030122	112.265625	12603.570557	4.121396	227.912944	put	WMT
827867	0.200164	0.243143	2.035609	112.265625	12603.570557	4.143704	228.528925	put	WMT

827868 rows × 9 columns

Then I start to visualize:

```
visualize_iv_surface(ticker="AAPL", option_type="call,put", drop_na=False, iv_type='iv_eu_pred_dt', model="Dumas - Decision Tree", iv_df=df_viz)
```

Next model is Random Forest, the process is the same:

Model training

```
[ ] from sklearn.ensemble import RandomForestRegressor
    import pickle
```

```
# # drop nan
# _dm_encoded = dm_encoded.dropna()

# # run Regression using sklearn
# X = _dm_encoded.drop(columns=[iv_type])
# y = _dm_encoded[iv_type]
```

```
[ ] # training time = 4mins
    # running time by loading saved model = 36 secs

    # # split train/test set with .75/.25 ratio
    # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)


    rf = RandomForestRegressor()
    rf.fit(X_train, y_train)

    # # load saved model
    # model_save_name = 'eu_random_forest.pickle'
    # path = f"/content/drive/My Drive/data_implied_volatility/models/{model_save_name}"
    # with open(path , 'rb') as f:
    #     rf = pickle.load(f)

    # y_pred = rf.predict(X_test)


    # The mean squared error
    rf_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    # The R2
    rf_r2 = r2_score(y_test, y_pred)
    print("Random Forest: Mean squared error: RMSE = %.6f" % rf_rmse)
    print("Random Forest: R2: RMSE = %.6f" % rf_r2)
```

```
Random Forest: Mean squared error: RMSE = 0.142390
Random Forest: R2: RMSE = 0.527823
```

Use model to predit missing value

```
## Use fitted model to predict missing value
def predict_iv_dm(ticker, option_type, strike, ttm, model=rf):
  if option_type == 'call':
    option_type = 1
  else:
    option_type = 0
  iv = np.nan
  X = pd.DataFrame(np.array([[iv, ttm, strike, strike**2, ttm**2, strike*ttm, option_type, ticker]]), columns=[iv_type, 'T', 'X', 'X^2', 'T^2', 'XT', 'option_type', 'ticker'])
  X_transformed = dm_encoder.transform(X)
  X_transformed.drop(columns=[iv_type], inplace=True)

  iv_pred = model.predict(X_transformed)
  return max(0.01, min(1, iv_pred[0]))

print("Predicted value:")
print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 200, ttm = 0.2, model=rf))
print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 1050, ttm = 0.2, model=rf))

print("\nActual value:")
print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==200)&(all_df_dm['T']==0.2)][iv_type].values[0])
print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==1050)&(all_df_dm['T']==0.2)][iv_type].values[0])
```

```
Predicted value:
0.9978890555823842
0.5611743197849536

Actual value:
0.999607542093466
0.5715448994044049
```

```
[ ] ## Use fitted model to predict missing value
# ## decoding ticker
# df_dm = dm_encoder.inverse_transform(dm_encoded)

# add predicted values
iv_eu_pred_rf = rf.predict(dm_encoded.drop(columns=[iv_type]).values)
df_dm.insert(loc=3, column='iv_eu_pred_rf', value=iv_eu_pred_rf)

df_dm
```

Then the last model is Gradient Boosting:

Model training

```
[ ] from sklearn.ensemble import GradientBoostingRegressor
```

```
[ ] # # drop nan
# _dm_encoded = dm_encoded.dropna()

# # run Regression using sklearn
# X = _dm_encoded.drop(columns=[iv_type])
# y = _dm_encoded[iv_type]
```

```
[ ] # training time = 1min

# # split train/test set with .75/.25 ratio
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

gb = GradientBoostingRegressor(random_state=0)
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)


# The mean squared error
gb_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
# The r2
gb_r2 = r2_score(y_test, y_pred)
print("Gradient Boosting - Mean squared error: RMSE = %.6f" % gb_rmse)
print("Gradient Boosting - R2: RMSE = %.6f" % gb_r2)
```

```
Gradient Boosting - Mean squared error: RMSE = 0.151075
Gradient Boosting - R2: RMSE = 0.468464
```

Use model to predit missing value

```
## Use fitted model to predict missing value
def predict_iv_dm(ticker, option_type, strike, ttm, model=gb):
  if option_type == 'call':
    option_type = 1
  else:
    option_type = 0
  iv = np.nan
  X = pd.DataFrame(np.array([[iv, ttm, strike, strike**2, ttm**2, strike*ttm, option_type, ticker]]), columns=[iv_type, 'T', 'X', 'X^2', 'T^2', 'XT', 'option_type', 'ticker'])
  X_transformed = dm_encoder.transform(X)
  X_transformed.drop(columns=[iv_type], inplace=True)

  iv_pred = model.predict(X_transformed)
  return max(0.01, min(1, iv_pred[0]))

print("Predicted value:")
print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 200, ttm = 0.2, model=gb))
print(predict_iv_dm(ticker = "AAPL", option_type='call', strike = 1050, ttm = 0.2, model=gb))

print("\nActual value:")
print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==200)&(all_df_dm['T']==0.2)][iv_type].values[0])
print(all_df_dm.loc[(all_df_dm['ticker']=="AAPL")&(all_df_dm['X']==1050)&(all_df_dm['T']==0.2)][iv_type].values[0])
```

```
Predicted value:
0.5548180874605053
0.5209281334216683

Actual value:
0.999607542093466
0.5715448994044049
```

```
## Use fitted model to predict missing value
# ## decoding ticker
# df_dm_gb = dm_encoder.inverse_transform(dm_encoded)

# add predicted values
iv_eu_pred_gb = gb.predict(dm_encoded.drop(columns=[iv_type]).values)
df_dm.insert(loc=4, column='iv_eu_pred_gb', value=iv_eu_pred_gb)

df_dm
```

At the end, I compare 4 models:

Compare models

```
model_names = ["Linear Regression", "Decision Tree", "Random Forest", "Gradient Boosting"]
rmses = [lr_rmse, dt_rmse, rf_rmse, gb_rmse]
r2 = [lr_r2, dt_r2, rf_r2, gb_r2]

models_compare = pd.DataFrame(list(zip(model_names, rmses, r2)), columns=["Models", "RMSE", "R2"])
# sort the result by RMSE
models_compare.sort_values(by="RMSE", inplace=True, ignore_index=True)
models_compare
```

	Models	RMSE	R2
0	Random Forest	0.131763	0.595669
1	Decision Tree	0.142390	0.527823
2	Gradient Boosting	0.151075	0.468464
3	Linear Regression	0.183144	0.218851

3.2.4 Predict Implied Volatility using Machine Learning for American option

The process in American option is similar to European option and the results is as below:

```
model_names = ["Linear Regression", "Decision Tree", "Random Forest", "Gradient Boosting"]
rmses = [lr_rmse, dt_rmse, rf_rmse, gb_rmse]
r2 = [lr_r2, dt_r2, rf_r2, gb_r2]

am_models_compare = pd.DataFrame(list(zip(model_names, rmses, r2)), columns=["Models", "RMSE", "R2"])
# sort the result by RMSE
am_models_compare.sort_values(by="RMSE", inplace=True, ignore_index=True)
am_models_compare['option_type'] = "AM"
am_models_compare
```

	Models	RMSE	R2	option_type
0	Random Forest	0.068114	0.871866	AM
1	Decision Tree	0.087163	0.790176	AM
2	Gradient Boosting	0.106985	0.683891	AM
3	Linear Regression	0.142467	0.439447	AM

References

References should be formatted using APA or Harvard style as detailed in NCI Library Referencing Guide available at https://libguides.ncirl.ie/referencing
You can use a reference management system such as Zotero or Mendeley to cite in MS Word.

Aroussi, R. (2023). yfinance: Yahoo! Finance market data downloader. [online] PyPI. Available at: https://pypi.org/project/yfinance/.

optiondata.org. (n.d.). Historical Options Data. [online] Available at: https://optiondata.org/#sampleId.

Srivastava, A. and Shastri, M. (2018). A study of relevance of Black-Scholes model on option prices of Indian stock market. International Journal of Governance and Financial Intermediation, 1(1), p.82. doi:https://doi.org/10.1504/ijgfi.2018.091495.

Wang, J., Chen, S., Tao, Q. and Zhang, T. (2017). Modelling the implied volatility surface based on Shanghai 50ETF options. Economic Modelling, 64, pp.295–301. doi:https://doi.org/10.1016/j.econmod.2017.04.009.