Money Machine

List of Binance coin listing announcements

1. skale-network (SKL) - 2020-11-30 10:00
2. celo (CELO) - 2021-01-05 04:36
3. gyen () - 2021-05-12 03:00
4. arweave (AR) - 2021-05-14 06:36
5. mask-network (MASK) - 2021-05-25 05:10
6. livepeer (LPT) - 2021-05-28 04:12
7. nucypher (NU) - 2021-06-04 04:27
8. gitcoin (GTC) - 2021-06-10 08:35
9. tornado-cash (TORN) - 2021-06-11 04:32
10. keep-network (KEEP) - 2021-06-17 04:34
11. klaytn (KLAY) - 2021-06-24 05:42
12. barnbridge (BOND) - 2021-07-05 04:05
13. enzyme (MLN) - 2021-07-05 04:05
14. clover-finance (CLV) - 2021-07-29 04:01
15. quant (QNT) - 2021-07-29 04:01
16. flow (FLOW) - 2021-07-30 09:00
17. mobox (MBOX) - 2021-08-19 04:59
18. wax (WAXP) - 2021-08-23 04:30
19. tribe (TRIBE) - 2021-08-24 03:59
20. gnosis (GNO) - 2021-08-30 04:01
21. list-gala (GALA) - 2021-09-13 04:00
22. illuvium (ILV) - 2021-09-22 05:26
23. bonfida (FIDA) - 2021-09-30 04:29
24. radicle (RAD) - 2021-10-07 05:24
25. superrare (RARE) - 2021-10-11 04:03
26. moonriver (MOVR) - 2021-11-08 02:59
27. benqi (QI) - 2021-11-15 02:58
28. jasmycoin (JASMY) - 2021-11-22 08:59

29. playdapp (PLA) - 2021-11-23 02:59

30. amp (AMP) - 2021-11-23 02:59

31. vulcan-forged-pyr (PYR) - 2021-11-26 04:45

32. render (RNDR) - 2021-11-27 06:57

33. alchemix (ALCX) - 2021-11-30 02:59

34. merit-circle (MC) - 2021-12-02 05:02

35. anyswap (ANY) - 2021-12-04 03:01

36. biconomy (BICO) - 2021-12-09 02:56

37. flux (FLUX) - 2021-12-10 02:58

38. highstreet (HIGH) - 2021-12-17 02:56

39. terrausd (UST) - 2021-12-24 02:58

40. spell-token (SPELL) - 2021-12-24 02:58

41. joe (JOE) - 2021-12-28 05:56

42. api3 (API3) - 2022-01-21 07:28

43. acala (ACA) - 2022-01-25 03:02

44. anchor-protocol (ANC) - 2022-01-25 05:52

45. kadena (KDA) - 2022-03-11 06:17

46. apecoin (APE) - 2022-03-17 08:02

47. nexo (NEXO) - 2022-04-29 07:58

48. mobilecoin (MOB) - 2022-04-29 07:58

49. optimism (OP) - 2022-06-01 03:00

50. aptos (APT) - 2022-10-18 03:04

51. neutron (NTRN) - 2023-10-10 09:45

52. ordi (ORDI) - 2023-11-07 07:44

53. blur (BLUR) - 2023-11-24 07:09

54. jito (JTO) - 2023-12-07 12:55

55. brc-20-sats (1000SATS) - 2023-12-12 07:09

56. bonk (BONK) - 2023-12-15 04:58

57. sleepless-ai (AI) - 2024-01-08 09:30

58. xai (XAI) - 2024-01-09 12:00

59. manta (MANTA) - 2024-01-18 12:10

60. altlayer (ALT) - 2024-01-25 12:10

61. jupiter (JUP) - 2024-01-31 17:40

62. pyth-network (PYTH) - 2024-02-02 14:20

63. ronin (RONIN) - 2024-02-05 15:32

64. dymension (DYM) - 2024-02-07 07:05

65. pixels (PIXEL) - 2024-02-19 12:00

66. axelar (AXL) - 2024-03-01 13:28

67. dogwifhat (WIF) - 2024-03-05 16:20

68. metis (METIS) - 2024-03-11 14:00

69. aevo (AEVO) - 2024-03-13 11:55

70. saga (SAGA) - 2024-04-09 10:41

71. omni-network (OMNI) - 2024-04-17 14:40

72. renzo (REZ) - 2024-04-30 14:32

73. notcoin (NOT) - 2024-05-16 14:30

74. io-net (IO) - 2024-06-11 14:30

75. zksync (ZK) - 2024-06-17 12:30

76. layerzero (ZRO) - 2024-06-20 15:30

77. gravity (G) - 2024-07-19 10:30

78. banana-gun (BANANA) - 2024-07-20 11:35

79. render (RENDER) - 2024-07-26 10:30

80. toncoin (TON) - 2024-08-08 12:30

81. eigen () - 2024-09-30 10:19

82. lumia () - 2024-10-18 10:30

83. thena (THE) - 2024-11-27 10:30

List of Binance coin listing announcements to integrate them in a Python-Code

```
{"name": "skale-network", "ticker": "SKL", "datetime": datetime(2020, 11, 30, 10, 0)},
{"name": "celo", "ticker": "CELO", "datetime": datetime(2021, 1, 5, 4, 36)},
{"name": "gyen", "ticker": "", "datetime": datetime(2021, 5, 12, 3, 0)},
{"name": "arweave", "ticker": "AR", "datetime": datetime(2021, 5, 14, 6, 36)},
{"name": "mask-network", "ticker": "MASK", "datetime": datetime(2021, 5, 25, 5, 10)},
{"name": "livepeer", "ticker": "LPT", "datetime": datetime(2021, 5, 28, 4, 12)},
{"name": "nucypher", "ticker": "NU", "datetime": datetime(2021, 6, 4, 4, 27)},
{"name": "gitcoin", "ticker": "GTC", "datetime": datetime(2021, 6, 10, 8, 35)},
{"name": "tornado-cash", "ticker": "TORN", "datetime": datetime(2021, 6, 11, 4, 32)},
{"name": "keep-network", "ticker": "KEEP", "datetime": datetime(2021, 6, 17, 4, 34)},
{"name": "klaytn", "ticker": "KLAY", "datetime": datetime(2021, 6, 24, 5, 42)},
{"name": "barnbridge", "ticker": "BOND", "datetime": datetime(2021, 7, 5, 4, 5)},
{"name": "enzyme", "ticker": "MLN", "datetime": datetime(2021, 7, 5, 4, 5)},
{"name": "clover-finance", "ticker": "CLV", "datetime": datetime(2021, 7, 29, 4, 1)},
{"name": "quant", "ticker": "QNT", "datetime": datetime(2021, 7, 29, 4, 1)},
{"name": "flow", "ticker": "FLOW", "datetime": datetime(2021, 7, 30, 9, 0)},
{"name": "mobox", "ticker": "MBOX", "datetime": datetime(2021, 8, 19, 4, 59)},
{"name": "wax", "ticker": "WAXP", "datetime": datetime(2021, 8, 23, 4, 30)},
{"name": "tribe", "ticker": "TRIBE", "datetime": datetime(2021, 8, 24, 3, 59)},
{"name": "gnosis", "ticker": "GNO", "datetime": datetime(2021, 8, 30, 4, 1)},
{"name": "list-gala", "ticker": "GALA", "datetime": datetime(2021, 9, 13, 4, 0)},
{"name": "illuvium", "ticker": "ILV", "datetime": datetime(2021, 9, 22, 5, 26)},
{"name": "bonfida", "ticker": "FIDA", "datetime": datetime(2021, 9, 30, 4, 29)},
{"name": "radicle", "ticker": "RAD", "datetime": datetime(2021, 10, 7, 5, 24)},
{"name": "superrare", "ticker": "RARE", "datetime": datetime(2021, 10, 11, 4, 3)},
{"name": "moonriver", "ticker": "MOVR", "datetime": datetime(2021, 11, 8, 2, 59)},
{"name": "benqi", "ticker": "QI", "datetime": datetime(2021, 11, 15, 2, 58)},
{"name": "jasmycoin", "ticker": "JASMY", "datetime": datetime(2021, 11, 22, 8, 59)},
{"name": "playdapp", "ticker": "PLA", "datetime": datetime(2021, 11, 23, 2, 59)},
{"name": "amp", "ticker": "AMP", "datetime": datetime(2021, 11, 23, 2, 59)},
{"name": "vulcan-forged-pyr", "ticker": "PYR", "datetime": datetime(2021, 11, 26, 4, 45)},
{"name": "render", "ticker": "RNDR", "datetime": datetime(2021, 11, 27, 6, 57)},
{"name": "alchemix", "ticker": "ALCX", "datetime": datetime(2021, 11, 30, 2, 59)},
{"name": "merit-circle", "ticker": "MC", "datetime": datetime(2021, 12, 2, 5, 2)},
{"name": "anyswap", "ticker": "ANY", "datetime": datetime(2021, 12, 4, 3, 1)},
{"name": "biconomy", "ticker": "BICO", "datetime": datetime(2021, 12, 9, 2, 56)},
{"name": "flux", "ticker": "FLUX", "datetime": datetime(2021, 12, 10, 2, 58)},
{"name": "highstreet", "ticker": "HIGH", "datetime": datetime(2021, 12, 17, 2, 56)},
{"name": "terrausd", "ticker": "UST", "datetime": datetime(2021, 12, 24, 2, 58)},
{"name": "spell-token", "ticker": "SPELL", "datetime": datetime(2021, 12, 24, 2, 58)},
{"name": "joe", "ticker": "JOE", "datetime": datetime(2021, 12, 28, 5, 56)},
{"name": "api3", "ticker": "API3", "datetime": datetime(2022, 1, 21, 7, 28)},
{"name": "acala", "ticker": "ACA", "datetime": datetime(2022, 1, 25, 3, 2)},
{"name": "anchor-protocol", "ticker": "ANC", "datetime": datetime(2022, 1, 25, 5, 52)},
{"name": "kadena", "ticker": "KDA", "datetime": datetime(2022, 3, 11, 6, 17)},
{"name": "apecoin", "ticker": "APE", "datetime": datetime(2022, 3, 17, 8, 2)},
{"name": "nexo", "ticker": "NEXO", "datetime": datetime(2022, 4, 29, 7, 58)},
{"name": "mobilecoin", "ticker": "MOB", "datetime": datetime(2022, 4, 29, 7, 58)},
{"name": "optimism", "ticker": "OP", "datetime": datetime(2022, 6, 1, 3, 0)},
{"name": "aptos", "ticker": "APT", "datetime": datetime(2022, 10, 18, 3, 4)},
{"name": "neutron", "ticker": "NTRN", "datetime": datetime(2023, 10, 10, 9, 45)},
{"name": "ordi", "ticker": "ORDI", "datetime": datetime(2023, 11, 7, 7, 44)},
{"name": "blur", "ticker": "BLUR", "datetime": datetime(2023, 11, 24, 7, 9)},
{"name": "jito", "ticker": "JTO", "datetime": datetime(2023, 12, 7, 12, 55)},
{"name": "brc-20-sats", "ticker": "1000SATS", "datetime": datetime(2023, 12, 12, 7, 9)},
{"name": "bonk", "ticker": "BONK", "datetime": datetime(2023, 12, 15, 4, 58)},
{"name": "sleepless-ai", "ticker": "AI", "datetime": datetime(2024, 1, 8, 9, 30)},
{"name": "xai", "ticker": "XAI", "datetime": datetime(2024, 1, 9, 12, 0)},
{"name": "manta", "ticker": "MANTA", "datetime": datetime(2024, 1, 18, 12, 10)},
{"name": "altlayer", "ticker": "ALT", "datetime": datetime(2024, 1, 25, 12, 10)},
{"name": "jupiter", "ticker": "JUP", "datetime": datetime(2024, 1, 31, 17, 40)},
{"name": "pyth-network", "ticker": "PYTH", "datetime": datetime(2024, 2, 2, 14, 20)},
{"name": "ronin", "ticker": "RONIN", "datetime": datetime(2024, 2, 5, 15, 32)},
{"name": "dymension", "ticker": "DYM", "datetime": datetime(2024, 2, 7, 7, 5)},
{"name": "pixels", "ticker": "PIXEL", "datetime": datetime(2024, 2, 19, 12, 0)},
{"name": "axelar", "ticker": "AXL", "datetime": datetime(2024, 3, 1, 13, 28)},
{"name": "dogwifhat", "ticker": "WIF", "datetime": datetime(2024, 3, 5, 16, 20)},
{"name": "metis", "ticker": "METIS", "datetime": datetime(2024, 3, 11, 14, 0)},
{"name": "aevo", "ticker": "AEVO", "datetime": datetime(2024, 3, 13, 11, 55)},
{"name": "saga", "ticker": "SAGA", "datetime": datetime(2024, 4, 9, 10, 41)},
{"name": "omni-network", "ticker": "OMNI", "datetime": datetime(2024, 4, 17, 14, 40)},
{"name": "renzo", "ticker": "REZ", "datetime": datetime(2024, 4, 30, 14, 32)},
{"name": "notcoin", "ticker": "NOT", "datetime": datetime(2024, 5, 16, 14, 30)},
{"name": "io-net", "ticker": "IO", "datetime": datetime(2024, 6, 11, 14, 30)},
{"name": "zksync", "ticker": "ZK", "datetime": datetime(2024, 6, 17, 12, 30)},
{"name": "layerzero", "ticker": "ZRO", "datetime": datetime(2024, 6, 20, 15, 30)},
{"name": "gravity", "ticker": "G", "datetime": datetime(2024, 7, 19, 10, 30)},
{"name": "banana-gun", "ticker": "BANANA", "datetime": datetime(2024, 7, 20, 11, 35)},
{"name": "render", "ticker": "RENDER", "datetime": datetime(2024, 7, 26, 10, 30)},
{"name": "toncoin", "ticker": "TON", "datetime": datetime(2024, 8, 8, 12, 30)},
{"name": "eigen", "ticker": "", "datetime": datetime(2024, 9, 30, 10, 19)},
{"name": "lumia", "ticker": "", "datetime": datetime(2024, 10, 18, 10, 30)},
{"name": "thena", "ticker": "THE", "datetime": datetime(2024, 11, 27, 10, 30)},
]
```

Python-Code for pulling data from the Binance API

```
from binance.client import Client
import pandas as pd

# API key (if needed)
API_KEY = "your_api_key"  # Replace with your API key
API_SECRET = "your_api_secret"  # Replace with your API secret

# Initialize Binance client
client = Client(API_KEY, API_SECRET)

# List of coins with their listing dates
coins = [
    {"symbol": "LUMIAUSDT", "listed_date": "2024-10-18 10:30"},
    {"symbol": "EIGENUSDT", "listed_date": "2024-09-30 10:19"},
    {"symbol": "THEUSDT", "listed_date": "2024-11-27 10:30"},
    {"symbol": "BONKUSDT", "listed_date": "2023-12-15 04:58"},
    {"symbol": "1000SATSUSDT", "listed_date": "2023-12-12 07:09"},
    {"symbol": "JTOUSDT", "listed_date": "2023-12-07 12:55"},
    {"symbol": "BLURUSDT", "listed_date": "2023-11-24 07:09"},
    {"symbol": "ORDIUSDT", "listed_date": "2023-11-07 07:44"},
    {"symbol": "NTRNUSDT", "listed_date": "2023-10-10 09:45"},
    {"symbol": "APTUSDT", "listed_date": "2022-10-18 03:04"},
    {"symbol": "OPUSDT", "listed_date": "2022-06-01 03:00"},
    {"symbol": "MOBUSDT", "listed_date": "2022-04-29 07:58"},
    {"symbol": "NEXOUSDT", "listed_date": "2022-04-29 07:58"},
    {"symbol": "APEUSDT", "listed_date": "2022-03-17 08:02"},
    {"symbol": "KDAUSDT", "listed_date": "2022-03-11 06:17"},
]

def get_daily_data(symbol, start_date):
    """
    Fetches daily kline (candlestick) data from Binance starting 
from the listing date.

    :param symbol: The coin symbol, e.g., 'BTCUSDT'.
    :param start_date: Start date (in the format '1 Jan, 2021').
    :return: Pandas DataFrame with kline data.
    """
    try:
        # Fetch daily klines
        klines = client.get_historical_klines(
            symbol=symbol,
            interval=Client.KLINE_INTERVAL_1DAY,
            start_str=start_date
        )

        # Convert to a DataFrame
        data = pd.DataFrame(klines, columns=[
            "Open time", "Open", "High", "Low", "Close", "Volume",
            "Close time", "Quote asset volume", "Number of 
trades", 
            "Taker buy base asset volume", "Taker buy quote asset 
volume", "Ignore"
        ])

        # Convert timestamps
        data["Open time"] = pd.to_datetime(data["Open time"], 
unit='ms')
        data["Close time"] = pd.to_datetime(data["Close time"], 
unit='ms')

        # Clean data (keep only relevant columns)
        data = data[["Open time", "Open", "High", "Low", "Close", 
"Volume"]]
        data.set_index("Open time", inplace=True)

        return data
    except Exception as e:
        print(f"Error fetching data for {symbol}: {e}")
        return None

# Fetch data for all coins
for coin in coins:
    symbol = coin["symbol"]
    listed_date = coin["listed_date"]

    print(f"Fetching data for {symbol}, listing date: 
{listed_date}")
    data = get_daily_data(symbol, listed_date)

    if data is not None:
        # Save data as CSV
        filename = f"{symbol}_daily_data.csv"
        data.to_csv(filename)
        print(f"Data for {symbol} saved in {filename}")
```

Python-Code to check if the data is clean

```
import os
import pandas as pd

# Path to the all_coins folder on the Desktop
folder_path = os.path.expanduser("~/Desktop/all_coins")

def check_csv_files(folder_path):
    # List for reports
    reports = []

    # Iterate through all CSV files in the folder
    for file in os.listdir(folder_path):
        if file.endswith(".csv"):
            file_path = os.path.join(folder_path, file)
            try:
                # Read the CSV file
                df = pd.read_csv(file_path)

                # Initialize the report
                report = {
                    "file": file,
                    "null_values": [],
                    "missing_values_rows": [],
                    "duplicate_dates": [],
                    "missing_dates": [],
                    "low_greater_than_high": [],
                    "high_less_than_low": []
                }

                # Check for null values in columns
                if df.isnull().values.any():
                    null_cols =
df.columns[df.isnull().any()].tolist()
                    report["null_values"] = null_cols

                # Check for rows with missing values
                if df.isnull().any(axis=1).any():
                    missing_rows =
df[df.isnull().any(axis=1)].index.tolist()
                    report["missing_values_rows"] = missing_rows

                # Check for duplicate date values, if a "Date" or
"Time" column exists
                date_col = None
                for col in df.columns:
                    if "date" in col.lower() or "time" in
col.lower():
                        date_col = col
                        break
                if date_col:
                    if df[date_col].duplicated().any():
                        duplicate_dates =
df[df[date_col].duplicated()][date_col].tolist()
                        report["duplicate_dates"] =
duplicate_dates

                    # Check for missing data (gaps in timestamps)
                    df[date_col] = pd.to_datetime(df[date_col])
                    df = df.sort_values(by=date_col)
                    full_range =
pd.date_range(start=df[date_col].min(), end=df[date_col].max(),
freq="D")
                    missing_dates =
full_range.difference(df[date_col])
                    report["missing_dates"] =
missing_dates.strftime('%Y-%m-%d').tolist()

                # Check for low values greater than high values
                if "low" in df.columns and "high" in df.columns:
                    low_greater = df[df["low"] > df["high"]]
                    if not low_greater.empty:
                        report["low_greater_than_high"] =
low_greater.index.tolist()

                # Check for high values less than low values
                if "low" in df.columns and "high" in df.columns:
                    high_less = df[df["high"] < df["low"]]
                    if not high_less.empty:
                        report["high_less_than_low"] =
high_less.index.tolist()

                reports.append(report)

            except Exception as e:
                print(f"Error processing file {file}: {e}")

    # Output the reports
    for report in reports:
        print(f"Report for file: {report['file']}")
        print(f"  Null values in columns:
{report['null_values']}")
        print(f"  Missing values in rows (indices):
{report['missing_values_rows']}")
        print(f"  Duplicate date values:
{report['duplicate_dates']}")
        print(f"  Missing data: {report['missing_dates']}")
        print(f"  Low > High errors in rows: 
{report['low_greater_than_high']}")
        print(f"  High < Low errors in rows: 
{report['high_less_than_low']}")
        print("-" * 50)

# Execute the function
check_csv_files(folder_path)
```

Python-Code to analyze how many coins were falling

```python
import os
import pandas as pd

def analyze_csv_files(folder_path):
    """
    Analyzes all CSV files in a folder and calculates whether the 
stock price
    has risen or fallen based on the 'Close' value.

    Outputs a summary of the rising and falling prices at the end.
    """
    # Ensure the folder exists
    if not os.path.exists(folder_path):
        print(f"The folder {folder_path} does not exist.")
        return

    # List all CSV files in the folder
    csv_files = [file for file in os.listdir(folder_path) if 
file.endswith('.csv')]
    if not csv_files:
        print("No CSV files found in the specified folder.")
        return

    # Counters for rising and falling prices
    risen_count = 0
    fallen_count = 0

    for csv_file in csv_files:
        file_path = os.path.join(folder_path, csv_file)
        print(f"Analyzing file: {csv_file}")

        # Load the CSV file
        try:
            df = pd.read_csv(file_path)

            # Check if the 'Close' column exists
            if 'Close' not in df.columns:
                print(f"The file {csv_file} does not contain a 
'Close' column. Skipping.")
                continue

            # Calculate the price change
            first_close = df['Close'].iloc[0]
            last_close = df['Close'].iloc[-1]
            percent_change = ((last_close - first_close) / 
first_close) * 100
```

```
            # Determine status
            if percent_change > 0:
                risen_count += 1
                print(f"{csv_file}: Risen ({percent_change:.2f}
%)")
            else:
                fallen_count += 1
                print(f"{csv_file}: Fallen ({percent_change:.2f}
%)")

        except Exception as e:
            print(f"Error processing the file {csv_file}: {e}")

    # Display summary
    print("\nSummary:")
    print(f"Rising prices: {risen_count}")
    print(f"Falling prices: {fallen_count}")

# Main function
if __name__ == "__main__":
    folder_path = "all_coins"  # Change this to the path of your 
folder containing the CSV files
    analyze_csv_files(folder_path)
```

Python-Code to calculate the average percentage drop of all coins since day 1

```
import os
import pandas as pd
from datetime import datetime

def analyze_total_average_change(folder_path):
    """
    Analyzes all CSV files in a folder and calculates the average 
percentage change
    for all coins based on the 'Close' value. It uses the first 
and last available
    days in each file.

    Outputs the overall average percentage change.
    """
    # Ensure the folder exists
    if not os.path.exists(folder_path):
        print(f"The folder {folder_path} does not exist.")
        return

    # List all CSV files in the folder
    csv_files = [file for file in os.listdir(folder_path) if 
file.endswith('.csv')]
    if not csv_files:
        print("No CSV files found in the specified folder.")
        return

    # List to store percentage changes
    percent_changes = []

    for csv_file in csv_files:
        file_path = os.path.join(folder_path, csv_file)
        print(f"Analyzing file: {csv_file}")

        # Load the CSV file
        try:
            df = pd.read_csv(file_path)

            # Check if the required columns exist
            if 'Open time' not in df.columns or 'Close' not in 
df.columns:
                print(f"The file {csv_file} does not contain the 
required columns 'Open time' or 'Close'. Skipping.")
                continue

            # Convert 'Open time' to datetime format
            df['Open time'] = pd.to_datetime(df['Open time'], 
errors='coerce')
```

```
            # Sort data by date
            df = df.sort_values(by='Open 
time').dropna(subset=['Open time'])

            # Ensure the file is not empty
            if df.empty or len(df) < 2:
                print(f"The file {csv_file} does not have enough 
data. Skipping.")
                continue

            # Find the first and last 'Close' values
            first_close = df['Close'].iloc[0]
            last_close = df['Close'].iloc[-1]

            if first_close == 0:  # Avoid division by zero
                print(f"The file {csv_file} has a 'Close' value of 
0 at the start. Skipping.")
                continue

            # Calculate percentage change
            percent_change = ((last_close - first_close) / 
first_close) * 100
            percent_changes.append(percent_change)

        except Exception as e:
            print(f"Error processing the file {csv_file}: {e}")

    # Calculate the average
    if percent_changes:
        average_change = sum(percent_changes) / 
len(percent_changes)
        print("\nOverall average percentage change:")
        print(f"{average_change:.2f}%")
    else:
        print("No valid data found for calculation.")

# Main function
if __name__ == "__main__":
    folder_path = "all_coins"  # Change this to the path of your 
folder containing the CSV files
    analyze_total_average_change(folder_path)
```

Python-Code to calculate the average percentage drop of all coins since day 1 (excluding the following coins: first date is younger than 12 months)

```python
import os
import pandas as pd
from datetime import datetime, timedelta

def analyze_total_average_change(folder_path):
    """
    Analyzes all CSV files in a folder and calculates the average 
percentage change
    for all coins based on the 'Close' value. Only includes data 
at least 12 months old.

    Outputs the overall average percentage change.
    """
    # Ensure the folder exists
    if not os.path.exists(folder_path):
        print(f"The folder {folder_path} does not exist.")
        return

    # List all CSV files in the folder
    csv_files = [file for file in os.listdir(folder_path) if 
file.endswith('.csv')]
    if not csv_files:
        print("No CSV files found in the specified folder.")
        return

    # List to store percentage changes
    percent_changes = []

    # Define the cutoff date (12 months ago from today)
    cutoff_date = datetime.now() - timedelta(days=365)

    for csv_file in csv_files:
        file_path = os.path.join(folder_path, csv_file)
        print(f"Analyzing file: {csv_file}")

        # Load the CSV file
        try:
            df = pd.read_csv(file_path)

            # Check if the required columns exist
            if 'Open time' not in df.columns or 'Close' not in 
df.columns:
                print(f"The file {csv_file} does not contain the 
required columns 'Open time' or 'Close'. Skipping.")
                continue

            # Convert 'Open time' to datetime format
            df['Open time'] = pd.to_datetime(df['Open time'], 
errors='coerce')

            # Filter rows with valid dates
            df = df.dropna(subset=['Open time'])

            # Filter data older than 12 months
            df = df[df['Open time'] <= cutoff_date]

            # Sort data by date
            df = df.sort_values(by='Open time')

            # Ensure the file has enough data
            if df.empty or len(df) < 2:
                print(f"The file {csv_file} does not have enough 
data older than 12 months. Skipping.")
                continue

            # Find the first and last 'Close' values
            first_close = df['Close'].iloc[0]
            last_close = df['Close'].iloc[-1]

            if first_close == 0:  # Avoid division by zero
                print(f"The file {csv_file} has a 'Close' value of 
0 at the start. Skipping.")
                continue

            # Calculate percentage change
            percent_change = ((last_close - first_close) / 
first_close) * 100
            percent_changes.append(percent_change)

        except Exception as e:
            print(f"Error processing the file {csv_file}: {e}")

    # Calculate the average
    if percent_changes:
        average_change = sum(percent_changes) / 
len(percent_changes)
        print("\nOverall average percentage change:")
        print(f"{average_change:.2f}%")
    else:
        print("No valid data found for calculation.")

# Main function
if __name__ == "__main__":
    folder_path = "all_coins"  # Change this to the path of your 
folder containing the CSV files
    analyze_total_average_change(folder_path)
```

Python-Code to calculate the average percentage drop of coins 18 days after day 1

```
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import pandas as pd
import glob
import os
import numpy as np

# Adjust paths
desktop_path = os.path.expanduser("~/Desktop")
csv_folder_path = os.path.join(desktop_path, "all_coins")
visuals_folder_path = os.path.join(desktop_path, "visuals")

# Ensure the visuals folder exists
os.makedirs(visuals_folder_path, exist_ok=True)

# Constants
num_days = 18  # Only the first 18 days

# List to store percentage changes
percentage_changes = []

# Find all CSV files in the folder
csv_files = glob.glob(os.path.join(csv_folder_path, "*.csv"))

# Calculate percentage changes
for day in range(num_days):
    daily_changes = []
    for file in csv_files:
        df = pd.read_csv(file)
        if len(df) > day:  # Check if the day exists in the 
dataset
            start_price = df["Close"].iloc[0]  # Starting price of 
the coin
            if start_price > 0:  # Avoid division by zero
                current_price = df["Close"].iloc[day]
                percentage_change = ((current_price - start_price) 
/ start_price) * 100  # Percentage change
                daily_changes.append(percentage_change)
    if daily_changes:
        percentage_changes.append(np.mean(daily_changes))  # 
Average percentage change
    else:
        percentage_changes.append(None)

# Create the plot
fig, ax = plt.subplots(figsize=(14, 7))

# Set background color (black)
fig.patch.set_facecolor("black")
```

```
ax.set_facecolor("black")

# Calculate Y-axis range
y_min = min([val for val in percentage_changes if val is not
None]) * 1.1
y_max = max([val for val in percentage_changes if val is not
None]) * 1.1
ax.set_xlim(0, len(percentage_changes))
ax.set_ylim(y_min, y_max)

# Gradient for the line (Pink -> Orange -> Yellow)
colors = ["magenta", "orange", "yellow"]
cmap = mcolors.LinearSegmentedColormap.from_list("gradient",
colors, N=len(percentage_changes))
line_colors = cmap(np.linspace(0, 1, len(percentage_changes)))

# Draw the line with gradient
for i in range(len(percentage_changes) - 1):
    if percentage_changes[i] is not None and percentage_changes[i
+ 1] is not None:
        ax.plot(
            [i, i + 1],
            [percentage_changes[i], percentage_changes[i + 1]],
            color=line_colors[i],
            linewidth=3,
            zorder=1,
        )

# Title and axis labels
title_font = {"fontsize": 16, "color": "white"}
label_font = {"fontsize": 14, "color": "lightgray"}
ax.set_title("Average Percentage Change Over First 18 Days",
**title_font, pad=20)
ax.set_xlabel("Days Since Listing", **label_font, labelpad=15)
ax.set_ylabel("Average Percentage Change (%)", **label_font,
labelpad=15)

# Set axis ticks to light color
ax.tick_params(colors="lightgray")

# Remove borders for a cleaner look
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.spines["bottom"].set_visible(False)

# Adjust layout and save the plot
plt.tight_layout()
output_path = os.path.join(visuals_folder_path,
"chart_without_leverage.png")
plt.savefig(output_path, dpi=300, bbox_inches="tight",
transparent=True)
```

```python
plt.close()

print(f"The chart without leverage has been saved in the 'visuals' 
folder: {output_path}")
```

Python-Code to visualize the coins one by one

```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

# Pfade und Einstellungen
desktop_path = Path.home() / "Desktop"
input_dir = desktop_path / "all_coins"  # Ordner für die CSV-
Dateien
output_dir = desktop_path / "visuals"  # Ordner für die
Visualisierungen
output_dir.mkdir(exist_ok=True)  # Erstelle den "visuals"-Ordner,
falls er nicht existiert

# Alle CSV-Dateien im Ordner "all_coins"
file_paths = list(input_dir.glob("*.csv"))

if not file_paths:
    print("Keine CSV-Dateien im Ordner 'all_coins' gefunden.")
else:
    # Verarbeite jede Datei
    for file_path in file_paths:
        try:
            # CSV-Datei laden
            df = pd.read_csv(file_path)

            # Stelle sicher, dass die Spalte 'Open time' im
Datetime-Format vorliegt
            if 'Open time' in df.columns:
                df['Open time'] = pd.to_datetime(df['Open time'])
# Konvertiere in Datetime-Format
                df = df.rename(columns={'Open time': 'Date'})  #
Standardisiere auf 'Date'
            else:
                print(f"'Open time'-Spalte nicht in {file_path}
gefunden. Überspringe die Datei.")
                continue

            # Nach Datum sortieren
            df = df.sort_values(by='Date')

            # Dateiname für die Grafik
            base_name = file_path.stem
            plot_file = output_dir / f"{base_name}_trend_plot.png"

            # Schlusskurse plotten
            if 'Close' in df.columns:
                plt.figure(figsize=(10, 6))
                plt.plot(df['Date'], df['Close'], marker='o', 
linestyle='-', label='Close Price')
                plt.title(f"Trend Over Time - {base_name}")
                plt.xlabel("Date")
                plt.ylabel("Close Price")
                plt.grid()
                plt.legend()
                plt.savefig(plot_file)
                plt.close()

                print(f"Trend-Grafik gespeichert für {base_name}: 
{plot_file}")
            else:
                print(f"'Close'-Spalte nicht in {file_path} 
gefunden. Überspringe die Datei.")
        except Exception as e:
            print(f"Fehler beim Verarbeiten der Datei {file_path}: 
{e}")
```

Python-Code to visualize all the coins in one graphic

```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

# Paths and settings
desktop_path = Path.home() / "Desktop"
input_dir = desktop_path / "all_coins"  # Folder containing the 
CSV files (changed to "all_coins")
output_dir = desktop_path / "visuals"  # Folder for the 
visualizations
output_dir.mkdir(exist_ok=True)  # Create the "visuals" folder if 
it doesn't exist

# All CSV files in the "all_coins" folder
file_paths = list(input_dir.glob("*.csv"))

# Remove OMNI files from the list
file_paths = [file for file in file_paths if "OMNI" not in 
file.stem]

# Check if files exist
if not file_paths:
    print("No suitable CSV files found in the 'all_coins' 
folder.")
else:
    # Initialize the plot
    plt.figure(figsize=(12, 8))
    processed_files = 0  # Counter for successfully processed 
files
    
    for file_path in file_paths:
        try:
            # Log the current file being processed
            print(f"Processing file: {file_path.stem}")
            
            # Load the CSV file
            df = pd.read_csv(file_path)
            
            # Ensure the 'Date' column is in datetime format
            if 'Date' in df.columns:
                df['Date'] = pd.to_datetime(df['Date'])
            elif 'Open time' in df.columns:  # Handle "Open time" 
column if present instead of "Date"
                df['Date'] = pd.to_datetime(df['Open time'])
            else:
                print(f"'Date' or 'Open time' column not found in 
{file_path}. Skipping this file.")
```

```
                    continue

                # Sort by date
                df = df.sort_values(by='Date')

                # Calculate days since the start
                df['days_since_start'] = (df['Date'] - 
df['Date'].iloc[0]).dt.days

                # Curve name (filename without extension)
                curve_name = file_path.stem

                # Plot close prices
                if 'Close' in df.columns:
                    plt.plot(df['days_since_start'], df['Close'], 
marker='o', linestyle='-', label=curve_name, alpha=0.8)
                    processed_files += 1  # Increment the counter
                else:
                    print(f"'Close' column not found in {file_path}. 
Skipping this file.")

        except Exception as e:
            print(f"Error processing file {file_path}: {e}")

    # Customize the plot
    plt.title("Trend Over Time (Days Since Start) - All Curves 
(Excl. OMNI)", fontsize=16)
    plt.xlabel("Days Since Listing", fontsize=12)
    plt.ylabel("Close Price", fontsize=12)
    plt.grid()
    plt.legend(title="Assets", loc="upper left", fontsize=10)

    # Save the combined plot
    combined_plot_file = output_dir / 
"combined_trend_plot_excl_OMNI.png"
    plt.savefig(combined_plot_file)
    plt.show()  # Optional: Display the plot directly
    print(f"Combined plot saved: {combined_plot_file}")
    print(f"Successfully plotted {processed_files} files.")
```

Binomial testing

```
from scipy.stats import binomtest

def binomial_test_greater_than_50(successes, trials):
    # Null hypothesis: p <= 0.5
    # Alternative hypothesis: p > 0.5
    p_value = binomtest(successes, trials, p=0.5, 
alternative='greater').pvalue
    return p_value

if __name__ == "__main__":
    # Example values: number of successes and trials
    successes = 30
    trials = 50

    # Calculate the p-value
    p_value = binomial_test_greater_than_50(successes, trials)
    print(f"The p-value of the test is: {p_value}")

    # Interpretation
    alpha = 0.05
    if p_value < alpha:
        print("The result is statistically significant. The 
probability of success is likely greater than 50%.")
    else:
        print("The result is not statistically significant. We 
cannot conclude that the probability of success is greater than 
50%.")
```