PROJECT REPORT

Implementing Option Strategies Using Implied Volatility in Code

Presented By

Mr. Sarthak Jain (sarthakjain324@gmail.com)
Mr. Garvit Arora (aroragarvit321@gmail.com)

Guided By
Dr. Aviral Sharma

INDEX

Abstract

The goal of this project is to analyse data from options trading, with Implied Volatility as crucial parameter which is used to calculate the Relative Strength Index (RSI), which is an important indicator for making decisions. The method uses the pandas_ta library to calculate RSI values, effectively analyses numerous CSV files containing financial data, and integrates a simple trading strategy. The code keeps a close eye on every trade's profit and loss, giving detailed information on each one as well as cumulative outcomes for put and call options.

Introduction

1.1 Purpose of project

The purpose of the project is to implement and analyze option trading strategies using the Relative Strength Index (RSI) as a key indicator, with a specific focus on incorporating Implied Volatility (IV) into the decision-making process. The code processes financial data from multiple CSV files, calculates RSI values using pandas_ta, and executes a straightforward trading strategy based on RSI conditions. The project aims to enhance the strategy by considering the impact of Implied Volatility on option trading decisions. The ultimate goal is to develop a robust and effective trading algorithm that combines RSI and IV insights for improved decision-making in options trading.

1.2 Target Beneficiary

The target beneficiaries of the project are individuals or entities involved in options trading, particularly those seeking an algorithmic approach to decision-making. Traders, investors, or financial professionals looking to enhance their strategies through the implementation of option trading algorithms based on the Relative Strength Index (RSI) and consideration of Implied Volatility (IV) would benefit from this project. The code aims to provide a tool that aids in making informed and data-driven decisions, potentially optimizing trading outcomes for those actively engaged in options markets.

1.3 Project Scope

The project scope involves developing and implementing algorithmic strategies for options trading, leveraging the Relative Strength Index (RSI) as a primary indicator and incorporating Implied Volatility (IV) considerations. The code will process financial data from multiple CSV files, calculate RSI values, and execute a trading strategy based on RSI conditions. The overarching goal is to create a comprehensive and adaptable solution that enhances decision-making in options trading by integrating insights from both RSI and IV. The scope includes refining the code for robustness, implementing error handling, providing clear documentation, and conducting thorough testing, to evaluate the effectiveness of the trading strategy.

Literature Review

- This seminal paper introduced the Black-Scholes model, a foundational framework for pricing European-style options. The model relies on parameters such as the strike price, time to expiration, risk-free interest rate, and, critically, implied volatility to determine option prices.[1]
- This paper discusses extensions of the Black-Scholes model, including variations that incorporate implied volatility surfaces. Implied volatility surfaces provide more nuanced insights into how volatility expectations vary across different strike prices and expiration dates.[2]
- This paper explores the relationship between implied volatility and option trading strategies. It investigates how implied volatility impacts portfolio returns, offering valuable insights for traders seeking to optimize their strategies based on implied volatility expectations.[3]
- Natenberg's book is a comprehensive resource that delves into the practical aspects of option trading. It explains how implied volatility informs various option strategies, helping traders make informed decisions based on market volatility expectations.[4]
- This review provides an overview of volatility forecasting methods, including statistical and econometric models. While not explicitly focused on implied volatility, it offers a foundation for understanding volatility prediction, which can be applied to implied volatility.[5]
- Ong's book covers risk management techniques in financial markets comprehensively. It includes discussions of how implied volatility is used in risk assessment and management within the context of options trading.[6]

Problem Statement

The problem addressed by this project is the need for an effective algorithmic approach in options trading. Traders often face challenges in decision-making, and this project aims to develop a solution using the Relative Strength Index (RSI) and Implied Volatility (IV). The problem is to create a robust and adaptable code that can process financial data, calculate RSI values, and execute a trading strategy, ultimately providing traders with a tool that enhances decision-making and potentially improves outcomes in options trading scenarios.

Objectives

Main Objective:

Develop a computational platform for implementing and optimizing option trading strategies, with a primary focus on incorporating the Relative Strength Index (RSI) and Implied Volatility (IV) as crucial parameters.

Sub-Objectives:

- Data Collection and Preprocessing
- Calculate RSI with the help of Implied Volatility (IV) from financial data.
- Implement Option Trading Strategy based on RSI and IV conditions.
- Customize the algorithm for adaptability and execute live trading simulations.

Methodology

Data Collection:

- Collect historical options data, including option prices, and spot data with underlying asset prices.
- Gather relevant market data, which may include interest rates, dividends, or any other factors influencing options pricing.

Data Preprocessing:

- Organize the acquired data for efficient processing.
- Sort data by date or any relevant identifiers for chronological analysis

Data Quality Assurance:

- Ensure the quality, accuracy, and consistency of the collected data.
- Implement checks to identify and handle any data discrepancies.
- Handle missing values appropriately, considering interpolation or elimination.

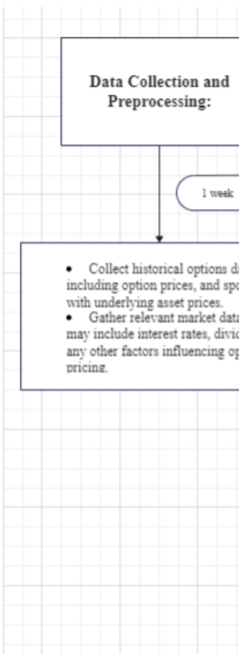
Implied Volatility Analysis:

- Implement algorithms to calculate RSI from the pre-processed financial data with the help of Implied Volatility (IV).
- Utilize appropriate formulas and libraries for accurate calculations.

Strategy Optimization:

- Develop algorithms for generating option trading signals based on RSI and IV conditions.
- Define criteria for buy and sell signals based on RSI thresholds and implied volatility levels.
- Define criteria for buy and sell signals, taking into account market conditions.

Pert Chart



Code and Results

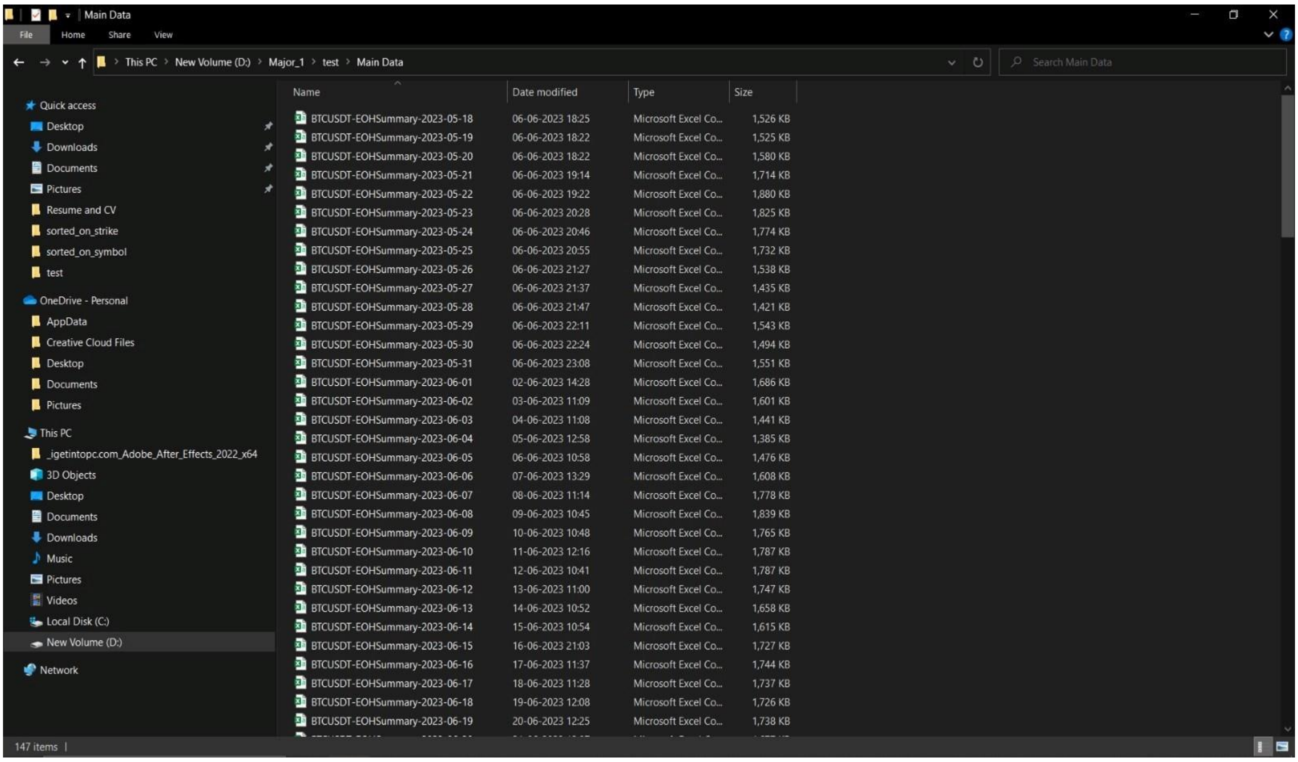
- ## Dataset And Input Format

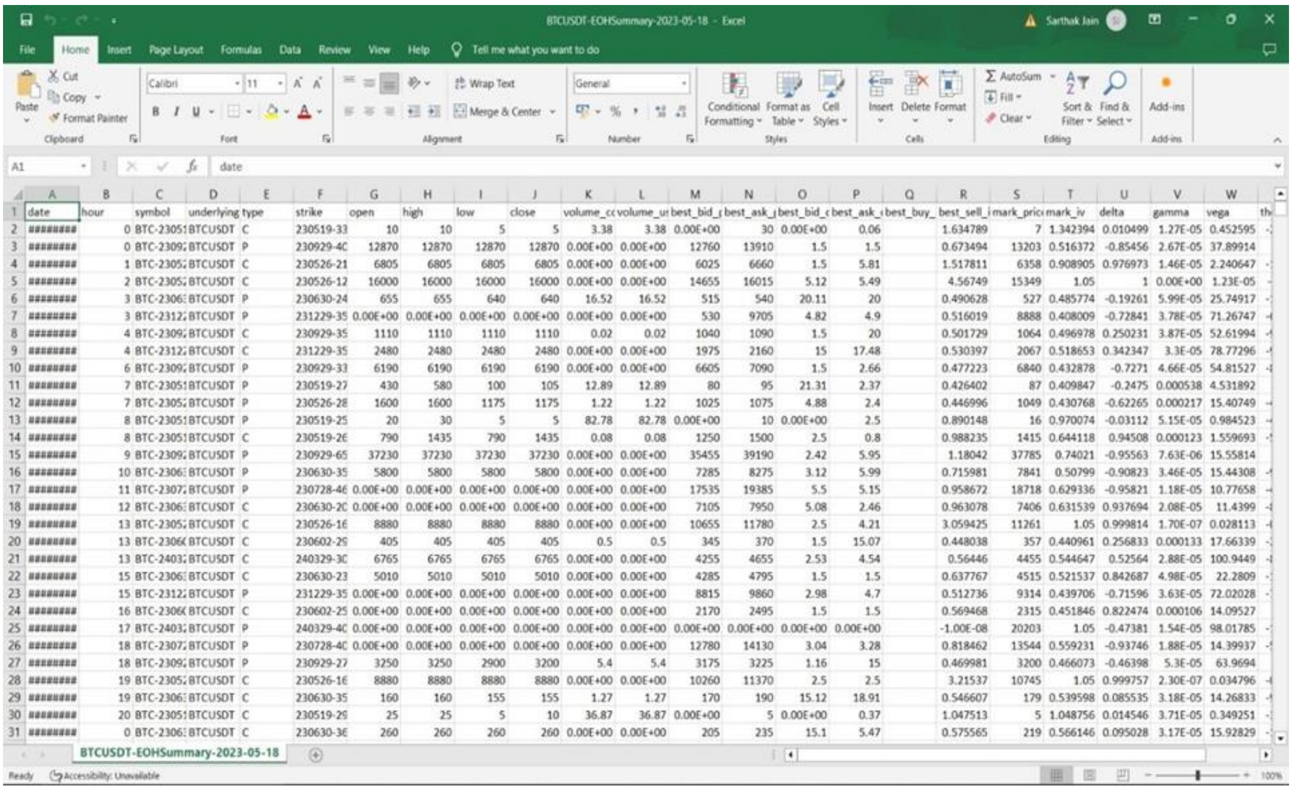
To access a dataset that includes historical options contract information with specific parameters, such as

date, hour, symbol, underlying, type, strike, open, high, low, close, volume_contracts, volume_usdt, best_bid_price, best_ask_price, best_bid_qty, best_ask_qty, best_sell_iv, mark_price, mark_iv, delta, gamma, vega, theta, openinterest_contracts, openinterest_usdt
We utilize the Binance API. This API is used to retrieve data from the following link:
https://www.binance.com/en/eoptions/BTCUSDT

```
binance_data_pull.py > ...
import requests

# Define the Binance API endpoint for fetching option data "startTime" : "1694667744"
base_api_url = "https://www.binance.com/en/eoptions/BTCUSDT?symbol=BTC-231014-26750-C"

# Pass an API key in headers
headers = {
    'X-MBX-APIKEY': 'q5OpDbcs15GWlY23sAuAJWavrkvziGGTl0LTgdUiWOCK5HaoDBkyCvLzACuQpryG'
}
parameters = {
    "symbol" : "BTC-231014-26500-C",
    "interval": "5m"
    }

# Make the HTTP GET request
response = requests.get(base_api_url)#, params = parameters, headers=headers)
print(response)

print(response.content)

folder_path = "D:\Major_1/test"
```

- The code imports the ‘requests’ module, which is essential for making HTTP requests to the Binance API**.**
- The ‘base_api_url’ variable stores the URL of the Binance API endpoint that will be used to fetch option data. It appears to be specific to the BTC/USDT trading pair with additional parameters.
- The **parameters** variable is a dictionary that contains two parameters:
- **"symbol"**: Specifies the option contract to be retrieved ("BTC-231814-26500-C"). **"interval"**: Specifies the time interval for the data
- The code sends an HTTP GET request to the Binance API using the **requests.get()** method. However, the code is currently commented out, and the **parameters** and **headers** are not included in the request.
- The **folder_path** variable contains the path where the code intends to store or process the fetched data. In this case, it is set to "D:\Major_1/test."

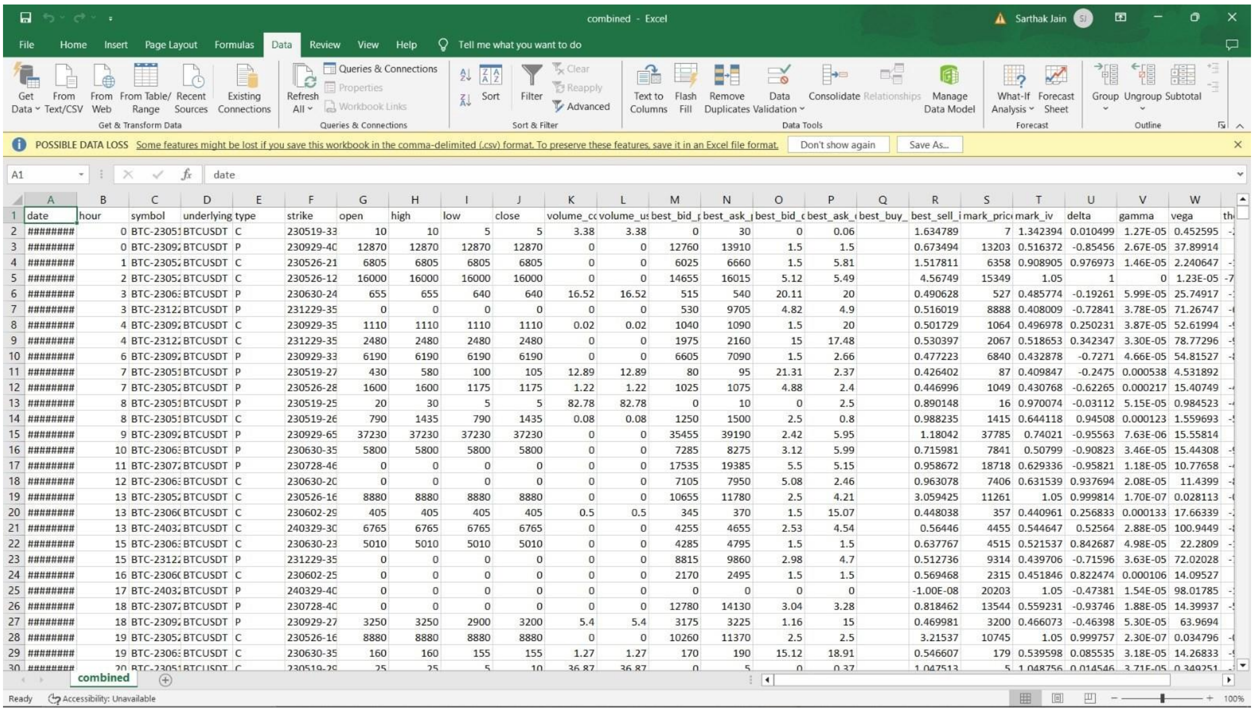

Main Data
File
Home
Share
View
This PC › New Volume (D:) › Major_1 › test › Main Data
Search Main Data
Name
Date modified
Type
Size
BTCUSDT-EOHSummary-2023-05-18 06-06-2023 18:25 Microsoft Excel Co... 1,526 KB
BTCUSDT-EOHSummary-2023-05-19 06-06-2023 18:22 Microsoft Excel Co... 1,525 KB
BTCUSDT-EOHSummary-2023-05-20 06-06-2023 18:22 Microsoft Excel Co... 1,580 KB
BTCUSDT-EOHSummary-2023-05-21 06-06-2023 19:14 Microsoft Excel Co... 1,714 KB
BTCUSDT-EOHSummary-2023-05-22 06-06-2023 19:22 Microsoft Excel Co... 1,880 KB
BTCUSDT-EOHSummary-2023-05-23 06-06-2023 20:28 Microsoft Excel Co... 1,825 KB
BTCUSDT-EOHSummary-2023-05-24 06-06-2023 20:46 Microsoft Excel Co... 1,774 KB
BTCUSDT-EOHSummary-2023-05-25 06-06-2023 20:55 Microsoft Excel Co... 1,732 KB
BTCUSDT-EOHSummary-2023-05-26 06-06-2023 21:27 Microsoft Excel Co... 1,538 KB
BTCUSDT-EOHSummary-2023-05-27 06-06-2023 21:37 Microsoft Excel Co... 1,435 KB
BTCUSDT-EOHSummary-2023-05-28 06-06-2023 21:47 Microsoft Excel Co... 1,421 KB
BTCUSDT-EOHSummary-2023-05-29 06-06-2023 22:11 Microsoft Excel Co... 1,543 KB
BTCUSDT-EOHSummary-2023-05-30 06-06-2023 22:24 Microsoft Excel Co... 1,494 KB
BTCUSDT-EOHSummary-2023-05-31 06-06-2023 23:08 Microsoft Excel Co... 1,551 KB
BTCUSDT-EOHSummary-2023-06-01 02-06-2023 14:28 Microsoft Excel Co... 1,686 KB
BTCUSDT-EOHSummary-2023-06-02 03-06-2023 11:09 Microsoft Excel Co... 1,601 KB
BTCUSDT-EOHSummary-2023-06-03 04-06-2023 11:08 Microsoft Excel Co... 1,441 KB
BTCUSDT-EOHSummary-2023-06-04 05-06-2023 12:58 Microsoft Excel Co... 1,385 KB
BTCUSDT-EOHSummary-2023-06-05 06-06-2023 10:58 Microsoft Excel Co... 1,476 KB
BTCUSDT-EOHSummary-2023-06-06 07-06-2023 13:29 Microsoft Excel Co... 1,608 KB
BTCUSDT-EOHSummary-2023-06-07 08-06-2023 11:14 Microsoft Excel Co... 1,778 KB
BTCUSDT-EOHSummary-2023-06-08 09-06-2023 10:45 Microsoft Excel Co... 1,839 KB
BTCUSDT-EOHSummary-2023-06-09 10-06-2023 10:48 Microsoft Excel Co... 1,765 KB
BTCUSDT-EOHSummary-2023-06-10 11-06-2023 12:16 Microsoft Excel Co... 1,787 KB
BTCUSDT-EOHSummary-2023-06-11 12-06-2023 10:41 Microsoft Excel Co... 1,787 KB
BTCUSDT-EOHSummary-2023-06-12 13-06-2023 11:00 Microsoft Excel Co... 1,747 KB
BTCUSDT-EOHSummary-2023-06-13 14-06-2023 10:52 Microsoft Excel Co... 1,658 KB
BTCUSDT-EOHSummary-2023-06-14 15-06-2023 10:54 Microsoft Excel Co... 1,615 KB
BTCUSDT-EOHSummary-2023-06-15 16-06-2023 21:03 Microsoft Excel Co... 1,727 KB
BTCUSDT-EOHSummary-2023-06-16 17-06-2023 11:37 Microsoft Excel Co... 1,744 KB
BTCUSDT-EOHSummary-2023-06-17 18-06-2023 11:28 Microsoft Excel Co... 1,737 KB
BTCUSDT-EOHSummary-2023-06-18 19-06-2023 12:08 Microsoft Excel Co... 1,726 KB
BTCUSDT-EOHSummary-2023-06-19 20-06-2023 12:25 Microsoft Excel Co... 1,738 KB
147 items

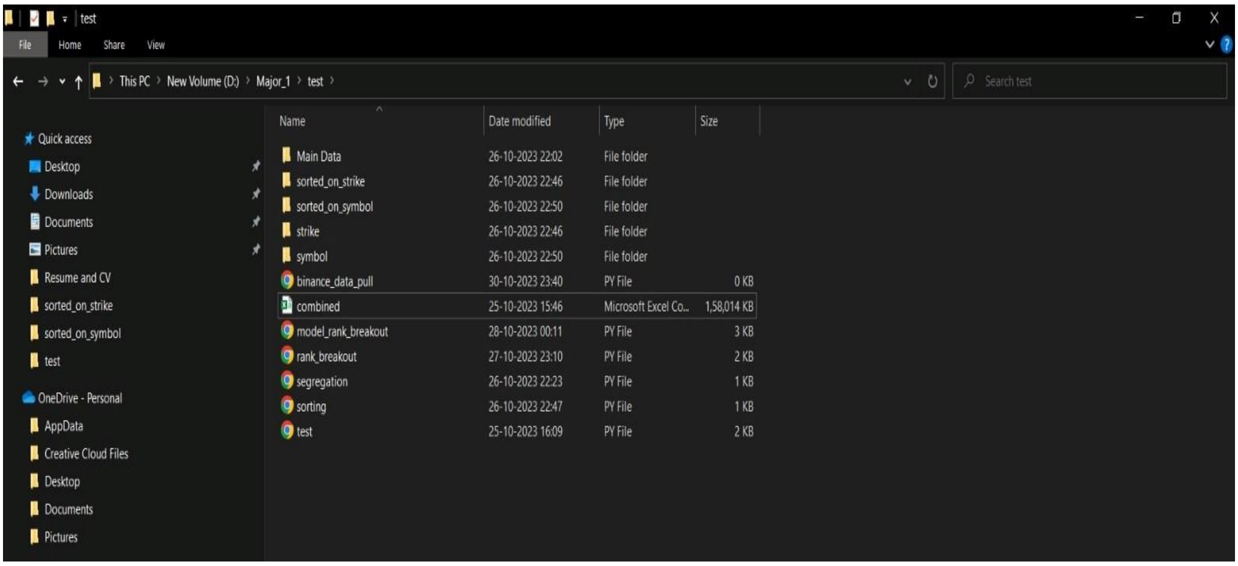

BTCUSDT-EOHSummary-2023-05-18 - Excel
BTCUSDT-EOHSummary-2023-05-18

After retrieving data through the API, we combine or merge all the individual files into a single file or dataset.

```
y > combine_csv_files
import pandas as pd
import os

def combine_csv_files(folder_path, output_file):
    combined_data = pd.DataFrame()

    for filename in os.listdir(folder_path):
        if filename.endswith(".csv"):
            file_path = os.path.join(folder_path, filename)

            # Read each CSV file into a DataFrame
            df = pd.read_csv(file_path)

            # Check if the columns in df match the columns in combined_data
            if combined_data.empty:
                combined_data = df
            else:
                # Check if the columns match
                if all(df.columns == combined_data.columns):
                    # Concatenate the data to the combined_data DataFrame
                    combined_data = pd.concat([combined_data, df], ignore_index=True)
                else:
                    print(f"Columns in {filename} do not match. Skipping the file.")

    # Write the combined data to a new CSV file
    combined_data.to_csv(output_file, index=False)

    print(f"Combined CSV files saved to {output_file}")

folder_path = "D:\Major_1/test"  # Replace with your folder path
output_file = "combined.csv"

combine_csv_files(folder_path, output_file)
```

- The code imports two essential libraries: **pandas** (as **pd**) for data manipulation and **os** for working with the operating system.
- The code defines a function called **combine_csv_files** that takes two arguments: **folder_path** (the directory where CSV files are located) and **output_file** (the name of the combined CSV file to be created).
- An empty DataFrame named combined_data is initialized to store the combined data from multiple CSV files. It checks if the current file being examined has a ".csv" extension by using the filename.endswith(".csv") condition.
- The code sets the folder_path variable to the path where the CSV files are located and output_file to the name of the combined CSV file. It then calls the combine_csv_files function with these arguments to execute the combining process.


combined - Excel
POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.
date
hour
symbol
underlying
type
strike
open
high
low
close
best_bid_
best_ask_
best_bid_
best_ask_
best_buy_
best_sell_i
mark_pric
mark_iv
delta
gamma
vega
combined


test
This PC › New Volume (D:) › Major_1 › test
Name
Date modified
Type
Size
Main Data 26-10-2023 22:02 File folder
sorted_on_strike 26-10-2023 22:46 File folder
sorted_on_symbol 26-10-2023 22:50 File folder
strike 26-10-2023 22:46 File folder
symbol 26-10-2023 22:50 File folder
binance_data_pull 30-10-2023 23:40 PY File 0 KB
combined 25-10-2023 15:46 Microsoft Excel Co... 1,58,014 KB
model_rank_breakout 28-10-2023 00:11 PY File 3 KB
rank_breakout 27-10-2023 23:10 PY File 2 KB
segregation 26-10-2023 22:23 PY File 1 KB
sorting 26-10-2023 22:47 PY File 1 KB
test 25-10-2023 16:09 PY File 2 KB

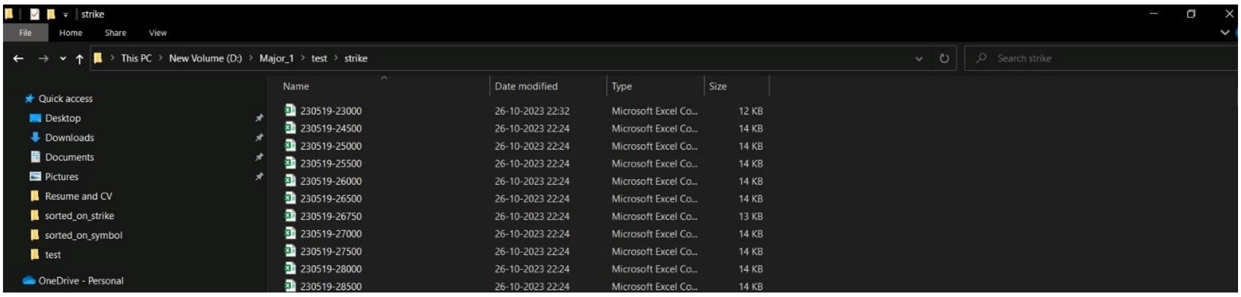
To facilitate the efficient processing of a large volume of unstructured data, we structured the data based on the Symbol and strike price using the following code.

```
egation.py > ...
import pandas as pd

# Load the original CSV file
df = pd.read_csv('combined.csv')

# Specify the column name for splitting
column_name = 'strike'

# Get unique values in the specified column
unique_values = df[column_name].unique()

# Create separate CSV files for each unique value
for value in unique_values:
    # Create a new DataFrame containing only rows with the current value
    sub_df = df[df[column_name] == value]

    # Define the name for the output CSV file based on the value
    output_filename = f'{value}.csv'

    # Save the sub DataFrame to a new CSV file
    sub_df.to_csv(output_filename, index=False)
```

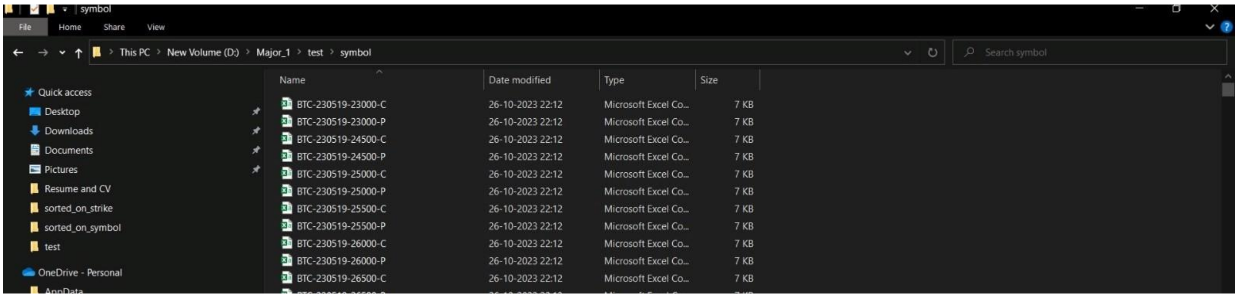
```
ting.py > ...
import os
import pandas as pd

# Specify the folder where your CSV files are located
folder_path = 'sorted_on_symbol'

# Specify the three columns for hierarchical sorting
sorting_columns = ['date', 'hour', 'type']

# Get a list of all CSV files in the folder
csv_files = [file for file in os.listdir(folder_path) if file.endswith('.csv')]

# Iterate through each CSV file and sort it hierarchically
for file in csv_files:
    # Read the CSV file into a DataFrame
    df = pd.read_csv(os.path.join(folder_path, file))

    # Sort the DataFrame hierarchically by the specified columns
    df_sorted = df.sort_values(by=sorting_columns)

    # Define the path for the sorted CSV file (you can customize the output path)
```

```
    output_path = os.path.join(folder_path, f'sorted_{file}')

    # Save the sorted DataFrame to a new CSV file
    df_sorted.to_csv(output_path, index=False)

print("CSV files have been sorted hierarchically!")
```

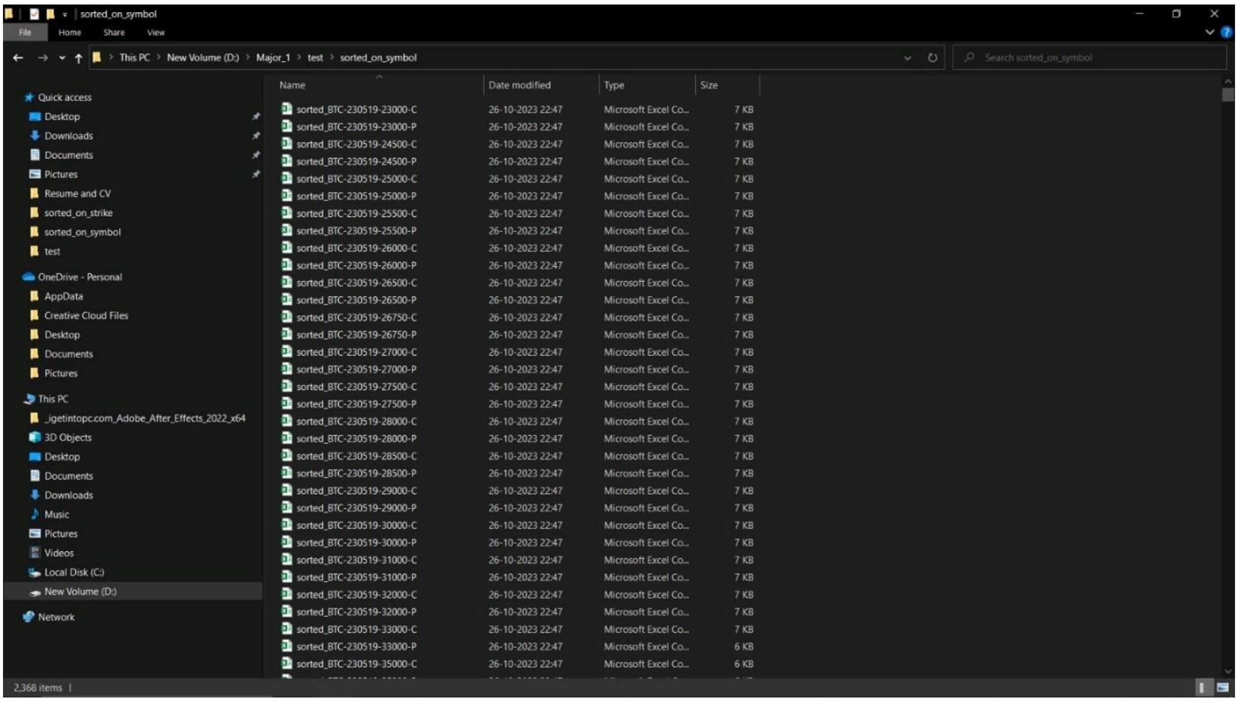
- The code starts by loading an original CSV file named 'combined.csv' into a Pandas DataFrame.
- It identifies a specific column to split the data, using the 'column_name' variable, which is set to ‘strike’.
- The unique values in the specified column are determined, and for each unique value, a new DataFrame, 'sub_df,' is created containing rows with that value.
- Each 'sub_df' is saved as a separate CSV file, where the filename is based on the unique value from the 'column_name'.
- The code then processes a folder containing multiple CSV files. The folder path is specified as 'folder_path,' which is set to 'sorted_on_symbol'.
- It iterates through each CSV file in the folder, reads it into a DataFrame, and sorts the DataFrame hierarchically based on the specified columns.
- The sorted DataFrame is saved as a new CSV file in the same folder, with a filename starting with 'sorted_' andf ollowed by the original filename.



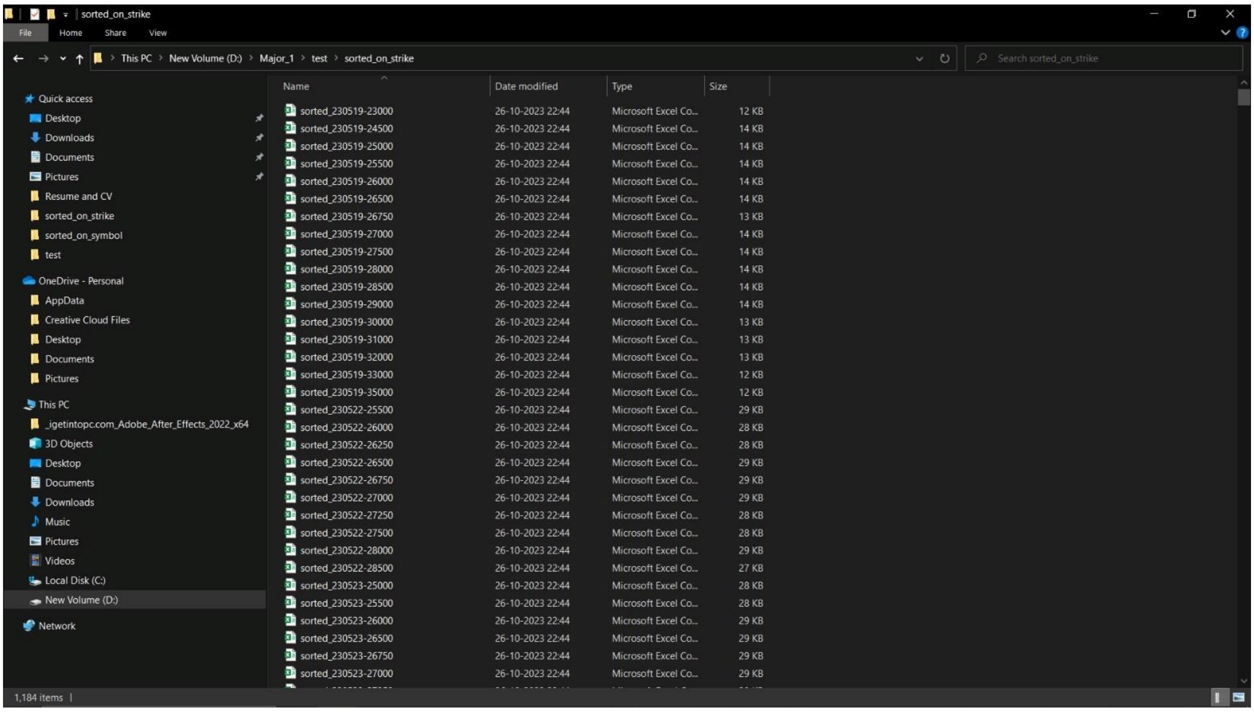
segregated on basis of Strike price



segregated on basis of Symbol



Sorted on basis of symbol



Sorted on basis of strike price

Pulling Spot Data from Binanace

```
Click here to ask Blackbox to help you code faster
from binance_historical_data import BinanceDataDumper
from datetime import datetime
```

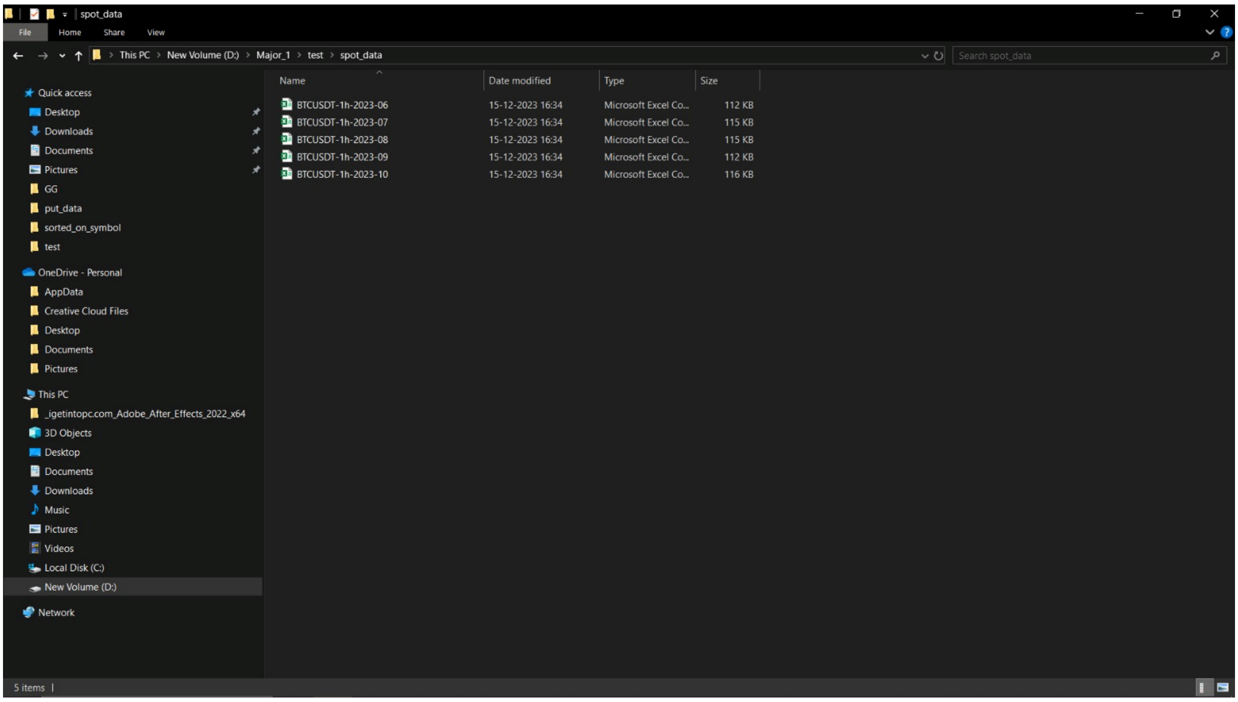
```
Click here to ask Blackbox to help you code faster
data_dumper = BinanceDataDumper(
    path_dir_where_to_dump="D:\\Major_1\\test\\",
    asset_class="spot",  # spot, um, cm
    data_type="klines",  # aggTrades, klines, trades
    data_frequency="1h",
)
```

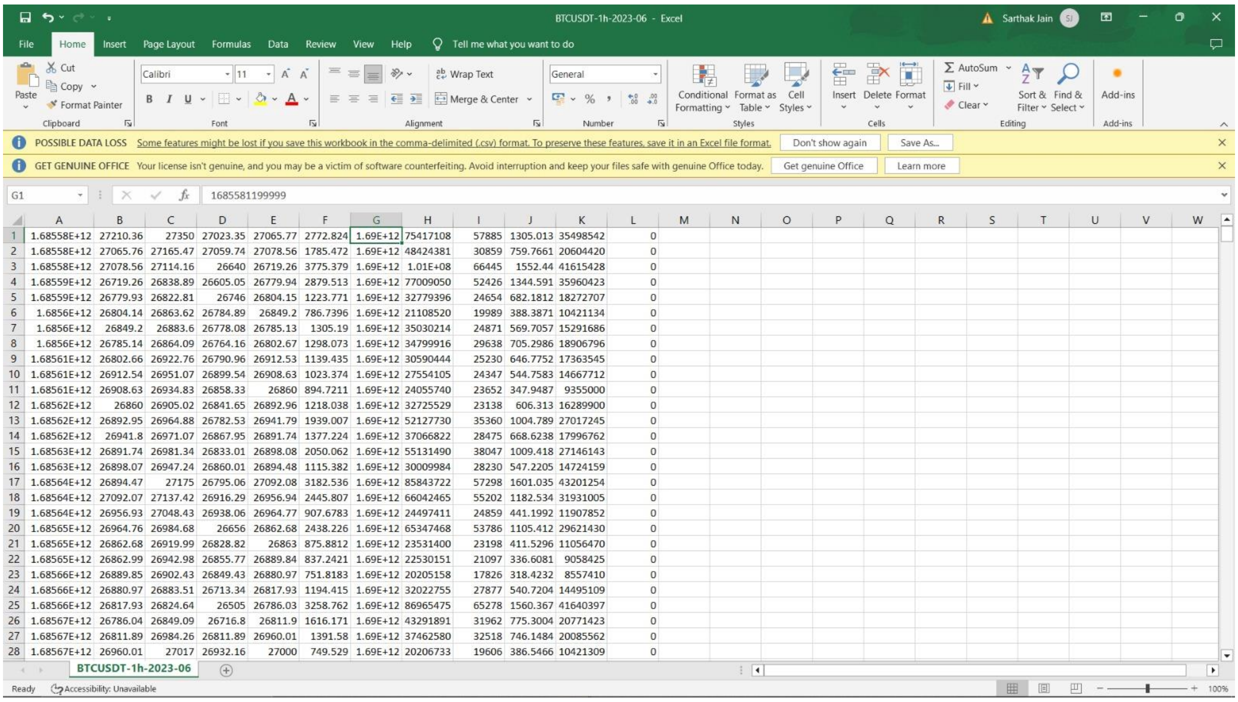
```
Click here to ask Blackbox to help you code faster
data_dumper.dump_data(
    tickers="BTCUSDT",
    date_start=None, #datetime.date(year=2023, month=5, day=18),
    date_end=None, #datetime.date(year=2023, month=10, day=24),
    is_to_update_existing=False,
    tickers_to_exclude=None,
)
```

```
---> Found overall tickers: 2396
---> Filter to asked tickers: 7
------> Tickers left: 1
Download full data for 1 tickers:
---> Data will be saved here: D:\Major_1\test\spot
---> Data Frequency: 1h
---> Start Date: 20170101
---> End Date: 20231215
```

```
💡 Click here to ask Blackbox to help you code faster
data_dumper.dump_data()

---> Found overall tickers: 254
---> Filter to USDT tickers
------> Tickers left: 211
Download full data for 211 tickers:
---> Data will be saved here: D:\Crypto_data\USDT_BTC\um
---> Data Frequency: 1m
---> Start Date: 20170101
---> End Date: 20230925

Tickers:   0%|          | 0/211 [00:00<?, ?it/s]

Tried to dump data for 211 tickers:
---> General stats:
------> NEW Data WAS dumped for 210 trading pairs
------> NEW Data WASN'T dumped for 1 trading pairs
---> New months saved:
------> For 17 tickers saved: 36 months
------> For 14 tickers saved: 7 months
------> For 14 tickers saved: 30 months
------> For 13 tickers saved: 43 months
------> For 10 tickers saved: 44 months
```

- The code imports the BinanceDataDumper class from the binance_historical_data module and the datetime class from the datetime module.
- An instance of the BinanceDataDumper class is created with specified parameters, such as the directory path where data will be dumped, the asset class (spot, um, cm), data type (aggTrades, klines, trades), and data frequency (1h).
- The dump_data method of the data_dumper object is then called to initiate the data dumping process.
- Parameters for data dumping are provided, including the trading pair symbol ("BTCUSDT"), start and end dates for historical data (None for the entire available data), a flag to update existing data (update_existing set to False), and an optional list of tickers to exclude.
- The dump_data method is called again without any parameters, indicating that it will use the default values specified during the object creation.

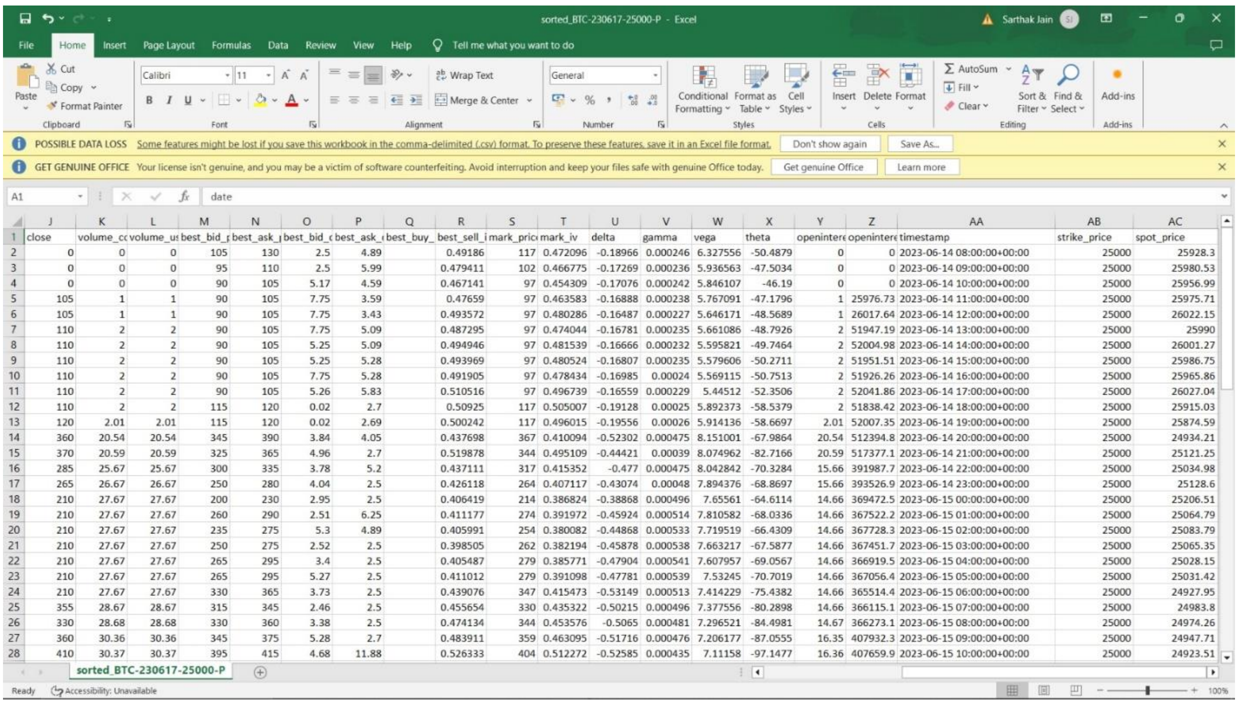

spot_data
This PC > New Volume (D:) > Major_1 > test > spot_data
Name
Date modified
Type
Size
BTCUSDT-1h-2023-06 15-12-2023 16:34 Microsoft Excel Co... 112 KB
BTCUSDT-1h-2023-07 15-12-2023 16:34 Microsoft Excel Co... 115 KB
BTCUSDT-1h-2023-08 15-12-2023 16:34 Microsoft Excel Co... 115 KB
BTCUSDT-1h-2023-09 15-12-2023 16:34 Microsoft Excel Co... 112 KB
BTCUSDT-1h-2023-10 15-12-2023 16:34 Microsoft Excel Co... 116 KB
5 items


BTCUSDT-1h-2023-06 - Excel
POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.
GET GENUINE OFFICE Your license isn't genuine, and you may be a victim of software counterfeiting. Avoid interruption and keep your files safe with genuine Office today.
G1
1685581199999
BTCUSDT-1h-2023-06

Pre-Processing of Spot Data

```
a_pp.py > ...
import os
import glob
import pandas as pd
from datetime import datetime, timezone

def convert_to_utc(row):
    date_str = f"{row['date']} {int(row['hour']):02d}:00"
    return datetime.strptime(date_str, "%Y-%m-%d %H:%M").replace(tzinfo=timezone.utc)

def unix_to_utc(unix_time):
    return datetime.utcfromtimestamp(int(unix_time) / 1000).replace(minute=0, second=0, microsecond=0, tzinfo=timezone.utc)

def extract_strike_price(strike_value):
    return strike_value.split('-')[1] if '-' in strike_value else None

if not os.path.exists('final_dat'):
    os.makedirs('final_dat')

# Create a dictionary for timestamps and spot prices from folder2
timestamp_spot_price = {}
for file_path in glob.glob('spot_data/*.csv'):
    df2 = pd.read_csv(file_path, header=None)
    df2['timestamp'] = df2[6].apply(unix_to_utc)
    for _, row in df2.iterrows():
        timestamp_spot_price[row['timestamp']] = row[4]
```

```
# Process files in folder1 and update with spot_price
for file_path in glob.glob('sorted_on_symbol/*.csv'):
    df1 = pd.read_csv(file_path)
    df1['timestamp'] = df1.apply(convert_to_utc, axis=1)
    df1['strike_price'] = df1['strike'].apply(extract_strike_price)

    df1['spot_price'] = df1['timestamp'].map(timestamp_spot_price)

    df1.to_csv(f"final_dat/{os.path.basename(file_path)}", index=False)

print("Processing complete. Modified files saved in 'final_dat' folder.")
```



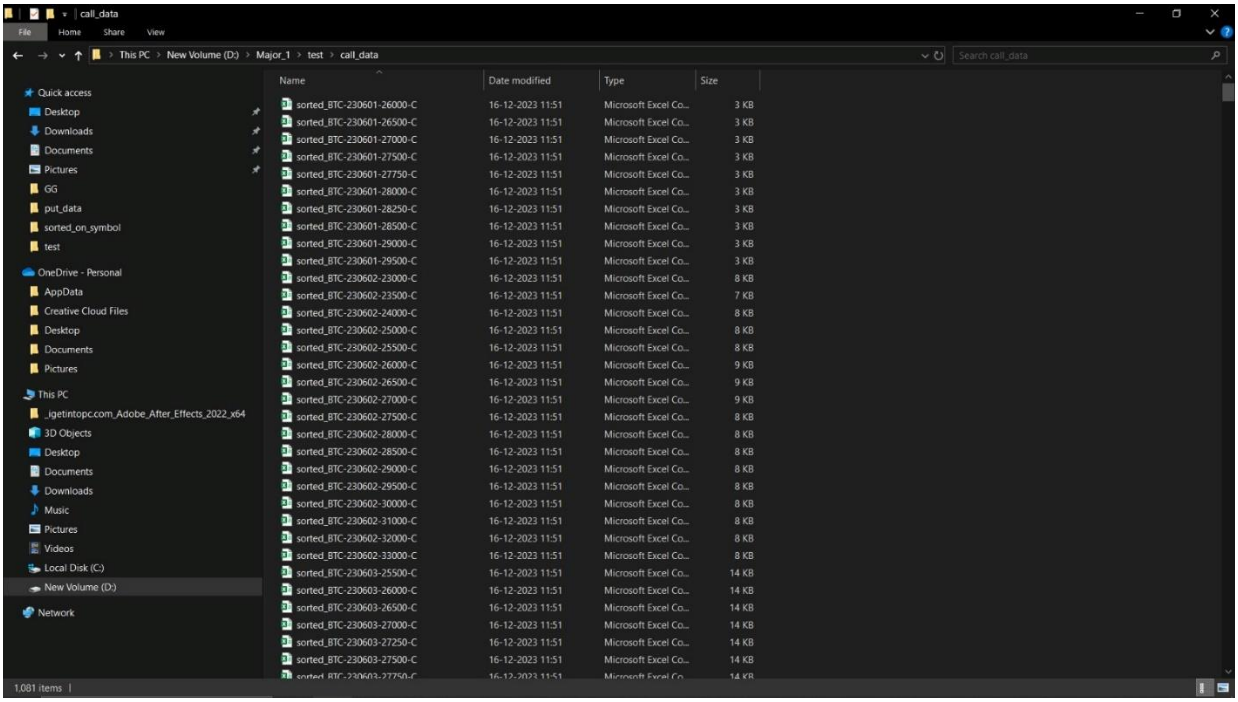
	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	close	volume_cc	volume_us	best_bid_p	best_ask_p	best_bid_c	best_ask_c	best_buy_	best_sell_i	mark_pric	mark_iv	delta	gamma	vega	theta	openintere	openintere	timestamp	strike_price	spot_price
2	0	0	0	105	130	2.5	4.89		0.49186	117	0.472096	-0.18966	0.000246	6.327556	-50.4879	0	0	2023-06-14 08:00:00+00:00	25000	25928.3
3	0	0	0	95	110	2.5	5.99		0.479411	102	0.466775	-0.17269	0.000236	5.936563	-47.5034	0	0	2023-06-14 09:00:00+00:00	25000	25980.53
4	0	0	0	90	105	5.17	4.59		0.467141	97	0.454309	-0.17076	0.000242	5.846107	-46.19	0	0	2023-06-14 10:00:00+00:00	25000	25956.99
5	105	1	1	90	105	7.75	3.59		0.47659	97	0.463583	-0.16888	0.000238	5.767091	-47.1796	1	25976.73	2023-06-14 11:00:00+00:00	25000	25975.71
6	105	1	1	90	105	7.75	3.43		0.493572	97	0.480286	-0.16487	0.000227	5.646171	-48.5689	1	26017.64	2023-06-14 12:00:00+00:00	25000	26022.15
7	110	2	2	90	105	7.75	5.09		0.487295	97	0.474044	-0.16781	0.000235	5.661086	-48.7926	2	51947.19	2023-06-14 13:00:00+00:00	25000	25990
8	110	2	2	90	105	5.25	5.09		0.494946	97	0.481539	-0.16666	0.000232	5.595821	-49.7464	2	52004.98	2023-06-14 14:00:00+00:00	25000	26001.27
9	110	2	2	90	105	5.25	5.28		0.493969	97	0.480524	-0.16807	0.000235	5.579606	-50.2711	2	51951.51	2023-06-14 15:00:00+00:00	25000	25986.75
10	110	2	2	90	105	7.75	5.28		0.491905	97	0.478434	-0.16985	0.00024	5.569115	-50.7513	2	51926.26	2023-06-14 16:00:00+00:00	25000	25965.86
11	110	2	2	90	105	5.26	5.83		0.510516	97	0.496739	-0.16559	0.000229	5.44512	-52.3506	2	52041.86	2023-06-14 17:00:00+00:00	25000	26027.04
12	110	2	2	115	120	0.02	2.7		0.50925	117	0.505007	-0.19128	0.00025	5.892373	-58.5379	2	51838.42	2023-06-14 18:00:00+00:00	25000	25915.03
13	120	2.01	2.01	115	120	0.02	2.69		0.500242	117	0.496015	-0.19556	0.00026	5.914136	-58.6697	2.01	52007.35	2023-06-14 19:00:00+00:00	25000	25874.59
14	360	20.54	20.54	345	390	3.84	4.05		0.437698	367	0.410094	-0.52302	0.000475	8.151001	-67.9864	20.54	512394.8	2023-06-14 20:00:00+00:00	25000	24934.21
15	370	20.59	20.59	325	365	4.96	2.7		0.519878	344	0.495109	-0.44421	0.00039	8.074962	-82.7166	20.59	517377.1	2023-06-14 21:00:00+00:00	25000	25121.25
16	285	25.67	25.67	300	335	3.78	5.2		0.437111	317	0.415352	-0.477	0.000475	8.042842	-70.3284	15.66	391987.7	2023-06-14 22:00:00+00:00	25000	25034.98
17	265	26.67	26.67	250	280	4.04	2.5		0.426118	264	0.407117	-0.43074	0.00048	7.894376	-68.8697	15.66	393526.9	2023-06-14 23:00:00+00:00	25000	25128.6
18	210	27.67	27.67	200	230	2.95	2.5		0.406419	214	0.386824	-0.38868	0.000496	7.65561	-64.6114	14.66	369472.5	2023-06-15 00:00:00+00:00	25000	25206.51
19	210	27.67	27.67	260	290	2.51	6.25		0.411177	274	0.391972	-0.45924	0.000514	7.810582	-68.0336	14.66	367522.2	2023-06-15 01:00:00+00:00	25000	25064.79
20	210	27.67	27.67	235	275	5.3	4.89		0.405991	254	0.380082	-0.44868	0.000533	7.719519	-66.4309	14.66	367728.3	2023-06-15 02:00:00+00:00	25000	25083.79
21	210	27.67	27.67	250	275	2.52	2.5		0.398505	262	0.382194	-0.45878	0.000538	7.663217	-67.5877	14.66	367451.7	2023-06-15 03:00:00+00:00	25000	25065.35
22	210	27.67	27.67	265	295	3.4	2.5		0.405487	279	0.385771	-0.47904	0.000541	7.607957	-69.0567	14.66	366919.5	2023-06-15 04:00:00+00:00	25000	25028.15
23	210	27.67	27.67	265	295	5.27	2.5		0.411012	279	0.391098	-0.47781	0.000539	7.53245	-70.7019	14.66	367056.4	2023-06-15 05:00:00+00:00	25000	25031.42
24	210	27.67	27.67	330	365	3.73	2.5		0.439076	347	0.415473	-0.53149	0.000513	7.414229	-75.4382	14.66	365514.4	2023-06-15 06:00:00+00:00	25000	24927.95
25	355	28.67	28.67	315	345	2.46	2.5		0.455654	330	0.435322	-0.50215	0.000496	7.377556	-80.2898	14.66	366115.1	2023-06-15 07:00:00+00:00	25000	24983.8
26	330	28.68	28.68	330	360	3.38	2.5		0.474134	344	0.453576	-0.5065	0.000481	7.296521	-84.4981	14.67	366273.1	2023-06-15 08:00:00+00:00	25000	24974.26
27	360	30.36	30.36	345	375	5.28	2.7		0.483911	359	0.463095	-0.51716	0.000476	7.206177	-87.0555	16.35	407932.3	2023-06-15 09:00:00+00:00	25000	24947.71
28	410	30.37	30.37	395	415	4.68	11.88		0.526333	404	0.512272	-0.52585	0.000435	7.11158	-97.1477	16.36	407659.9	2023-06-15 10:00:00+00:00	25000	24923.51

Sorting and Cleaning of the Spot data

```
t_pp.py > ...
Click here to ask Blackbox to help you code faster
import os
import shutil
import glob
import pandas as pd

folder_path = 'final_dat'  # Replace with the path to your folder
put_data_folder = 'put_data'
call_data_folder = 'call_data'

# Create the 'put_data' and 'call_data' folders if they don't exist
os.makedirs(put_data_folder, exist_ok=True)
os.makedirs(call_data_folder, exist_ok=True)

# Iterate over each file in the folder
for file_path in glob.glob(folder_path + '/*.csv'):
    df = pd.read_csv(file_path)

    # Check if 'spot_price' column exists and is not empty
    if 'spot_price' in df and not df['spot_price'].isnull().all():
        # Remove rows where 'spot_price' is empty and save the file
        df = df.dropna(subset=['spot_price'])
        df.to_csv(file_path, index=False)
```

```
        # Determine the destination folder based on the file name
        if file_path.endswith('P.csv'):
            shutil.move(file_path, os.path.join(put_data_folder, os.path.basename(file_path)))
        elif file_path.endswith('C.csv'):
            shutil.move(file_path, os.path.join(call_data_folder, os.path.basename(file_path)))
    else:
        # Delete the file if 'spot_price' column is empty or doesn't exist
        os.remove(file_path)
```

- he code utilizes Python modules such as os, shutil, glob, and pandas for file and folder operations, file movement, and data manipulation.
- It defines folder paths for input data (folder_path), output folders for 'put' and 'call' data (put_data_folder and call_data_folder).
- Folders for 'put' and 'call' data are created using os.makedirs with the exist_ok=True parameter, ensuring folders are created only if they don't exist.
- It iterates over each CSV file in the specified folder, reads the data into a pandas DataFrame, and checks if the 'spot_price' column exists and is not entirely empty. If not, it removes rows with empty 'spot_price' values and saves the file.
- Based on the file name, it moves the processed CSV file to either the 'put data' or 'call data' folder. If the 'spot_price' column is empty or doesn't exist, the file is deleted.


call_data
File
Home
Share
View
This PC > New Volume (D:) > Major_1 > test > call_data
Search call_data
Quick access
Desktop
Downloads
Documents
Pictures
GG
put_data
sorted_on_symbol
test
OneDrive - Personal
This PC
Network
Name
Date modified
Type
Size
1,081 items

Final Code

Click here to ask Blackbox to help you code faster

```
import glob
import pandas as pd
import pandas_ta as pta

folder_path_C = 'call_data'
folder_path_P = 'put_data'

# Function to process each file and calculate RSI
def process_file(file_path):
    df = pd.read_csv(file_path)

    # Skip files with very few rows
    if len(df) <= 15:
        print(f"File {file_path} has 10 or fewer rows. Skipping.")
        return []
```

```
            elif row['RSI'] > 70 and flag:
                print("Out RSI", row['close'])
                dif_abv_70 = row['strike_price'] - row['spot_price']
                print(f"dif_abv_70: {dif_abv_70}")
                pnl.append(row['close'] - entry)
                flag = False

    return pnl

# Process all files in the 'call_data' folder
for file_path in glob.glob(folder_path_C + '/*.csv'):
    pnl = process_file(file_path)
    print(f"File: {file_path}, PNL: {pnl}, Sum of PNL_C: {sum(pnl)}")

# Process all files in the 'put_data' folder
for file_path in glob.glob(folder_path_P + '/*.csv'):
    pnl = process_file(file_path)
    print(f"File: {file_path}, PNL: {pnl}, Sum of PNL_P: {sum(pnl)}")
```

```
# Ensure necessary columns are present and numeric
if 'mark_iv' not in df.columns or 'strike_price' not in df.columns or 'spot_price' not in df.columns:
    print(f"Missing required columns in {file_path}. Skipping file.")
    return []

df['mark_iv'] = pd.to_numeric(df['mark_iv'], errors='coerce')
df['strike_price'] = pd.to_numeric(df['strike_price'], errors='coerce')
df['spot_price'] = pd.to_numeric(df['spot_price'], errors='coerce')

df['RSI'] = pta.rsi(df['mark_iv'], length=14)

flag = False
entry = 0
pnl = []
for index, row in df.iterrows():
    if pd.notna(row['RSI']):
        if row['RSI'] < 30 and not flag:
            print("In", row['close'])
            entry = row['close']
            dif_bel_30 = row['strike_price'] - row['spot_price']
            print(f"dif_bel_30: {dif_bel_30}")
            flag = True
```

Importing Libraries:

- glob: Library for file path expansion (used to iterate through CSV files in a directory).
- pandas: Data manipulation library.
- pandas_ta: Technical analysis library for pandas.

Define Folder Paths:

folder_path_C = 'call_data'
folder_path_P = 'put_data'

- These variables store the paths to folders containing call and put option data files, respectively.

Processing Function:

- A function named process_file is defined to read and process each CSV file in the given folder paths.
- The function calculates the Relative Strength Index (RSI) based on the 'mark_iv' column.
- It also checks if the file has an adequate number of rows and required columns.
- The RSI values are used to generate trading signals based on RSI thresholds (30 and 70).

The Relative Strength Index (RSI) is calculated using the pandas_ta:

The code utilizes the **pta.rsi** function from the **pandas_ta** library to calculate the RSI for the 'markiv' column in the DataFrame

  - Calculating the average gain and average loss for the specified period (14 in this case).
  - Smoothing the average gain and average loss using a weighting factor.
  - Calculating the Relative Strength (RS) as the ratio of the average gain to the average loss.
  - Transforming the RS to a value between 0 and 100 using the formula: RSI = 100 - (100 / (1 + RS)).

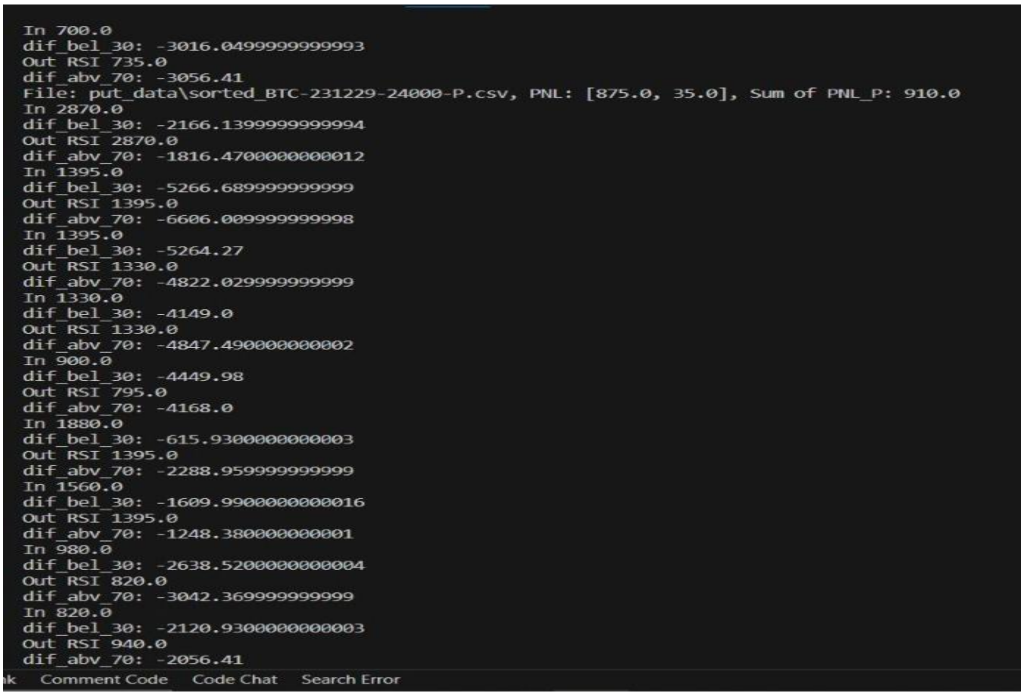
- The code iterates through each row in the DataFrame using df.iterrows().

It checks two conditions related to the RSI value:

- If row['RSI'] is less than 30 and the flag is not set (not flag), it prints "In" along with the closing price (row['close']).
- Sets entry to the closing price.
- Calculates the difference between the strike price and the spot price (dif_bel_30).
- Prints the difference (dif_bel_30).
- Sets the flag to True.

- If row['RSI'] is greater than 70 and the flag is set (flag), it prints "Out RSI" along with the closing price.
- Calculates the difference between the strike price and the spot price (dif_abv_70).
- Prints the difference (dif_abv_70).
- Appends the profit or loss (row['close'] - entry) to the pnl list.
- Sets the flag to False

Return Statement:
The function returns the pnl list, which accumulates the profit or loss for each trade based on the RSI conditions.



```
In 700.0
dif_bel_30: -3016.0499999999993
Out RSI 735.0
dif_abv_70: -3056.41
File: put_data\sorted_BTC-231229-24000-P.csv, PNL: [875.0, 35.0], Sum of PNL_P: 910.0
In 2870.0
dif_bel_30: -2166.1399999999994
Out RSI 2870.0
dif_abv_70: -1816.4700000000012
In 1395.0
dif_bel_30: -5266.689999999999
Out RSI 1395.0
dif_abv_70: -6606.009999999998
In 1395.0
dif_bel_30: -5264.27
Out RSI 1330.0
dif_abv_70: -4822.029999999999
In 1330.0
dif_bel_30: -4149.0
Out RSI 1330.0
dif_abv_70: -4847.490000000002
In 900.0
dif_bel_30: -4449.98
Out RSI 795.0
dif_abv_70: -4168.0
In 1880.0
dif_bel_30: -615.9300000000003
Out RSI 1395.0
dif_abv_70: -2288.959999999999
In 1560.0
dif_bel_30: -1609.9900000000016
Out RSI 1395.0
dif_abv_70: -1248.380000000001
In 980.0
dif_bel_30: -2638.5200000000004
Out RSI 820.0
dif_abv_70: -3042.369999999999
In 820.0
dif_bel_30: -2120.9300000000003
Out RSI 940.0
dif_abv_70: -2056.41
```


Conclusion

Project focused on analyzing options trading data using the Relative Strength Index (RSI) as a key indicator for making trading decisions. The code processes multiple CSV files containing financial data, calculates RSI values using the pandas_ta library, and implements a straightforward trading strategy. The strategy involves buying options when RSI is below 30 and there is no active position, and selling when RSI exceeds 70 and there is an existing position. Profit and loss for each trade are tracked, and the code outputs information about individual trades and the cumulative profit and loss for each category (call or put data). While the code provides a foundation for an RSI-based trading strategy, it requires refinement, error handling, and additional documentation for robustness and clarity.

References

1. "The Black-Scholes Model" by Black & Scholes (1973) [1]
2. "Options Pricing: A Simplified Approach" by Robert Geske and H. E. Johnson (1984) [2]
3. "Implied Volatility and Future Portfolio Returns" by Jackwerth (2000) [3]
4. "Option Volatility and Pricing" by Sheldon Natenberg (1994) [4]
5. "Forecasting Volatility in Financial Markets: A Review" by Andersen, T.G., T. Bollerslev, and F.X. Diebold (2007) [5]
6. "Risk Management in Financial Markets" by Michael Ong (2004) [6]