```
!pip install kiteconnect

from kiteconnect import KiteConnect

api_key = "x1x6m9ngri8vgbq7"  # New API Key
kite = KiteConnect(api_key=api_key)

print("🔗 Login here:")
print(kite.login_url())
```

```
Collecting kiteconnect
  Downloading kiteconnect-5.0.1-py3-none-any.whl.metadata (8.0 kB)
Collecting service-identity>=18.1.0 (from kiteconnect)
  Downloading service_identity-24.2.0-py3-none-any.whl.metadata (5.1 kB)
Requirement already satisfied: requests>=2.18.4 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (2.32.3)
Requirement already satisfied: six>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (1.17.0)
Requirement already satisfied: pyOpenSSL>=17.5.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (24.2.1)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (2.9.0.post0)
Collecting autobahn==19.11.2 (from autobahn[twisted]==19.11.2->kiteconnect)
  Downloading autobahn-19.11.2-py2.py3-none-any.whl.metadata (16 kB)
Collecting txaio>=18.8.1 (from autobahn==19.11.2->autobahn[twisted]==19.11.2->kiteconnect)
  Downloading txaio-25.6.1-py2.py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: cryptography>=2.7 in /usr/local/lib/python3.11/dist-packages (from autobahn==19.11.2->autobahn[twisted]==
Collecting zope.interface>=3.6.0 (from autobahn[twisted]==19.11.2->kiteconnect)
  Downloading zope.interface-7.2-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metad
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.4/44.4 kB 2.3 MB/s eta 0:00:00
Collecting Twisted>=12.1.0 (from autobahn[twisted]==19.11.2->kiteconnect)
  Downloading twisted-25.5.0-py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (2.4.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (2025.
Requirement already satisfied: attrs>=19.1.0 in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (25
Requirement already satisfied: pyasn1 in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (0.6.1)
Requirement already satisfied: pyasn1-modules in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (0
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography>=2.7->autobahn==19.11.2->autobah
Collecting automat>=24.8.0 (from Twisted>=12.1.0->autobahn[twisted]==19.11.2->kiteconnect)
  Downloading automat-25.4.16-py3-none-any.whl.metadata (8.4 kB)
Collecting constantly>=15.1 (from Twisted>=12.1.0->autobahn[twisted]==19.11.2->kiteconnect)
  Downloading constantly-23.10.4-py3-none-any.whl.metadata (1.8 kB)
Collecting hyperlink>=17.1.1 (from Twisted>=12.1.0->autobahn[twisted]==19.11.2->kiteconnect)
  Downloading hyperlink-21.0.0-py2.py3-none-any.whl.metadata (1.5 kB)
Collecting incremental>=24.7.0 (from Twisted>=12.1.0->autobahn[twisted]==19.11.2->kiteconnect)
  Downloading incremental-24.7.2-py3-none-any.whl.metadata (8.1 kB)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twist
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from zope.interface>=3.6.0->autobahn[twisted]==19.
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography>=2.7->autobahn==19.11
Downloading kiteconnect-5.0.1-py3-none-any.whl (24 kB)
Downloading autobahn-19.11.2-py2.py3-none-any.whl (771 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 771.5/771.5 kB 15.0 MB/s eta 0:00:00
Downloading service_identity-24.2.0-py3-none-any.whl (11 kB)
Downloading twisted-25.5.0-py3-none-any.whl (3.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 62.3 MB/s eta 0:00:00
Downloading txaio-25.6.1-py2.py3-none-any.whl (31 kB)
Downloading zope.interface-7.2-cp311-cp311-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (259 kB
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 259.8/259.8 kB 16.9 MB/s eta 0:00:00
Downloading automat-25.4.16-py3-none-any.whl (42 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.8/42.8 kB 3.1 MB/s eta 0:00:00
Downloading constantly-23.10.4-py3-none-any.whl (13 kB)
Downloading hyperlink-21.0.0-py2.py3-none-any.whl (74 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 74.6/74.6 kB 6.1 MB/s eta 0:00:00
Downloading incremental-24.7.2-py3-none-any.whl (20 kB)
Installing collected packages: zope.interface, txaio, incremental, hyperlink, constantly, automat, Twisted, service-identity, autobahn,
Successfully installed Twisted-25.5.0 autobahn-19.11.2 automat-25.4.16 constantly-23.10.4 hyperlink-21.0.0 incremental-24.7.2 kiteconnec
🔗 Login here:
https://kite.zerodha.com/connect/login?api_key=x1x6m9ngri8vgbq7&v=3
```

```
from kiteconnect import KiteConnect

api_key = "x1x6m9ngri8vgbq7"
api_secret = "697s2fxhn1mwimimhv964defnv157k4u"
request_token = "0HCarZhpP8cqMCFcwcx3Gut6Ug6YqMuP"

kite = KiteConnect(api_key=api_key)
data = kite.generate_session(request_token, api_secret=api_secret)

access_token = data["access_token"]
kite.set_access_token(access_token)
```

```
print("✅ Access token:", access_token)
```

```
✅ Access token: rK8IOWlAQcAcI2PG1g2QysYkfh3Eqz7m
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

# Step 1: Setup
kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("rK8IOWlAQcAcI2PG1g2QysYkfh3Eqz7m")  # 🔁 Replace this with your token

# Step 2: Reliance NSE instrument token
instrument_token = 738561  # Reliance Industries

# Step 3: Define time range
start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

# Step 4: Fetch data in 60-day chunks
final_data = []
chunk_start = start_date

def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error on chunk {chunk_start.date()} to {chunk_end.date()}: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)  # 🔁 Avoid rate limit

# Step 5: Save to CSV
df = pd.DataFrame(final_data)

if not df.empty:
    df['date'] = pd.to_datetime(df['date'])
    df.to_csv("reliance_1hr_intraday.csv", index=False)
    print("✅ Done! Data saved to reliance_1hr_intraday.csv")
else:
    print("⚠️ No data fetched. Check token, permissions, or internet.")
```

```
📦 Fetching from 2021-07-07 to 2021-09-04
📦 Fetching from 2021-09-05 to 2021-11-03
📦 Fetching from 2021-11-04 to 2022-01-02
📦 Fetching from 2022-01-03 to 2022-03-03
📦 Fetching from 2022-03-04 to 2022-05-02
📦 Fetching from 2022-05-03 to 2022-07-01
📦 Fetching from 2022-07-02 to 2022-08-30
📦 Fetching from 2022-08-31 to 2022-10-29
📦 Fetching from 2022-10-30 to 2022-12-28
📦 Fetching from 2022-12-29 to 2023-02-26
📦 Fetching from 2023-02-27 to 2023-04-27
📦 Fetching from 2023-04-28 to 2023-06-26
📦 Fetching from 2023-06-27 to 2023-08-25
📦 Fetching from 2023-08-26 to 2023-10-24
📦 Fetching from 2023-10-25 to 2023-12-23
📦 Fetching from 2023-12-24 to 2024-02-21
📦 Fetching from 2024-02-22 to 2024-04-21
📦 Fetching from 2024-04-22 to 2024-06-20
📦 Fetching from 2024-06-21 to 2024-08-19
📦 Fetching from 2024-08-20 to 2024-10-18
📦 Fetching from 2024-10-19 to 2024-12-17
📦 Fetching from 2024-12-18 to 2025-02-15
📦 Fetching from 2025-02-16 to 2025-04-16
📦 Fetching from 2025-04-17 to 2025-06-15
📦 Fetching from 2025-06-16 to 2025-07-07
✅ Done! Data saved to reliance_1hr_intraday.csv
```

```
from google.colab import files
files.download('reliance_1hr_intraday.csv')
```

Start coding or generate with AI.

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("rK8IOWlAQcAcI2PG1g2QysYkfh3Eqz7m")  # Paste your token

instrument_token = 341249  # HDFC Bank NSE

start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)

df = pd.DataFrame(final_data)

if not df.empty:
    df['date'] = pd.to_datetime(df['date'])
    df.to_csv("hdfc_1hr_intraday.csv", index=False)
    print("✅ Data saved to hdfc_1hr_intraday.csv")
else:
    print("⚠️ No data fetched.")
```

```
📦 Fetching from 2021-07-07 to 2021-09-04
📦 Fetching from 2021-09-05 to 2021-11-03
📦 Fetching from 2021-11-04 to 2022-01-02
📦 Fetching from 2022-01-03 to 2022-03-03
📦 Fetching from 2022-03-04 to 2022-05-02
📦 Fetching from 2022-05-03 to 2022-07-01
📦 Fetching from 2022-07-02 to 2022-08-30
📦 Fetching from 2022-08-31 to 2022-10-29
📦 Fetching from 2022-10-30 to 2022-12-28
📦 Fetching from 2022-12-29 to 2023-02-26
📦 Fetching from 2023-02-27 to 2023-04-27
📦 Fetching from 2023-04-28 to 2023-06-26
📦 Fetching from 2023-06-27 to 2023-08-25
📦 Fetching from 2023-08-26 to 2023-10-24
📦 Fetching from 2023-10-25 to 2023-12-23
📦 Fetching from 2023-12-24 to 2024-02-21
📦 Fetching from 2024-02-22 to 2024-04-21
📦 Fetching from 2024-04-22 to 2024-06-20
📦 Fetching from 2024-06-21 to 2024-08-19
📦 Fetching from 2024-08-20 to 2024-10-18
📦 Fetching from 2024-10-19 to 2024-12-17
📦 Fetching from 2024-12-18 to 2025-02-15
📦 Fetching from 2025-02-16 to 2025-04-16
📦 Fetching from 2025-04-17 to 2025-06-15
📦 Fetching from 2025-06-16 to 2025-07-07
✅ Data saved to hdfc_1hr_intraday.csv
```

```
from google.colab import files
files.download('hdfc_1hr_intraday.csv')
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("rK8IOWlAQcAcI2PG1g2QysYkfh3Eqz7m")  # Use your valid token

instrument_token = 1270529  # ICICI Bank NSE

start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)

df = pd.DataFrame(final_data)

if not df.empty:
    df['date'] = pd.to_datetime(df['date'])
    df.to_csv("icici_1hr_intraday.csv", index=False)
    print("✅ Data saved to icici_1hr_intraday.csv")
else:
    print("⚠️ No data fetched.")
```

```
📦 Fetching from 2021-07-07 to 2021-09-04
📦 Fetching from 2021-09-05 to 2021-11-03
📦 Fetching from 2021-11-04 to 2022-01-02
📦 Fetching from 2022-01-03 to 2022-03-03
📦 Fetching from 2022-03-04 to 2022-05-02
📦 Fetching from 2022-05-03 to 2022-07-01
📦 Fetching from 2022-07-02 to 2022-08-30
📦 Fetching from 2022-08-31 to 2022-10-29
📦 Fetching from 2022-10-30 to 2022-12-28
📦 Fetching from 2022-12-29 to 2023-02-26
📦 Fetching from 2023-02-27 to 2023-04-27
📦 Fetching from 2023-04-28 to 2023-06-26
📦 Fetching from 2023-06-27 to 2023-08-25
📦 Fetching from 2023-08-26 to 2023-10-24
📦 Fetching from 2023-10-25 to 2023-12-23
📦 Fetching from 2023-12-24 to 2024-02-21
📦 Fetching from 2024-02-22 to 2024-04-21
📦 Fetching from 2024-04-22 to 2024-06-20
📦 Fetching from 2024-06-21 to 2024-08-19
📦 Fetching from 2024-08-20 to 2024-10-18
📦 Fetching from 2024-10-19 to 2024-12-17
📦 Fetching from 2024-12-18 to 2025-02-15
📦 Fetching from 2025-02-16 to 2025-04-16
📦 Fetching from 2025-04-17 to 2025-06-15
📦 Fetching from 2025-06-16 to 2025-07-07
✅ Data saved to icici_1hr_intraday.csv
```

```
from google.colab import files
files.download("icici_1hr_intraday.csv")
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("rK8IOWlAQcAcI2PG1g2QysYkfh3Eqz7m")  # 🔁 Replace with valid token

instrument_token = 408065  # Infosys NSE

start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)

df = pd.DataFrame(final_data)

if not df.empty:
    df['date'] = pd.to_datetime(df['date'])
    df.to_csv("infosys_1hr_intraday.csv", index=False)
    print("✅ Data saved to infosys_1hr_intraday.csv")
else:
    print("⚠️ No data fetched.")
```

```
📦 Fetching from 2021-07-07 to 2021-09-04
📦 Fetching from 2021-09-05 to 2021-11-03
📦 Fetching from 2021-11-04 to 2022-01-02
📦 Fetching from 2022-01-03 to 2022-03-03
📦 Fetching from 2022-03-04 to 2022-05-02
📦 Fetching from 2022-05-03 to 2022-07-01
📦 Fetching from 2022-07-02 to 2022-08-30
📦 Fetching from 2022-08-31 to 2022-10-29
📦 Fetching from 2022-10-30 to 2022-12-28
📦 Fetching from 2022-12-29 to 2023-02-26
📦 Fetching from 2023-02-27 to 2023-04-27
📦 Fetching from 2023-04-28 to 2023-06-26
📦 Fetching from 2023-06-27 to 2023-08-25
📦 Fetching from 2023-08-26 to 2023-10-24
📦 Fetching from 2023-10-25 to 2023-12-23
📦 Fetching from 2023-12-24 to 2024-02-21
📦 Fetching from 2024-02-22 to 2024-04-21
📦 Fetching from 2024-04-22 to 2024-06-20
📦 Fetching from 2024-06-21 to 2024-08-19
📦 Fetching from 2024-08-20 to 2024-10-18
📦 Fetching from 2024-10-19 to 2024-12-17
📦 Fetching from 2024-12-18 to 2025-02-15
📦 Fetching from 2025-02-16 to 2025-04-16
📦 Fetching from 2025-04-17 to 2025-06-15
📦 Fetching from 2025-06-16 to 2025-07-07
✅ Data saved to infosys_1hr_intraday.csv
```

```
from google.colab import files
files.download("infosys_1hr_intraday.csv")
```

```
from kiteconnect import KiteConnect

api_key = "x1x6m9ngri8vgbq7"
api_secret = "697s2fxhn1mwimimhv964defnv157k4u"

kite = KiteConnect(api_key=api_key)
print("Login URL:", kite.login_url())  # Step 1: Get new login URL
```

```
Login URL: https://kite.zerodha.com/connect/login?api_key=x1x6m9ngri8vgbq7&v=3
```

```
from kiteconnect import KiteConnect

api_key = "x1x6m9ngri8vgbq7"

kite = KiteConnect(api_key=api_key)

# 🔗 This will print the URL you need to log in
print("🔐 Login here to generate request_token:")
print(kite.login_url())
```

```
🔐 Login here to generate request_token:
https://kite.zerodha.com/connect/login?api_key=x1x6m9ngri8vgbq7&v=3
```

```
Start coding or generate with AI.
```

```
!pip install kiteconnect

from kiteconnect import KiteConnect

# ✅ Your Kite app credentials
api_key = "x1x6m9ngri8vgbq7"
api_secret = "697s2fxhn1mwimimhv964defnv157k4u"
request_token = "7y8UOIrx6uv1uTxQpNyhBmC7H26YaO1Z"  # 🟢 Replace this

# 🚀 Generate access token
kite = KiteConnect(api_key=api_key)
data = kite.generate_session(request_token, api_secret=api_secret)

access_token = data['access_token']
kite.set_access_token(access_token)

print("✅ Access token:", access_token)
```

```
Requirement already satisfied: kiteconnect in /usr/local/lib/python3.11/dist-packages (5.0.1)
Requirement already satisfied: service-identity>=18.1.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (24.2.0)
Requirement already satisfied: requests>=2.18.4 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (2.32.3)
Requirement already satisfied: six>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (1.17.0)
Requirement already satisfied: pyOpenSSL>=17.5.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (24.2.1)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (2.9.0.post0)
Requirement already satisfied: autobahn==19.11.2 in /usr/local/lib/python3.11/dist-packages (from autobahn[twisted]==19.11.2->kiteconnec
Requirement already satisfied: txaio>=18.8.1 in /usr/local/lib/python3.11/dist-packages (from autobahn==19.11.2->autobahn[twisted]==19.1
Requirement already satisfied: cryptography>=2.7 in /usr/local/lib/python3.11/dist-packages (from autobahn==19.11.2->autobahn[twisted]==
Requirement already satisfied: zope.interface>=3.6.0 in /usr/local/lib/python3.11/dist-packages (from autobahn[twisted]==19.11.2->kitecc
Requirement already satisfied: Twisted>=12.1.0 in /usr/local/lib/python3.11/dist-packages (from autobahn[twisted]==19.11.2->kiteconnect)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (2.4.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (2025.
Requirement already satisfied: attrs>=19.1.0 in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (25
Requirement already satisfied: pyasn1 in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (0.6.1)
Requirement already satisfied: pyasn1-modules in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (0
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography>=2.7->autobahn==19.11.2->autobah
Requirement already satisfied: automat>=24.8.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==19.1
Requirement already satisfied: constantly>=15.1 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==19.
Requirement already satisfied: hyperlink>=17.1.1 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==19
Requirement already satisfied: incremental>=24.7.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twist
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from zope.interface>=3.6.0->autobahn[twisted]==19.
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography>=2.7->autobahn==19.11
✅ Access token: ePg44yrJzv92nopffw2Xs3eYDuYlI4xO
```

```
from kiteconnect import KiteConnect

api_key = "x1x6m9ngri8vgbq7"
kite = KiteConnect(api_key=api_key)

print("🔗 Login URL:", kite.login_url())
```

```
🔗 Login URL: https://kite.zerodha.com/connect/login?api_key=x1x6m9ngri8vgbq7&v=3
```

```
!pip install kiteconnect
from kiteconnect import KiteConnect

api_key = "x1x6m9ngri8vgbq7"
api_secret = "697s2fxhn1mwimimhv964defnv157k4u"
request_token = "DJ1Dlh0kGtKlOxzM8A7kt4inJxlzcka9"  # From redirect URL

kite = KiteConnect(api_key=api_key)
data = kite.generate_session(request_token, api_secret=api_secret)

access_token = data['access_token']
kite.set_access_token(access_token)

print("✅ Access Token:", access_token)
```

```
Requirement already satisfied: kiteconnect in /usr/local/lib/python3.11/dist-packages (5.0.1)
Requirement already satisfied: service-identity>=18.1.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (24.2.0)
Requirement already satisfied: requests>=2.18.4 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (2.32.3)
Requirement already satisfied: six>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (1.17.0)
Requirement already satisfied: pyOpenSSL>=17.5.0 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (24.2.1)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.11/dist-packages (from kiteconnect) (2.9.0.post0)
Requirement already satisfied: autobahn==19.11.2 in /usr/local/lib/python3.11/dist-packages (from autobahn[twisted]==19.11.2->kiteconnec
Requirement already satisfied: txaio>=18.8.1 in /usr/local/lib/python3.11/dist-packages (from autobahn==19.11.2->autobahn[twisted]==19.1
Requirement already satisfied: cryptography>=2.7 in /usr/local/lib/python3.11/dist-packages (from autobahn==19.11.2->autobahn[twisted]==
Requirement already satisfied: zope.interface>=3.6.0 in /usr/local/lib/python3.11/dist-packages (from autobahn[twisted]==19.11.2->kiteco
Requirement already satisfied: Twisted>=12.1.0 in /usr/local/lib/python3.11/dist-packages (from autobahn[twisted]==19.11.2->kiteconnect)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (2.4.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.18.4->kiteconnect) (2025.
Requirement already satisfied: attrs>=19.1.0 in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (25
Requirement already satisfied: pyasn1 in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (0.6.1)
Requirement already satisfied: pyasn1-modules in /usr/local/lib/python3.11/dist-packages (from service-identity>=18.1.0->kiteconnect) (0
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography>=2.7->autobahn==19.11.2->autobah
Requirement already satisfied: automat>=24.8.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==19.1
Requirement already satisfied: constantly>=15.1 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==19.
Requirement already satisfied: hyperlink>=17.1.1 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==19
Requirement already satisfied: incremental>=24.7.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twisted]==
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.11/dist-packages (from Twisted>=12.1.0->autobahn[twist
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from zope.interface>=3.6.0->autobahn[twisted]==19.
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography>=2.7->autobahn==19.11
✅ Access Token: ePg44yrJzv92nopffw2Xs3eYDuYlI4xO
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("ePg44yrJzv92nopffw2Xs3eYDuYlI4xO")  # Replace with your valid access token

# Step 1: Get instrument token for TCS
```

```
instruments = kite.instruments("NSE")
symbol = "TCS"

try:
    instrument_token = [i['instrument_token'] for i in instruments if i['tradingsymbol'] == symbol and i['exchange'] == "NSE"][0]
    print(f"🎯 Found TCS token: {instrument_token}")
except IndexError:
    print("❌ TCS token not found.")
    exit()

# Step 2: Date range and fetch logic
start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)  # 🔁 Respect rate limits

# Step 3: Save to CSV
df = pd.DataFrame(final_data)

if not df.empty:
    df['date'] = pd.to_datetime(df['date'])
    df.to_csv("tcs_1hr_intraday.csv", index=False)
    print("✅ Data saved to tcs_1hr_intraday.csv")
else:
    print("⚠️ No data fetched.")
```

```
🎯 Found TCS token: 2953217
📦 Fetching from 2021-07-07 to 2021-09-04
📦 Fetching from 2021-09-05 to 2021-11-03
📦 Fetching from 2021-11-04 to 2022-01-02
📦 Fetching from 2022-01-03 to 2022-03-03
📦 Fetching from 2022-03-04 to 2022-05-02
📦 Fetching from 2022-05-03 to 2022-07-01
📦 Fetching from 2022-07-02 to 2022-08-30
📦 Fetching from 2022-08-31 to 2022-10-29
📦 Fetching from 2022-10-30 to 2022-12-28
📦 Fetching from 2022-12-29 to 2023-02-26
📦 Fetching from 2023-02-27 to 2023-04-27
📦 Fetching from 2023-04-28 to 2023-06-26
📦 Fetching from 2023-06-27 to 2023-08-25
📦 Fetching from 2023-08-26 to 2023-10-24
📦 Fetching from 2023-10-25 to 2023-12-23
📦 Fetching from 2023-12-24 to 2024-02-21
📦 Fetching from 2024-02-22 to 2024-04-21
📦 Fetching from 2024-04-22 to 2024-06-20
📦 Fetching from 2024-06-21 to 2024-08-19
📦 Fetching from 2024-08-20 to 2024-10-18
📦 Fetching from 2024-10-19 to 2024-12-17
📦 Fetching from 2024-12-18 to 2025-02-15
📦 Fetching from 2025-02-16 to 2025-04-16
📦 Fetching from 2025-04-17 to 2025-06-15
📦 Fetching from 2025-06-16 to 2025-07-07
✅ Data saved to tcs_1hr_intraday.csv
```

```
from google.colab import files
files.download("tcs_1hr_intraday.csv")
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("ePg44yrJzv92nopffw2Xs3eYDuYlI4xO")  # ← Make sure this is your valid access token

# Step 1: Get instrument token for ITC
instruments = kite.instruments("NSE")
symbol = "ITC"

try:
    instrument_token = [item['instrument_token'] for item in instruments
                        if item['tradingsymbol'] == symbol and item['exchange'] == "NSE"][0]
    print(f"🎯 Found ITC token: {instrument_token}")
except IndexError:
    print("❌ ITC token not found.")
    exit()

# Step 2: Set your date range
start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

# Function to fetch data in chunks
def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

# Step 3: Fetch data in 60-day chunks
while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error fetching chunk {chunk_start.date()} to {chunk_end.date()}: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)

# Step 4: Save results
df = pd.DataFrame(final_data)
if not df.empty:
    df['date'] = pd.to_datetime(df['date'])
    df.to_csv("itc_1hr_intraday.csv", index=False)
    print("✅ Data saved to itc_1hr_intraday.csv")
else:
    print("⚠️ No data fetched. Check token or date range.")
```

```
🎯 Found ITC token: 424961
📦 Fetching from 2021-07-07 to 2021-09-04
📦 Fetching from 2021-09-05 to 2021-11-03
📦 Fetching from 2021-11-04 to 2022-01-02
📦 Fetching from 2022-01-03 to 2022-03-03
📦 Fetching from 2022-03-04 to 2022-05-02
📦 Fetching from 2022-05-03 to 2022-07-01
📦 Fetching from 2022-07-02 to 2022-08-30
📦 Fetching from 2022-08-31 to 2022-10-29
📦 Fetching from 2022-10-30 to 2022-12-28
📦 Fetching from 2022-12-29 to 2023-02-26
📦 Fetching from 2023-02-27 to 2023-04-27
📦 Fetching from 2023-04-28 to 2023-06-26
📦 Fetching from 2023-06-27 to 2023-08-25
📦 Fetching from 2023-08-26 to 2023-10-24
📦 Fetching from 2023-10-25 to 2023-12-23
📦 Fetching from 2023-12-24 to 2024-02-21
📦 Fetching from 2024-02-22 to 2024-04-21
📦 Fetching from 2024-04-22 to 2024-06-20
📦 Fetching from 2024-06-21 to 2024-08-19
📦 Fetching from 2024-08-20 to 2024-10-18
📦 Fetching from 2024-10-19 to 2024-12-17
📦 Fetching from 2024-12-18 to 2025-02-15
📦 Fetching from 2025-02-16 to 2025-04-16
📦 Fetching from 2025-04-17 to 2025-06-15
📦 Fetching from 2025-06-16 to 2025-07-07
✅ Data saved to itc_1hr_intraday.csv
```

```
from google.colab import files
files.download("itc_1hr_intraday.csv")
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("ePg44yrJzv92nopffw2Xs3eYDuYlI4xO")  # ← Use your valid access token

# Step 1: Get instrument token for L&T
instruments = kite.instruments("NSE")
symbol = "LT"  # L&T's Kite trading symbol

try:
    instrument_token = [
        item["instrument_token"]
        for item in instruments
        if item["tradingsymbol"] == symbol and item["exchange"] == "NSE"
    ][0]
    print(f"✅ Found L&T token: {instrument_token}")
except IndexError:
    print("❌ L&T token not found.")
    exit()

# Step 2: Define your date range
start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

# Step 3: Chunk fetch function
def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

# Step 4: Fetch data chunk-by-chunk
while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching L&T data from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error fetching chunk {chunk_start.date()}–{chunk_end.date()}: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)

# Step 5: Save to CSV
df = pd.DataFrame(final_data)
if not df.empty:
    df["date"] = pd.to_datetime(df["date"])
    df.to_csv("lnt_1hr_intraday.csv", index=False)
    print("✅ L&T intraday data saved to lnt_1hr_intraday.csv")
else:
    print("⚠️ No data fetched—double-check your token or date range.")
```

```
✅ Found L&T token: 2939649
📦 Fetching L&T data from 2021-07-07 to 2021-09-04
📦 Fetching L&T data from 2021-09-05 to 2021-11-03
📦 Fetching L&T data from 2021-11-04 to 2022-01-02
📦 Fetching L&T data from 2022-01-03 to 2022-03-03
📦 Fetching L&T data from 2022-03-04 to 2022-05-02
📦 Fetching L&T data from 2022-05-03 to 2022-07-01
📦 Fetching L&T data from 2022-07-02 to 2022-08-30
📦 Fetching L&T data from 2022-08-31 to 2022-10-29
📦 Fetching L&T data from 2022-10-30 to 2022-12-28
📦 Fetching L&T data from 2022-12-29 to 2023-02-26
📦 Fetching L&T data from 2023-02-27 to 2023-04-27
📦 Fetching L&T data from 2023-04-28 to 2023-06-26
📦 Fetching L&T data from 2023-06-27 to 2023-08-25
📦 Fetching L&T data from 2023-08-26 to 2023-10-24
📦 Fetching L&T data from 2023-10-25 to 2023-12-23
📦 Fetching L&T data from 2023-12-24 to 2024-02-21
📦 Fetching L&T data from 2024-02-22 to 2024-04-21
📦 Fetching L&T data from 2024-04-22 to 2024-06-20
📦 Fetching L&T data from 2024-06-21 to 2024-08-19
📦 Fetching L&T data from 2024-08-20 to 2024-10-18
📦 Fetching L&T data from 2024-10-19 to 2024-12-17
📦 Fetching L&T data from 2024-12-18 to 2025-02-15
📦 Fetching L&T data from 2025-02-16 to 2025-04-16
📦 Fetching L&T data from 2025-04-17 to 2025-06-15
📦 Fetching L&T data from 2025-06-16 to 2025-07-07
✅ L&T intraday data saved to lnt_1hr_intraday.csv
```

```
from google.colab import files
files.download("lnt_1hr_intraday.csv")
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("ePg44yrJzv92nopffw2Xs3eYDuYlI4xO")  # ← Use your valid access token

# Step 1: Get instrument token for Kotak Mahindra
instruments = kite.instruments("NSE")
symbol = "KOTAKBANK"

try:
    instrument_token = [
        item["instrument_token"]
        for item in instruments
        if item["tradingsymbol"] == symbol and item["exchange"] == "NSE"
    ][0]
    print(f"✅ Found Kotak token: {instrument_token}")
except IndexError:
    print("❌ Kotak token not found.")
    exit()

# Step 2: Define your date range
start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

# Step 3: Chunk fetch function
def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

# Step 4: Fetch data in chunks
```

```
while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f"📦 Fetching Kotak data from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"❌ Error fetching chunk {chunk_start.date()}–{chunk_end.date()}: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)  # Respect Kite rate limits

# Step 5: Save to CSV
df = pd.DataFrame(final_data)
if not df.empty:
    df["date"] = pd.to_datetime(df["date"])
    df.to_csv("kotakbank_1hr_intraday.csv", index=False)
    print("✅ Kotak Mahindra intraday data saved to kotakbank_1hr_intraday.csv")
else:
    print("⚠️ No data fetched—double-check your token, symbol, or date range.")
```

```
✅ Found Kotak token: 492033
📦 Fetching Kotak data from 2021-07-07 to 2021-09-04
📦 Fetching Kotak data from 2021-09-05 to 2021-11-03
📦 Fetching Kotak data from 2021-11-04 to 2022-01-02
📦 Fetching Kotak data from 2022-01-03 to 2022-03-03
📦 Fetching Kotak data from 2022-03-04 to 2022-05-02
📦 Fetching Kotak data from 2022-05-03 to 2022-07-01
📦 Fetching Kotak data from 2022-07-02 to 2022-08-30
📦 Fetching Kotak data from 2022-08-31 to 2022-10-29
📦 Fetching Kotak data from 2022-10-30 to 2022-12-28
📦 Fetching Kotak data from 2022-12-29 to 2023-02-26
📦 Fetching Kotak data from 2023-02-27 to 2023-04-27
📦 Fetching Kotak data from 2023-04-28 to 2023-06-26
📦 Fetching Kotak data from 2023-06-27 to 2023-08-25
📦 Fetching Kotak data from 2023-08-26 to 2023-10-24
📦 Fetching Kotak data from 2023-10-25 to 2023-12-23
📦 Fetching Kotak data from 2023-12-24 to 2024-02-21
📦 Fetching Kotak data from 2024-02-22 to 2024-04-21
📦 Fetching Kotak data from 2024-04-22 to 2024-06-20
📦 Fetching Kotak data from 2024-06-21 to 2024-08-19
📦 Fetching Kotak data from 2024-08-20 to 2024-10-18
📦 Fetching Kotak data from 2024-10-19 to 2024-12-17
📦 Fetching Kotak data from 2024-12-18 to 2025-02-15
📦 Fetching Kotak data from 2025-02-16 to 2025-04-16
📦 Fetching Kotak data from 2025-04-17 to 2025-06-15
📦 Fetching Kotak data from 2025-06-16 to 2025-07-07
✅ Kotak Mahindra intraday data saved to kotakbank_1hr_intraday.csv
```

```
from google.colab import files
files.download("kotakbank_1hr_intraday.csv")
```

```
from kiteconnect import KiteConnect
from datetime import datetime, timedelta
import pandas as pd
import time

kite = KiteConnect(api_key="x1x6m9ngri8vgbq7")
kite.set_access_token("ePg44yrJzv92nopffw2Xs3eYDuYlI4xO")  # ← Use your valid access token

# Step 1: Get instrument token for Axis Bank
instruments = kite.instruments("NSE")
symbol = "AXISBANK"

try:
    instrument_token = [
        item["instrument_token"]
        for item in instruments
        if item["tradingsymbol"] == symbol and item["exchange"] == "NSE"
    ][0]
    print(f"✅ Found Axis Bank token: {instrument_token}")
except IndexError:
    print(" Axis Bank token not found.")
    exit()

# Step 2: Define your date range
start_date = datetime(2021, 7, 7)
start_date = datetime(2021, 7, 7)
end_date = datetime(2025, 7, 7)

final_data = []
chunk_start = start_date

# Step 3: Chunk fetch function
def fetch_chunk(start, end):
    return kite.historical_data(
        instrument_token=instrument_token,
        from_date=start,
        to_date=end,
        interval="60minute",
        continuous=False
    )

# Step 4: Fetch data in 60-day chunks
while chunk_start < end_date:
    chunk_end = min(chunk_start + timedelta(days=59), end_date)
    print(f" Fetching Axis Bank data from {chunk_start.date()} to {chunk_end.date()}")
    try:
        data = fetch_chunk(chunk_start, chunk_end)
        final_data.extend(data)
    except Exception as e:
        print(f"Error fetching chunk {chunk_start.date()}–{chunk_end.date()}: {e}")
    chunk_start = chunk_end + timedelta(days=1)
    time.sleep(1)  # Respect rate limits

# Step 5: Save to CSV
df = pd.DataFrame(final_data)
if not df.empty:
    df["date"] = pd.to_datetime(df["date"])
    df.to_csv("axisbank_1hr_intraday.csv", index=False)
    print(" Axis Bank intraday data saved to axisbank_1hr_intraday.csv")
else:
    print("⚠️ No data fetched—check your token, symbol, or date range.")
```

```
✅ Found Axis Bank token: 1510401
📦 Fetching Axis Bank data from 2021-07-07 to 2021-09-04
📦 Fetching Axis Bank data from 2021-09-05 to 2021-11-03
📦 Fetching Axis Bank data from 2021-11-04 to 2022-01-02
📦 Fetching Axis Bank data from 2022-01-03 to 2022-03-03
📦 Fetching Axis Bank data from 2022-03-04 to 2022-05-02
📦 Fetching Axis Bank data from 2022-05-03 to 2022-07-01
📦 Fetching Axis Bank data from 2022-07-02 to 2022-08-30
📦 Fetching Axis Bank data from 2022-08-31 to 2022-10-29
📦 Fetching Axis Bank data from 2022-10-30 to 2022-12-28
📦 Fetching Axis Bank data from 2022-12-29 to 2023-02-26
📦 Fetching Axis Bank data from 2023-02-27 to 2023-04-27
📦 Fetching Axis Bank data from 2023-04-28 to 2023-06-26
📦 Fetching Axis Bank data from 2023-06-27 to 2023-08-25
📦 Fetching Axis Bank data from 2023-08-26 to 2023-10-24
📦 Fetching Axis Bank data from 2023-10-25 to 2023-12-23
📦 Fetching Axis Bank data from 2023-12-24 to 2024-02-21
📦 Fetching Axis Bank data from 2024-02-22 to 2024-04-21
📦 Fetching Axis Bank data from 2024-04-22 to 2024-06-20
📦 Fetching Axis Bank data from 2024-06-21 to 2024-08-19
📦 Fetching Axis Bank data from 2024-08-20 to 2024-10-18
📦 Fetching Axis Bank data from 2024-10-19 to 2024-12-17
📦 Fetching Axis Bank data from 2024-12-18 to 2025-02-15
📦 Fetching Axis Bank data from 2025-02-16 to 2025-04-16
📦 Fetching Axis Bank data from 2025-04-17 to 2025-06-15
📦 Fetching Axis Bank data from 2025-06-16 to 2025-07-07
✅ Axis Bank intraday data saved to axisbank_1hr_intraday.csv
```

```
from google.colab import files
files.download("axisbank_1hr_intraday.csv")
```

```
import pandas as pd
from datetime import datetime
from google.colab import files

# Step 1: Upload file
uploaded = files.upload()

# Step 2: Automatically detect and read file
for fname in uploaded.keys():
```

```
for fname in uploaded.keys():
    if fname.endswith('.xlsx'):
        df = pd.read_excel(fname)
    elif fname.endswith('.csv'):
        df = pd.read_csv(fname)
    else:
        raise ValueError("Unsupported file format. Please upload .xlsx or .csv")

# Step 3: Parse and fix 'Date' column

def parse_custom_date(d):
    try:
        return datetime.strptime(d.strip() + " 2021", "%b. %d %Y").date()
    except:
        return pd.NaT

df['formatted_date'] = df['Date'].astype(str).apply(parse_custom_date)

# Step 4: Show result
print("✅ Date column formatted. Sample:")
display(df[['Date', 'formatted_date']].head())


df.to_csv("cleaned_headlines_with_dates.csv", index=False)
print("File saved as cleaned_headlines_with_dates.csv")
```

```
Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to
enable.
Saving headlinesICICI_with_sentiment.xlsx to headlinesICICI_with_sentiment.xlsx
✅ Date column formatted. Sample:
          Date  formatted_date
0   Jun. 26,2025           NaT
1   Apr. 28,2025           NaT
2   Apr. 24,2025           NaT
3   Apr. 24,2025           NaT
4       Apr. ,2025         NaT
📁 File saved as cleaned_headlines_with_dates.csv
```

```
from google.colab import files
files.download("cleaned_headlines_with_dates.csv")
```

```
import pandas as pd
from datetime import datetime, timedelta
from google.colab import files

# ✅ Step 1: Upload price data
print("📈 Upload hourly price CSV file:")
uploaded_price = files.upload()
for fname in uploaded_price:
    df_price = pd.read_csv(fname)

# ✅ Step 2: Upload headline data
print("📰 Upload headline sentiment file (CSV or XLSX):")
uploaded_headlines = files.upload()
for fname in uploaded_headlines:
    if fname.endswith('.csv'):
        df_headlines = pd.read_csv(fname)
    elif fname.endswith('.xlsx'):
        df_headlines = pd.read_excel(fname)

# ✅ Rest of the logic remains the same...


# Step 2: Auto-detect which is which
for fname in uploaded:
    if "price" in fname.lower():
        df_price = pd.read_csv(fname)
    elif "headline" in fname.lower():
        if fname.endswith('.csv'):
            df_headlines = pd.read_csv(fname)
        elif fname.endswith('.xlsx'):
            df_headlines = pd.read_excel(fname)

# Step 3: Format datetime in price data
df_price['datetime'] = pd.to_datetime(df_price['date'], errors='coerce')
df_price['date'] = df_price['datetime'].dt.date
df_price['hour'] = df_price['datetime'].dt.strftime('%H:%M')

# Step 4: Fix headline dates (add year if needed)
def parse_custom_date(d):
    try:
        return datetime.strptime(d.strip() + " 2021", "%b. %d %Y").date()
    except:
        try:
            return pd.to_datetime(d).date()
        except:
            return pd.NaT


# Detect and parse the date column
def parse_custom_date(d):
    try:
        return datetime.strptime(d.strip() + " 2021", "%b. %d %Y").date()
    except:
        try:
            return pd.to_datetime(d).date()
        except:
            return pd.NaT

# Auto-detect the column that contains date info
date_col = [col for col in df_headlines.columns if 'date' in col.lower()][0]
df_headlines['date'] = df_headlines[date_col].astype(str).apply(parse_custom_date)


df_headlines['sentiment_score'] = df_headlines['Sentiment'].map({
    'Positive': 1,
    'Negative': -1,
    'Neutral': 0
}).fillna(0)

# Step 5: Build event window dataset
event_rows = []

for i, row in df_headlines.iterrows():
    symbol = row.get('symbol', 'HDFC')  # default symbol
    headline_date = row['date']
    sentiment = row['sentiment_score']

    if pd.isna(headline_date):
        continue

    for offset, label in zip([0, 1], ['T', 'T+1']):
        target_date = headline_date + timedelta(days=offset)
        price_rows = df_price[df_price['date'] == target_date].copy()
        price_rows['headline_date'] = headline_date
        price_rows['event_day'] = label
        price_rows['sentiment'] = sentiment
        event_rows.append(price_rows)

# Step 6: Concatenate all and reset index
df_event = pd.concat(event_rows).reset_index(drop=True)

# Step 7: Calculate return per row
df_event['return'] = (df_event['close'] - df_event['open']) / df_event['open']

# Step 8: Save to file
df_event.to_csv("headline_event_window_data.csv", index=False)
print("✅ Saved as 'headline_event_window_data.csv'")

# Show sample
df_event[['headline_date', 'event_day', 'date', 'hour', 'open', 'close', 'return', 'sentiment']].head(10)
```

```
📈 Upload hourly price CSV file:
```

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

```
Saving reliance_1hr_intraday.csv to reliance_1hr_intraday (1).csv
📰 Upload headline sentiment file (CSV or XLSX):
```

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

```
Saving headlinesReliance_with_sentiment (1).xlsx to headlinesReliance_with_sentiment (1) (2).xlsx
✅ Saved as 'headline_event_window_data.csv'
```

	headline_date	event_day	date	hour	open	close	return	sentiment
0	2025-04-25	T	2025-04-25	09:15	1303.5	1298.4	-0.003913	1
1	2025-04-25	T	2025-04-25	10:15	1298.0	1296.8	-0.000924	1
2	2025-04-25	T	2025-04-25	11:15	1296.4	1293.2	-0.002468	1
3	2025-04-25	T	2025-04-25	12:15	1293.2	1300.0	0.005258	1
4	2025-04-25	T	2025-04-25	13:15	1299.9	1301.4	0.001154	1
5	2025-04-25	T	2025-04-25	14:15	1301.4	1297.4	-0.003074	1
6	2025-04-25	T	2025-04-25	15:15	1297.8	1300.9	0.002389	1
7	2025-04-25	T	2025-04-25	09:15	1303.5	1298.4	-0.003913	1
8	2025-04-25	T	2025-04-25	10:15	1298.0	1296.8	-0.000924	1

```python
from google.colab import files
files.download("headline_event_window_data.csv")
```

```python
import pandas as pd
from datetime import datetime, timedelta
from google.colab import files

# Step 1: Upload files
print("📈 Upload hourly price data CSV:")
price_file = files.upload()

print("📰 Upload headline sentiment file (CSV or Excel):")
headline_file = files.upload()

# Step 2: Read files
for fname in price_file:
    df_price = pd.read_csv(fname)

for fname in headline_file:
    if fname.endswith('.xlsx'):
        df_headlines = pd.read_excel(fname)
    else:
        df_headlines = pd.read_csv(fname)

# Step 3: Clean price data
df_price['datetime'] = pd.to_datetime(df_price['date'], errors='coerce')
df_price = df_price.dropna(subset=['datetime']).sort_values('datetime')
df_price['return'] = df_price['close'].pct_change()

# Step 4: Create full hourly index
start = df_price['datetime'].min().floor('D')
end = df_price['datetime'].max().ceil('D')
all_hours = pd.date_range(start=start, end=end, freq='H')
market_hours = all_hours[(all_hours.hour >= 9) & (all_hours.hour <= 15)]
df_full = pd.DataFrame({'datetime': market_hours})
df_full = df_full.merge(df_price[['datetime', 'return']], on='datetime', how='left')

# Step 5: Clean headlines and sentiment
def parse_date(d):
    try:
        return datetime.strptime(d.strip() + " 2021", "%b. %d %Y").date()
    except:
        try:
            return pd.to_datetime(d).date()
        except:
            return pd.NaT
```

```
# Auto-detect date and sentiment columns
date_col = [col for col in df_headlines.columns if 'date' in col.lower()][0]
sent_col = [col for col in df_headlines.columns if 'sentiment' in col.lower()][0]

df_headlines['parsed_date'] = df_headlines[date_col].astype(str).apply(parse_date)
df_headlines['sentiment_score'] = df_headlines[sent_col].map({
    'Positive': 1,
    'Negative': -1,
    'Neutral': 0
}).fillna(0)

# Step 6: Inject sentiment into 10:15am each headline day
df_full['sentiment'] = 0
for _, row in df_headlines.iterrows():
    target_time = pd.Timestamp(datetime.combine(row['parsed_date'], datetime.strptime("10:15", "%H:%M").time()))
    if target_time in df_full['datetime'].values:
        df_full.loc[df_full['datetime'] == target_time, 'sentiment'] = row['sentiment_score']

# Step 7: Drop missing returns and reset index
df_full = df_full.dropna(subset=['return']).reset_index(drop=True)

# Step 8: Save and preview
df_full.to_csv("hourly_with_sentiment_ready.csv", index=False)
print("✅ File saved as 'hourly_with_sentiment_ready.csv'")

df_full[['datetime', 'return', 'sentiment']].head(10)
```

```
📈 Upload hourly price data CSV:
Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to
enable.
Saving reliance_1hr_intraday.csv to reliance_1hr_intraday (2).csv
📰 Upload headline sentiment file (CSV or Excel):
Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to
enable.
Saving headlinesReliance_with_sentiment (1).xlsx to headlinesReliance_with_sentiment (1) (3).xlsx
✅ File saved as 'hourly_with_sentiment_ready.csv'
/tmp/ipython-input-11-3273355573.py:30: FutureWarning: 'H' is deprecated and will be removed in a future version, please use 'h' instead
  all_hours = pd.date_range(start=start, end=end, freq='H')
```

```
from google.colab import files
files.download("hourly_with_sentiment_ready.csv")
```

```
import pandas as pd
from datetime import datetime, timedelta
from google.colab import files

# Step 1: Upload price and headline files
print("📈 Upload hourly price data CSV:")
uploaded_price = files.upload()

print("📰 Upload headline sentiment file (CSV or Excel):")
uploaded_headlines = files.upload()

# Step 2: Read uploaded files
for fname in uploaded_price:
    df_price = pd.read_csv(fname)

for fname in uploaded_headlines:
    if fname.endswith('.xlsx'):
        df_headlines = pd.read_excel(fname)
    else:
        df_headlines = pd.read_csv(fname)

# Step 3: Process price data
df_price['datetime'] = pd.to_datetime(df_price['date'], errors='coerce')
df_price = df_price.dropna(subset=['datetime'])
df_price = df_price.sort_values('datetime')
df_price['datetime'] = df_price['datetime'].dt.floor('min')

# Filter only between 9:00 and 15:30 (market hours)
df_price = df_price[(df_price['datetime'].dt.hour >= 9) & (df_price['datetime'].dt.hour <= 15)]

# Step 4: Build complete hourly time index
start = df_price['datetime'].min().floor('D')
```

```
end = df_price['datetime'].max().ceil('D')
all_hours = pd.date_range(start=start, end=end, freq='60min')
market_hours = all_hours[(all_hours.hour >= 9) & (all_hours.hour <= 15)]

df_full = pd.DataFrame({'datetime': market_hours})
df_full = df_full.merge(df_price[['datetime', 'open', 'close']], on='datetime', how='left')
df_full = df_full.sort_values('datetime')

# Step 5: Compute returns
df_full['return'] = df_full['close'].pct_change()

# Step 6: Process sentiment
date_col = [col for col in df_headlines.columns if 'date' in col.lower()][0]
sentiment_col = [col for col in df_headlines.columns if 'sentiment' in col.lower()][0]

df_headlines['parsed_date'] = pd.to_datetime(df_headlines[date_col], errors='coerce').dt.date
df_headlines['sentiment_score'] = df_headlines[sentiment_col].map({
    'Positive': 1, 'Negative': -1, 'Neutral': 0
}).fillna(0)

# Step 7: Inject sentiment into 10:15 AM slot
df_full['sentiment'] = 0
for _, row in df_headlines.iterrows():
    t = pd.Timestamp(datetime.combine(row['parsed_date'], datetime.strptime("10:15", "%H:%M").time()))
    if t in df_full['datetime'].values:
        df_full.loc[df_full['datetime'] == t, 'sentiment'] = row['sentiment_score']

# Step 8: Drop rows without return (first row will always be NaN)
df_final = df_full.dropna(subset=['return']).reset_index(drop=True)

# Step 9: Save to file
df_final.to_csv("hourly_with_sentiment_ready.csv", index=False)
print("✅ File saved as 'hourly_with_sentiment_ready.csv'")

# Show preview
df_final[['datetime', 'return', 'sentiment']].head(10)
```

```
📈 Upload hourly price data CSV:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving reliance_1hr_intraday.csv to reliance_1hr_intraday (4).csv
📰 Upload headline sentiment file (CSV or Excel):
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving headlinesReliance_with_sentiment (1).xlsx to headlinesReliance_with_sentiment (1) (5).xlsx
✅ File saved as 'hourly_with_sentiment_ready.csv'
/tmp/ipython-input-14-188709662.py:42: FutureWarning: The default fill_method='pad' in Series.pct_change is deprecated and will be remov
  df_full['return'] = df_full['close'].pct_change()
/tmp/ipython-input-14-188709662.py:48: UserWarning: Could not infer format, so each element will be parsed individually, falling back tc
  df_headlines['parsed_date'] = pd.to_datetime(df_headlines[date_col], errors='coerce').dt.date
    datetime  return  sentiment
```

```
import pandas as pd

# Step 1: Load your event-level dataset
df = pd.read_csv("headline_event_window_data.csv")

# Step 2: Make sure columns are in the right format
df['headline_date'] = pd.to_datetime(df['headline_date'], errors='coerce')
df['event_day'] = df['event_day'].fillna("T")

# Step 3: Group by headline and event_day to calculate stats
summary = df.groupby(['headline_date', 'sentiment', 'event_day']).agg(
    max_return=('return', 'max'),
    min_return=('return', 'min'),
    avg_return=('return', 'mean'),
    std_return=('return', 'std')
).reset_index()

# Step 4: Pivot so T and T+1 are side by side
pivot = summary.pivot(index=['headline_date', 'sentiment'], columns='event_day')
pivot.columns = ['_'.join(col).strip() for col in pivot.columns.values]
pivot = pivot.reset_index()

# Step 5: Save to CSV
pivot.to_csv("headline_return_summary.csv", index=False)
print("✅ File saved as 'headline_return_summary.csv'")
print("✅ File saved as 'headline_return_summary.csv'")
```

```
✅ File saved as 'headline_return_summary.csv'
```

```
from google.colab import files
files.download("headline_return_summary.csv")
```

```
# Step 1: Upload the file
from google.colab import files
import pandas as pd
import statsmodels.api as sm

print("📁 Upload your 'headline_return_summary_FIXED.csv' file:")
uploaded = files.upload()

# Automatically load the uploaded file
for fname in uploaded.keys():
    df = pd.read_csv(fname)

# Step 2: Drop missing values (to avoid errors in regression)
df = df.dropna(subset=['sentiment', 'avg_return_T', 'avg_return_T+1'])

# 🧪 Step 3: Run regression - Sentiment vs Avg Return on T (same day)
X1 = sm.add_constant(df['sentiment'])
y1 = df['avg_return_T']

model1 = sm.OLS(y1, X1).fit()
print(" Regression: Sentiment → Avg Return on Same Day (T)")
print(model1.summary())

# 🧪 Step 4: Run regression - Sentiment vs Avg Return on T+1 (next day)
X2 = sm.add_constant(df['sentiment'])
y2 = df['avg_return_T+1']

model2 = sm.OLS(y2, X2).fit()
print("\n Regression: Sentiment → Avg Return on Next Day (T+1)")
print(model2.summary())
```

```
📁 Upload your 'headline_return_summary_FIXED.csv' file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving headline_return_summary_complete.csv to headline_return_summary_complete.csv
📊 Regression: Sentiment → Avg Return on Same Day (T)
                            OLS Regression Results
==============================================================================
Dep. Variable:           avg_return_T   R-squared:                       0.099
Model:                            OLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.9942
Date:                Sun, 20 Jul 2025   Prob (F-statistic):              0.345
Time:                        18:49:37   Log-Likelihood:                 57.891
No. Observations:                  11   AIC:                            -111.8
Df Residuals:                       9   BIC:                            -111.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0011      0.000     -2.517      0.033      -0.002      -0.000
sentiment      0.0005      0.001      0.997      0.345      -0.001       0.002
==============================================================================
Omnibus:                        0.781   Durbin-Watson:                   3.389
Prob(Omnibus):                  0.677   Jarque-Bera (JB):                0.696
Skew:                          -0.458   Prob(JB):                        0.706
Kurtosis:                       2.175   Cond. No.                         1.29
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

📊 Regression: Sentiment → Avg Return on Next Day (T+1)
                            OLS Regression Results
==============================================================================
Dep. Variable:         avg_return_T+1   R-squared:                       0.072
Model:                            OLS   Adj. R-squared:                 -0.031
Method:                 Least Squares   F-statistic:                    0.7010
Date:                Sun, 20 Jul 2025   Prob (F-statistic):              0.424
Time:                        18:49:37   Log-Likelihood:                 56.606
No. Observations:                  11   AIC:                            -109.2
Df Residuals:                       9   BIC:                            -108.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0007      0.000     -1.466      0.177      -0.002       0.000
sentiment     -0.0005      0.001     -0.837      0.424      -0.002       0.001
==============================================================================
Omnibus:                        0.699   Durbin-Watson:                   1.333
Prob(Omnibus):                  0.705   Jarque-Bera (JB):                0.580
Skew:                           0.117   Prob(JB):                        0.748
Kurtosis:                       1.900   Cond. No.                         1.29
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:430: UserWarning: `kurtosistest` p-value may be inaccurate wit
  return hypotest_fun_in(*args, **kwds)
/usr/local/lib/python3.11/dist-packages/scipy/stats/_axis_nan_policy.py:430: UserWarning: `kurtosistest` p-value may be inaccurate wit
  return hypotest_fun_in(*args, **kwds)
```

```
print("Sentiment columns:", sentiment_df.columns.tolist())
print("Price_long columns:", price_long.columns.tolist())
```

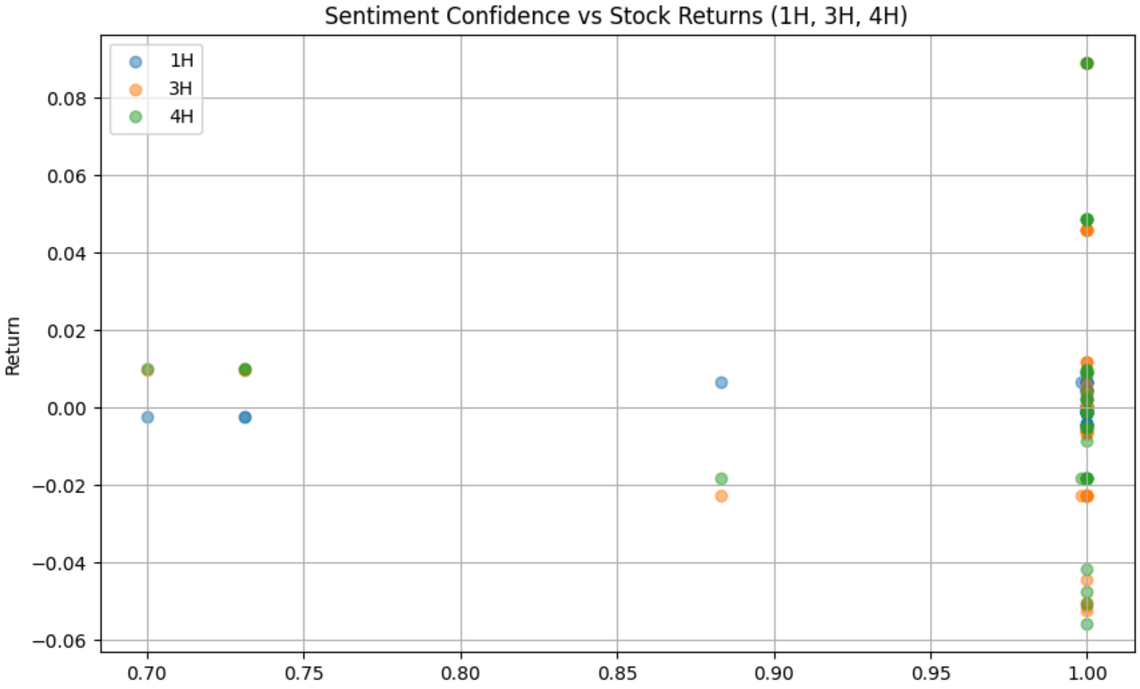
```
Sentiment columns: ['timestamp', 'headline', 'sentiment', 'confidence']
Price_long columns: ['timestamp', 'price', 'company', 'return_1h', 'return_3h', 'return_4h']
```

```
from google.colab import files
import pandas as pd
import io
import re
import matplotlib.pyplot as plt
from datetime import datetime

# Upload files
print(" Upload Sentiment File:")
sentiment_upload = files.upload()

print(" Upload Price Excel (multiple sheets):")
```

```
price_upload = files.upload()

# Load Sentiment
sentiment_file = list(sentiment_upload.keys())[0]
sentiment_df = pd.read_excel(io.BytesIO(sentiment_upload[sentiment_file]))

# Clean and combine date/time
sentiment_df.columns = [col.strip().lower() for col in sentiment_df.columns]
date_col = next((col for col in sentiment_df.columns if 'date' in col), None)
time_col = next((col for col in sentiment_df.columns if 'time' in col), None)

def clean_datetime(row):
    raw = f"{row[date_col]} {row[time_col]}"
    try:
        return datetime.strptime(raw.replace("A.M", "AM").replace("P.M", "PM").replace(".", "").strip(), "%b %d,%Y %I:%M %p")
    except:
        return pd.NaT

sentiment_df['timestamp'] = sentiment_df.apply(clean_datetime, axis=1)
sentiment_df.dropna(subset=['timestamp'], inplace=True)

# Identify relevant columns
headline_col = next((col for col in sentiment_df.columns if 'highlight' in col), None)
sentiment_col = next((col for col in sentiment_df.columns if 'sentiment' in col), None)
confidence_col = next((col for col in sentiment_df.columns if 'conf' in col), None)

sentiment_df = sentiment_df.rename(columns={
    headline_col: 'headline',
    sentiment_col: 'sentiment',
    confidence_col: 'confidence'
})

sentiment_df = sentiment_df[['timestamp', 'headline', 'sentiment', 'confidence']]

# Use known companies list to extract company from headlines
company_aliases = {
    'tcs': 'TCS',
    'reliance': 'Reliance',
    'infosys': 'Infosys',
    'l&t': 'L&T',
    'hdfc bank': 'HDFC Bank',
    'icici bank': 'ICICI Bank',
    'kotak mahindra': 'Kotak Mahindra Bank',
    'itc': 'ITC',
    'axis bank': 'Axis Bank'
}

def extract_company_from_headline(text):
    if not isinstance(text, str):
        return None
    text = text.lower()
    for key in company_aliases.keys():
        if key in text:
            return company_aliases[key]
    return None

sentiment_df['company'] = sentiment_df['headline'].apply(extract_company_from_headline)
sentiment_df.dropna(subset=['company'], inplace=True)

# Load price workbook
price_file = list(price_upload.keys())[0]
xls = pd.ExcelFile(io.BytesIO(price_upload[price_file]))

# Build price_long from sheets
price_long = pd.DataFrame()

for company in sentiment_df['company'].unique():
    if company in xls.sheet_names:
        df = xls.parse(company)
        df.columns = [c.strip().lower() for c in df.columns]
        time_col = next((c for c in df.columns if 'time' in c), df.columns[0])
        price_col = next((c for c in df.columns if 'close' in c or 'price' in c), df.columns[1])

        temp = df[[time_col, price_col]].copy()
        temp.columns = ['timestamp', 'price']
        temp['timestamp'] = pd.to_datetime(temp['timestamp'], errors='coerce')
        temp['price'] = pd.to_numeric(temp['price'], errors='coerce')
        temp.dropna(inplace=True)
        temp['company'] = company
        temp.sort_values('timestamp', inplace=True)

        # Add returns
        temp['return_1h'] = temp['price'].pct_change(periods=1)
        temp['return_3h'] = temp['price'].pct_change(periods=3)
        temp['return_4h'] = temp['price'].pct_change(periods=4)

        price_long = pd.concat([price_long, temp], ignore_index=True)

# Prepare timestamps
sentiment_df['timestamp'] = sentiment_df['timestamp'].dt.tz_localize(None)
# Final fix: Build price_long from sheets robustly
price_long_list = []

for company in sentiment_df['company'].unique():
    if company.lower() in [s.lower() for s in xls.sheet_names]:
        # Get the sheet name exactly as stored
        sheet_name = next(s for s in xls.sheet_names if s.lower() == company.lower())
        df = xls.parse(sheet_name)

        # If first row looks like column names (like in axis bank), use it as header
        if 'Unnamed' in str(df.columns[1]):
            df.columns = df.iloc[0]  # first row becomes header
            df = df[1:]
            df.columns = [str(c).strip().lower() for c in df.columns]
        else:
            df.columns = [c.strip().lower() for c in df.columns]

        # Ensure we have required columns
        if 'date' not in df.columns or 'close' not in df.columns:
            print(f" Skipping sheet '{sheet_name}' — missing 'date' or 'close' columns.")
            continue

        try:
            temp = df[['date', 'close']].copy()
            temp.rename(columns={'date': 'timestamp', 'close': 'price'}, inplace=True)
            temp['timestamp'] = pd.to_datetime(temp['timestamp'], errors='coerce')
            temp['price'] = pd.to_numeric(temp['price'], errors='coerce')
            temp.dropna(inplace=True)

            temp['company'] = sheet_name.strip().title()
            temp.sort_values('timestamp', inplace=True)

            temp['return_1h'] = temp['price'].pct_change(periods=1)
            temp['return_3h'] = temp['price'].pct_change(periods=3)
            temp['return_4h'] = temp['price'].pct_change(periods=4)

            price_long_list.append(temp)
        except Exception as e:
            print(f" Error processing sheet '{sheet_name}':", str(e))

# Combine valid sheets
price_long = pd.concat(price_long_list, ignore_index=True)

# Merge with asof logic
# Continue with timestamp cleaning, merge, and analysis...
sentiment_df['timestamp'] = pd.to_datetime(sentiment_df['timestamp']).dt.tz_localize(None)
price_long['timestamp'] = pd.to_datetime(price_long['timestamp']).dt.tz_localize(None)

merged = pd.merge_asof(
    sentiment_df.sort_values('timestamp'),
    price_long.sort_values('timestamp'),
    on='timestamp',
    by='company',
    direction='forward',
    tolerance=pd.Timedelta('1h')
)


# Drop invalid rows
merged.dropna(subset=['confidence', 'return_1h', 'return_3h', 'return_4h'], inplace=True)

# Print correlations
print("Correlations between Sentiment Confidence and Returns:")
for h in [1, 3, 4]:
    corr = merged['confidence'].corr(merged[f'return_{h}h'])
```

```
        corr = merged['confidence'].corr(merged[f'return_{h}h'])
        print(f" - Confidence vs Return {h}h: {corr:.4f}")

    # Plot all
    plt.figure(figsize=(10, 6))
    plt.scatter(merged['confidence'], merged['return_1h'], label='1H', alpha=0.5)
    plt.scatter(merged['confidence'], merged['return_3h'], label='3H', alpha=0.5)
    plt.scatter(merged['confidence'], merged['return_4h'], label='4H', alpha=0.5)
    plt.xlabel("Sentiment Confidence")
    plt.ylabel("Return")
    plt.title("Sentiment Confidence vs Stock Returns (1H, 3H, 4H)")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Save merged dataset to Excel
    output_path = "/content/merged_sentiment_price_data.xlsx"
    merged.to_excel(output_path, index=False)

    # Download link for Colab
    from google.colab import files
    files.download(output_path)
```

```
📁 Upload Sentiment File:
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving club.xlsx to club (12).xlsx
📁 Upload Price Excel (multiple sheets):
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving club prices.xlsx to club prices (12).xlsx
📊 Correlations between Sentiment Confidence and Returns:
 - Confidence vs Return 1h: 0.1600
 - Confidence vs Return 3h: -0.0177
 - Confidence vs Return 4h: -0.0170
```



```
    # STEP 1: Upload your file
    from google.colab import files
    import pandas as pd
    import numpy as np

    uploaded = files.upload()
    filename = next(iter(uploaded))

    # STEP 2: Load Excel file
    xls = pd.ExcelFile(filename)
    sheet_names = xls.sheet_names

    # STEP 3: Map internal sheet names to standard names used in headlines CSV
```

```
normalized_names = {
    "reliance": "Reliance",
    "hdfc ": "HDFC Bank",
    "icici": "ICICI Bank",
    "infosys": "Infosys",
    "itc": "ITC",
    "kotak mahindra bank": "Kotak Mahindra Bank",
    "lnt": "L&T",
    "tcs": "TCS",
    "axis bank": "Axis Bank"
}

# STEP 4: Clean each sheet without removing any rows
def clean_sheet(df, sheet_name):
    # Fix header for Axis Bank
    if sheet_name == "axis bank":
        df.columns = df.iloc[0]
        df = df.drop(0).reset_index(drop=True)

    # Standardize column names
    df.columns = [str(col).strip().lower() for col in df.columns]

    # Detect and rename date column
    date_col = "date" if "date" in df.columns else df.columns[0]
    df = df.rename(columns={date_col: "timestamp"})

    # Convert timestamp and numeric columns
    df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
    df[['open', 'high', 'low', 'close', 'volume']] = df[['open', 'high', 'low', 'close', 'volume']].apply(pd.to_numeric, errors='coerce')

    # Assign company name for merging
    df['company'] = normalized_names[sheet_name]

    return df[['timestamp', 'company', 'open', 'high', 'low', 'close', 'volume']].dropna()

# STEP 5: Combine all sheets
all_price_data = pd.DataFrame()

for sheet in sheet_names:
    if sheet in normalized_names:
        try:
            raw_df = xls.parse(sheet)
            clean_df = clean_sheet(raw_df, sheet)
            all_price_data = pd.concat([all_price_data, clean_df], ignore_index=True)
        except Exception as e:
            print(f"❌ Error in {sheet}: {e}")

# STEP 6: Export cleaned dataset
all_price_data.to_csv("Cleaned_Price_Data.csv", index=False)
print("✅ Done! Cleaned data saved as 'Cleaned_Price_Data.csv'")
files.download("Cleaned_Price_Data.csv")
```

```
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to
enable.
Saving club prices.xlsx to club prices.xlsx
✅ Done! Cleaned data saved as 'Cleaned_Price_Data.csv'
```

```
# STEP 1: Install dependencies
!pip install transformers
!pip install -q torch torchvision torchaudio

# STEP 2: Import libraries
import pandas as pd
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import pipeline
from google.colab import files

# STEP 3: Upload the headline CSV
uploaded = files.upload()
headline_file = next(iter(uploaded))

# STEP 4: Load the headlines data
df = pd.read_csv(headline_file)

# STEP 5: Load FinBERT model (financial sentiment)
model_name = "yiyanghkust/finbert-tone"
```

```
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name)

# Load pipeline
nlp = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

# STEP 6: Apply sentiment analysis to headlines
def get_sentiment(text):
    try:
        result = nlp(text[:512])[0]  # Truncate text if too long
        return result['label'].lower()
    except:
        return 'error'

df['sentiment'] = df['headline'].apply(get_sentiment)

# STEP 7: Save results
df.to_csv("Headlines_with_Sentiment.csv", index=False)
print("✅ Sentiment analysis complete. File ready.")
files.download("Headlines_with_Sentiment.csv")
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.53.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.33.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.2)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0-
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->trans
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.7.14)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 84.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 73.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 36.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 12.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 60.4 MB/s eta 0:00:00
```

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

```
Saving Company_Headlines_2021_2025_with_Axis.csv to Company_Headlines_2021_2025_with_Axis.csv
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

vocab.txt: 226k/? [00:00<00:00, 5.00MB/s]

config.json: 100% 533/533 [00:00<00:00, 23.3kB/s]

pytorch_model.bin: 100% 439M/439M [00:02<00:00, 182MB/s]

```
Device set to use cpu
```

model.safetensors: 100% 439M/439M [00:04<00:00, 158MB/s]

```
✅ Sentiment analysis complete. File ready.
```

```
# STEP 1: Install dependencies
!pip install transformers
!pip install -q torch torchvision torchaudio

# STEP 2: Import libraries
import pandas as pd
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import pipeline
from google.colab import files
```

```
# STEP 3: Upload the headline CSV
uploaded = files.upload()
headline_file = next(iter(uploaded))

# STEP 4: Load the headlines data
df = pd.read_csv(headline_file)

# STEP 5: Load FinBERT model (financial sentiment)
model_name = "yiyanghkust/finbert-tone"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name)

# Load pipeline
nlp = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer, return_all_scores=False)

# STEP 6: Apply sentiment + confidence
def get_sentiment_and_confidence(text):
    try:
        result = nlp(text[:512])[0]  # truncate long text
        return pd.Series([result['label'].lower(), round(result['score'], 4)])
    except:
        return pd.Series(['error', 0.0])

df[['sentiment', 'confidence']] = df['headline'].apply(get_sentiment_and_confidence)

# STEP 7: Save results
df.to_csv("Headlines_with_Sentiment_Confidence.csv", index=False)
print("✅ Sentiment + confidence added.")
files.download("Headlines_with_Sentiment_Confidence.csv")
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.53.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.33.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.2)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0-
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->trans
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.7.14)
```

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

```
Saving Company_Headlines_2021_2025_with_Axis.csv to Company_Headlines_2021_2025_with_Axis (1).csv
Device set to use cpu
/usr/local/lib/python3.11/dist-packages/transformers/pipelines/text_classification.py:106: UserWarning: `return_all_scores` is now depre
  warnings.warn(
✅ Sentiment + confidence added.
```

```
# STEP 1: Upload price data file
from google.colab import files
import pandas as pd

print("📁 Please upload your Cleaned_Price_Data.csv file:")
uploaded_price = files.upload()
price_filename = next(iter(uploaded_price))
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])

# STEP 2: Upload headline + sentiment file
print("\n📁 Now upload your Headlines_with_Sentiment_Confidence.csv file:")
uploaded_headlines = files.upload()
headline_filename = next(iter(uploaded_headlines))
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 3: Map sentiment to numeric score
sentiment_map = {'positive': 1, 'neutral': 0, 'negative': -1}
headline_df['sentiment_score'] = headline_df['sentiment'].map(sentiment_map)

# STEP 4: Merge on company and timestamp
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
```

```
# STEP 5: Calculate next-hour return
merged_df = merged_df.sort_values(['company', 'timestamp'])
merged_df['next_close'] = merged_df.groupby('company')['close'].shift(-1)
merged_df['return'] = (merged_df['next_close'] - merged_df['close']) / merged_df['close']
merged_df = merged_df.dropna(subset=['return'])

# STEP 6: Regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

X = merged_df[['sentiment_score', 'confidence']]
y = merged_df['return']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# STEP 7: Display Results
# STEP 7: Display Results
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("\n✅ Linear Regression Results:")
print(f"R² Score: {r2:.4f}")
print(f"RMSE: {rmse:.6f}")
print("Coefficients:")
for name, coef in zip(X.columns, model.coef_):
    print(f"  {name}: {coef:.6f}")

# STEP 8: Export final merged file
merged_df['predicted_return'] = model.predict(X)
merged_df.to_csv("Merged_Sentiment_Price_Data_with_Returns.csv", index=False)
files.download("Merged_Sentiment_Price_Data_with_Returns.csv")
```

```
📁 Please upload your Cleaned_Price_Data.csv file:
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (4).csv

📁 Now upload your Headlines_with_Sentiment_Confidence.csv file:
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Headlines_with_Sentiment_Confidence.csv to Headlines_with_Sentiment_Confidence (3).csv

✅ Linear Regression Results:
R² Score: -0.0026
RMSE: 0.039365
Coefficients:
  sentiment_score: 0.002381
```

```
# STEP 1: Install libraries
!pip install -q pandas scikit-learn

# STEP 2: Upload files
from google.colab import files
import pandas as pd

print("📁 Upload your CLEANED PRICE DATA file:")
price_file = files.upload()
price_filename = next(iter(price_file))

print("\n📁 Upload your PRICE-DRIVEN HEADLINES file:")
headline_file = files.upload()
headline_filename = next(iter(headline_file))

# STEP 3: Load both datasets
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 4: Merge on company and timestamp
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
```

```
# STEP 5: Calculate next-hour returns
merged_df = merged_df.sort_values(['company', 'timestamp'])
merged_df['next_close'] = merged_df.groupby('company')['close'].shift(-1)
merged_df['return'] = (merged_df['next_close'] - merged_df['close']) / merged_df['close']
merged_df = merged_df.dropna(subset=['return'])

# STEP 6: Prepare for regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

X = merged_df[['sentiment_score', 'confidence']]
y = merged_df['return']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# STEP 7: Evaluation
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
coefficients = dict(zip(X.columns, model.coef_))

print("\n✅ Linear Regression Summary:")
print(f"R² Score: {r2:.4f}")
print(f"RMSE: {rmse:.6f}")
print("\nCoefficients:")
for var, coef in coefficients.items():
    print(f"  {var}: {coef:.6f}")

# STEP 8: Export final merged file with predictions (optional)
merged_df['predicted_return'] = model.predict(X)
merged_df.to_csv("Merged_Sentiment_Price_Data_with_Returns.csv", index=False)
files.download("Merged_Sentiment_Price_Data_with_Returns.csv")
```

```
📁 Upload your CLEANED PRICE DATA file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (5).csv

📁 Upload your PRICE-DRIVEN HEADLINES file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025.csv

✅ Linear Regression Summary:
R² Score: -0.0106
RMSE: 0.052561

Coefficients:
  sentiment_score: 0.001771
```

```
# STEP 1: Install packages
!pip install -q scikit-learn pandas

# STEP 2: Upload your files
from google.colab import files
import pandas as pd

print("📁 Upload your CLEANED PRICE DATA file:")
price_file = files.upload()
price_filename = next(iter(price_file))

print("\n📁 Upload your PRICE-DRIVEN HEADLINES file:")
headline_file = files.upload()
headline_filename = next(iter(headline_file))

# STEP 3: Load datasets
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 4: Merge on timestamp + company
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])
```

```
# STEP 5: Create 4-hour forward return + classify direction
merged_df['next_close_4h'] = merged_df.groupby('company')['close'].shift(-4)
merged_df['return_4h'] = (merged_df['next_close_4h'] - merged_df['close']) / merged_df['close']
merged_df = merged_df.dropna(subset=['return_4h'])
merged_df['direction'] = (merged_df['return_4h'] > 0).astype(int)  # 1 = UP, 0 = DOWN/FLAT

# STEP 6: Train logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

X = merged_df[['sentiment_score', 'confidence']]
y = merged_df['direction']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

logit_model = LogisticRegression()
logit_model.fit(X_train, y_train)
y_pred = logit_model.predict(X_test)

# STEP 7: Evaluate performance
print("\nLogistic Regression Performance (4h Direction Prediction):")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# STEP 8: Export merged dataset with predictions (optional)
merged_df['predicted_direction'] = logit_model.predict(X)
merged_df.to_csv("Merged_Sentiment_Price_Direction.csv", index=False)
files.download("Merged_Sentiment_Price_Direction.csv")
```

```
📁 Upload your CLEANED PRICE DATA file:
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (6).csv

📁 Upload your PRICE-DRIVEN HEADLINES file:
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (1).csv

✅ Logistic Regression Performance (4h Direction Prediction):
Accuracy: 0.6325

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        43
           1       0.63      1.00      0.77        74

    accuracy                           0.63       117
   macro avg       0.32      0.50      0.39       117
weighted avg       0.40      0.63      0.49       117

Confusion Matrix:
[[ 0 43]
 [ 0 74]]
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
```

```
# STEP 1: Install dependencies
!pip install -q scikit-learn pandas

# STEP 2: Upload your files again
from google.colab import files
import pandas as pd

print(" Upload your CLEANED PRICE DATA file:")
price_file = files.upload()
price_filename = next(iter(price_file))

print("\n Upload your PRICE-DRIVEN HEADLINES file:")
headline_file = files.upload()
headline_file = files.upload()
headline_filename = next(iter(headline_file))

# STEP 3: Load data
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 4: Merge + create return direction
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])

merged_df['next_close_4h'] = merged_df.groupby('company')['close'].shift(-4)
merged_df['return_4h'] = (merged_df['next_close_4h'] - merged_df['close']) / merged_df['close']
merged_df = merged_df.dropna(subset=['return_4h'])
merged_df['direction'] = (merged_df['return_4h'] > 0).astype(int)

# STEP 5: Train Random Forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

X = merged_df[['sentiment_score', 'confidence']]
y = merged_df['direction']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# STEP 6: Evaluation
print("\n Random Forest Model Performance (4h Price Direction):")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# STEP 7: Export predictions (optional)
merged_df['predicted_direction'] = rf_model.predict(X)
merged_df.to_csv("RandomForest_Price_Direction.csv", index=False)
files.download("RandomForest_Price_Direction.csv")
```

```
📁 Upload your CLEANED PRICE DATA file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (7).csv

📁 Upload your PRICE-DRIVEN HEADLINES file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (2).csv

✅ Random Forest Model Performance (4h Price Direction):
Accuracy: 0.4957

Classification Report:
              precision    recall  f1-score   support

           0       0.32      0.33      0.32        43
           1       0.60      0.59      0.60        74

    accuracy                           0.50       117
   macro avg       0.46      0.46      0.46       117
weighted avg       0.50      0.50      0.50       117

Confusion Matrix:
[[14 29]
```

```
# STEP 1: Install dependencies
!pip install -q scikit-learn pandas

# STEP 2: Upload your files
from google.colab import files
import pandas as pd

print("Upload your CLEANED PRICE DATA file:")
price_file = files.upload()
price_filename = next(iter(price_file))
```

```
print("\n Upload your PRICE-DRIVEN HEADLINES file:")
headline_file = files.upload()
headline_filename = next(iter(headline_file))

# STEP 3: Load data
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 4: Merge + create next-day return direction
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])

#  Assuming 1 row = 1 hour, we shift by 7 for next-day return
merged_df['next_close_1d'] = merged_df.groupby('company')['close'].shift(-7)
merged_df['return_1d'] = (merged_df['next_close_1d'] - merged_df['close']) / merged_df['close']
merged_df = merged_df.dropna(subset=['return_1d'])
merged_df['direction'] = (merged_df['return_1d'] > 0).astype(int)  # 1 = up, 0 = down or same

# STEP 5: Train Random Forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

X = merged_df[['sentiment_score', 'confidence']]
y = merged_df['direction']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# STEP 6: Evaluation
print("\n Random Forest Model Performance (Next-Day Direction):")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# STEP 7: Export predictions (optional)
merged_df['predicted_direction'] = rf_model.predict(X)
merged_df.to_csv("RandomForest_NextDay_Direction.csv", index=False)
files.download("RandomForest_NextDay_Direction.csv")
```

```
📁 Upload your CLEANED PRICE DATA file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data.csv

📁 Upload your PRICE-DRIVEN HEADLINES file:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025.csv

✅ Random Forest Model Performance (Next-Day Direction):
Accuracy: 0.4911

Classification Report:
              precision    recall  f1-score   support

           0       0.39      0.22      0.28        51
           1       0.52      0.72      0.61        61

    accuracy                           0.49       112
   macro avg       0.46      0.47      0.44       112
weighted avg       0.46      0.49      0.46       112

Confusion Matrix:
[[11 40]
```

```
# STEP 1: Install packages
!pip install -q scikit-learn pandas

# STEP 2: Upload files
from google.colab import files
import pandas as pd
import numpy as np
import numpy as np

print(" Upload your CLEANED PRICE DATA file:")
price_file = files.upload()
price_filename = next(iter(price_file))

print("\n Upload your PRICE-DRIVEN HEADLINES file:")
headline_file = files.upload()
headline_filename = next(iter(headline_file))

# STEP 3: Load data
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 4: Merge
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])

# STEP 5: Technical indicators
merged_df['lagged_return'] = merged_df.groupby('company')['close'].pct_change()
merged_df['momentum_3h'] = merged_df.groupby('company')['close'].transform(lambda x: x - x.shift(3))
merged_df['volatility_3h'] = merged_df.groupby('company')['close'].transform(lambda x: x.pct_change().rolling(3).std())

# STEP 6: Next-day return direction (shift -7 for 1 day)
merged_df['next_close_1d'] = merged_df.groupby('company')['close'].shift(-7)
merged_df['return_1d'] = (merged_df['next_close_1d'] - merged_df['close']) / merged_df['close']
merged_df['direction'] = (merged_df['return_1d'] > 0).astype(int)

# Drop any missing values
merged_df = merged_df.dropna(subset=['lagged_return', 'momentum_3h', 'volatility_3h', 'return_1d'])

# STEP 7: Train Random Forest with class_weight='balanced'
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Features to include
features = ['sentiment_score', 'confidence', 'lagged_return', 'momentum_3h', 'volatility_3h']
X = merged_df[features]
y = merged_df['direction']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf_model = RandomForestClassifier(n_estimators=100, max_depth=5, class_weight='balanced', random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# STEP 8: Evaluate
print("\n✅ Random Forest with Technical Indicators + Class Weighting")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# STEP 9: Export (optional)
merged_df['predicted_direction'] = rf_model.predict(X)
merged_df.to_csv("Balanced_RF_NextDay_Direction_With_TechIndicators.csv", index=False)
files.download("Balanced_RF_NextDay_Direction_With_TechIndicators.csv")
```

```
📁 Upload your CLEANED PRICE DATA file:
```

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

```
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (1).csv

📁 Upload your PRICE-DRIVEN HEADLINES file:
```

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

```
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (1).csv

✅ Random Forest with Technical Indicators + Class Weighting
Accuracy: 0.5140
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.45      0.49        55
           1       0.50      0.58      0.54        52

    accuracy                           0.51       107
   macro avg       0.52      0.52      0.51       107
weighted avg       0.52      0.51      0.51       107

Confusion Matrix:
[[25 30]
```

```
# STEP 1: Install & import packages
!pip install -q pandas seaborn matplotlib

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# STEP 2: Upload your files
from google.colab import files
print("Upload your CLEANED PRICE DATA file:")
price_file = files.upload()
price_filename = next(iter(price_file))

print("Upload your HEADLINES WITH SENTIMENT file:")
headline_file = files.upload()
headline_filename = next(iter(headline_file))

# STEP 3: Load and merge data
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

merged = pd.merge(price_df, headline_df, on=['timestamp', 'company'], how='inner')
merged = merged.sort_values(['company', 'timestamp'])

# STEP 4: Calculate returns
merged['return_1h'] = merged.groupby('company')['close'].pct_change(1)
merged['return_4h'] = merged.groupby('company')['close'].pct_change(4)
merged['return_1d'] = merged.groupby('company')['close'].pct_change(7)  # Assuming ~7 hours in a trading day

# STEP 5: Drop missing values for correlation analysis
corr_df = merged[['sentiment_score', 'confidence', 'return_1h', 'return_4h', 'return_1d']].dropna()

# STEP 6: Compute correlation matrix
corr_matrix = corr_df.corr(method='pearson')

# STEP 7: Visualize correlation
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title(" Correlation Between Sentiment & Returns at 1h, 4h, 1d Horizons")
plt.show()
```

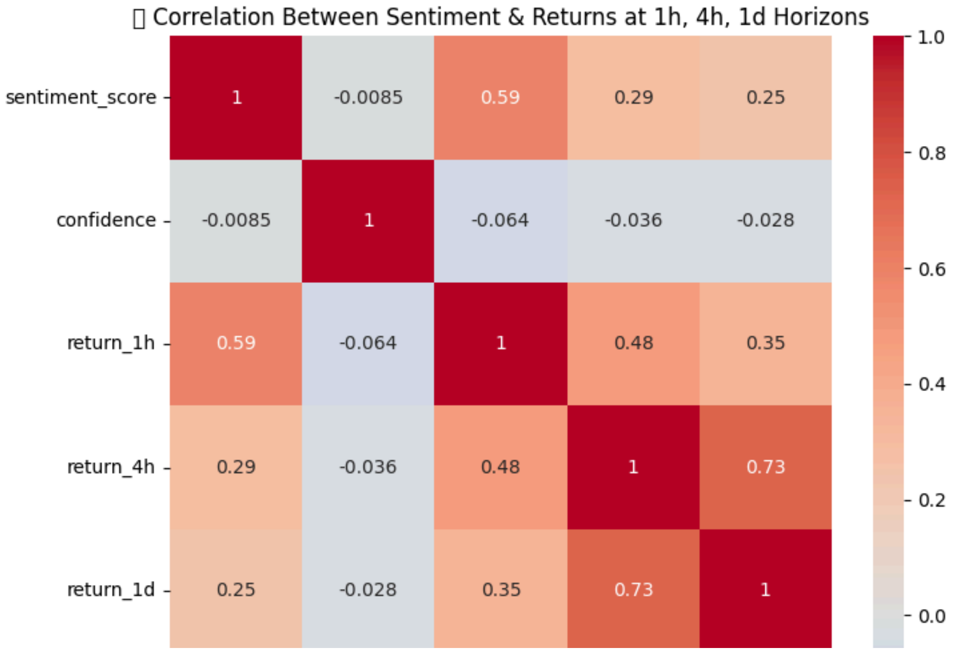
```
📁 Upload your CLEANED PRICE DATA file:
```

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

```
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data.csv
📁 Upload your HEADLINES WITH SENTIMENT file:
```

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

```
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025.csv
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) D
  fig.canvas.print_figure(bytes_io, **kw)
```

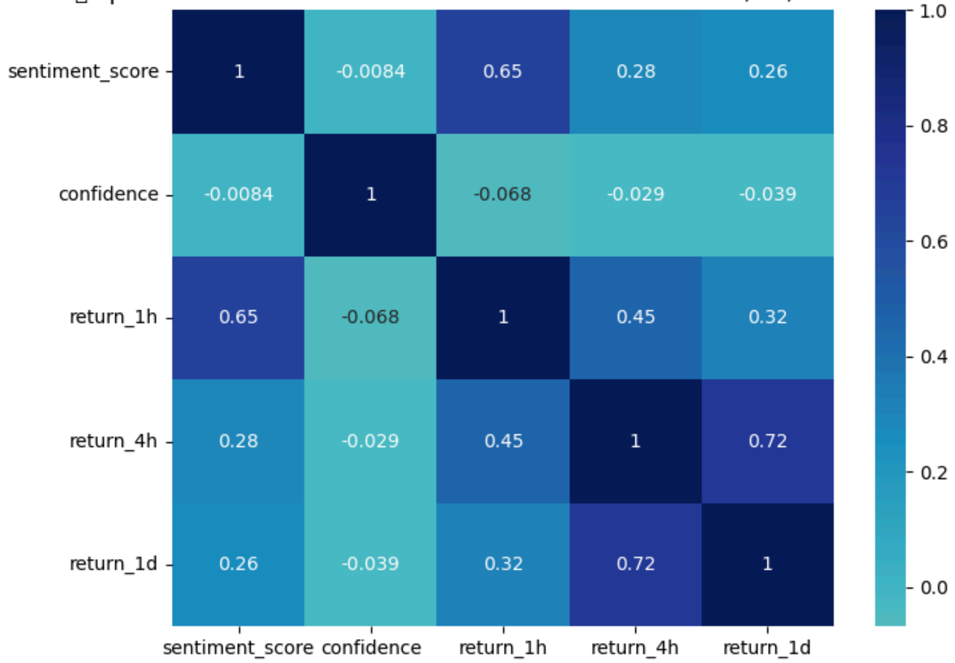


□ Correlation Between Sentiment & Returns at 1h, 4h, 1d Horizons

sentiment_score	1	-0.0085	0.59	0.29	0.25
confidence	-0.0085	1	-0.064	-0.036	-0.028
return_1h	0.59	-0.064	1	0.48	0.35
return_4h	0.29	-0.036	0.48	1	0.73
return_1d	0.25	-0.028	0.35	0.73	1

```
import seaborn as sns
import matplotlib.pyplot as plt

# Drop missing values (just like in Pearson)
spearman_df = merged[['sentiment_score', 'confidence', 'return_1h', 'return_4h', 'return_1d']].dropna()

# Compute Spearman correlation
spearman_corr = spearman_df.corr(method='spearman')

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(spearman_corr, annot=True, cmap='YlGnBu', center=0)
plt.title(" Spearman Correlation Between Sentiment & Returns at 1h, 4h, 1d Horizons")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) D
  fig.canvas.print_figure(bytes_io, **kw)
```

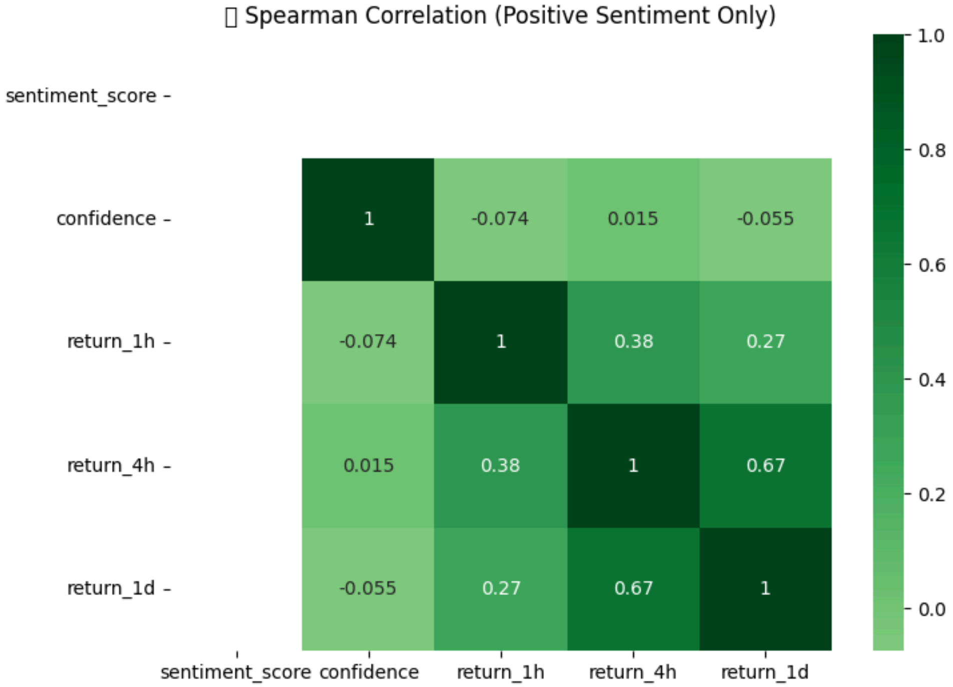


□ Spearman Correlation Between Sentiment & Returns at 1h, 4h, 1d Horizons

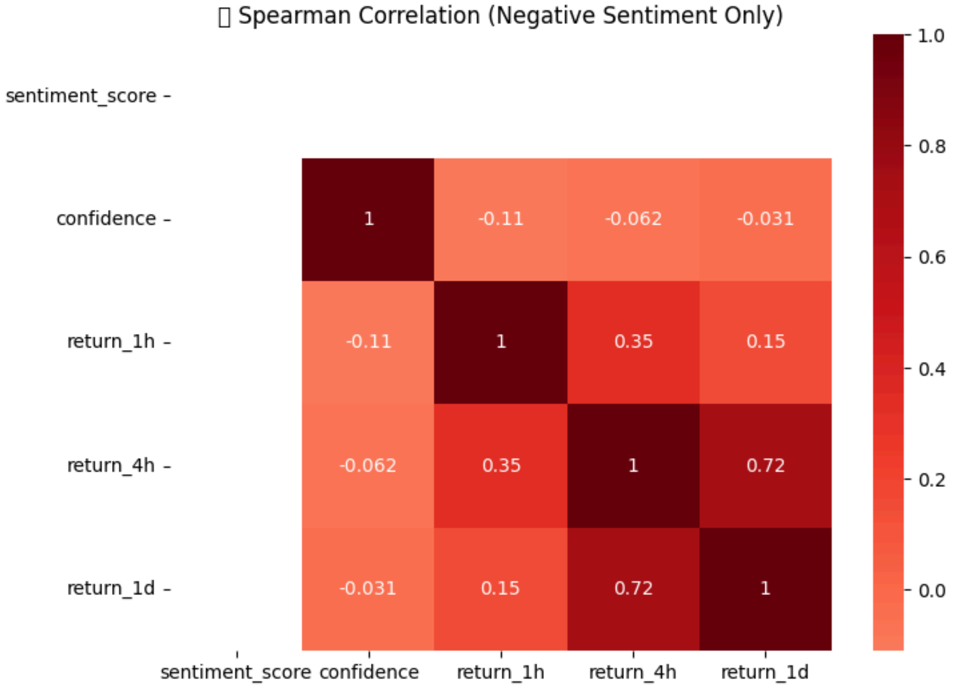
	sentiment_score	confidence	return_1h	return_4h	return_1d
sentiment_score	1	-0.0084	0.65	0.28	0.26
confidence	-0.0084	1	-0.068	-0.029	-0.039
return_1h	0.65	-0.068	1	0.45	0.32
return_4h	0.28	-0.029	0.45	1	0.72
return_1d	0.26	-0.039	0.32	0.72	1

```
import seaborn as sns
import matplotlib.pyplot as plt

# STEP 1: Filter for Positive and Negative Sentiment
positive_df = merged[merged['sentiment_score'] > 0][['sentiment_score', 'confidence', 'return_1h', 'return_4h', 'return_1d']].dropna()
negative_df = merged[merged['sentiment_score'] < 0][['sentiment_score', 'confidence', 'return_1h', 'return_4h', 'return_1d']].dropna()

# STEP 2: Compute Spearman Correlation
spearman_pos = positive_df.corr(method='spearman')
spearman_neg = negative_df.corr(method='spearman')

# STEP 3: Plot Positive Sentiment Correlation
plt.figure(figsize=(8, 6))
sns.heatmap(spearman_pos, annot=True, cmap='Greens', center=0)
plt.title(" Spearman Correlation (Positive Sentiment Only)")
plt.show()

# STEP 4: Plot Negative Sentiment Correlation
plt.figure(figsize=(8, 6))
sns.heatmap(spearman_neg, annot=True, cmap='Reds', center=0)
plt.title("Spearman Correlation (Negative Sentiment Only)")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) missing
  fig.canvas.print_figure(bytes_io, **kw)
```



```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missi
  fig.canvas.print_figure(bytes_io, **kw)
```



```
# STEP 1: Install required package
!pip install -q pandas numpy statsmodels

# STEP 2: Upload both files again
from google.colab import files
import pandas as pd
import numpy as np

print("Upload CLEANED PRICE DATA:")
price_file = files.upload()
price_filename = next(iter(price_file))

print("\n Upload HEADLINES with sentiment:")
headline_file = files.upload()
headline_filename = next(iter(headline_file))
```

```
# STEP 3: Load both files
price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 4: Merge
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])

# STEP 5: Feature engineering
merged_df['next_close_1h'] = merged_df.groupby('company')['close'].shift(-1)
merged_df['next_close_4h'] = merged_df.groupby('company')['close'].shift(-4)
merged_df['next_close_1d'] = merged_df.groupby('company')['close'].shift(-7)  # Assuming 7 1h bars = 1 day

merged_df['return_1h'] = (merged_df['next_close_1h'] - merged_df['close']) / merged_df['close']
merged_df['return_4h'] = (merged_df['next_close_4h'] - merged_df['close']) / merged_df['close']
merged_df['return_1d'] = (merged_df['next_close_1d'] - merged_df['close']) / merged_df['close']

# Lagged return
merged_df['lagged_return'] = merged_df.groupby('company')['close'].pct_change().shift(1)

# Volatility (3-hour rolling std dev of returns)
merged_df['volatility_3h'] = merged_df.groupby('company')['close'].pct_change().rolling(window=3).std().shift(1)

# Drop incomplete rows
merged_df = merged_df.dropna(subset=['sentiment_score', 'return_1h', 'return_4h', 'return_1d',
                                     'lagged_return', 'volatility_3h'])

print("✅ Data is ready! Now you can run the Granger causality tests.")
```

```
📁 Upload CLEANED PRICE DATA:
Choose Files No file chosen       Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (1).csv

📁 Upload HEADLINES with sentiment:
Choose Files No file chosen       Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (1).csv
```

```
# STEP 1: Install required libraries
!pip install -q pandas numpy statsmodels

# STEP 2: Upload both datasets
from google.colab import files
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import grangercausalitytests

print("📁 Upload CLEANED PRICE DATA:")
price_upload = files.upload()
price_file = next(iter(price_upload))

print("\n📁 Upload HEADLINES WITH SENTIMENT:")
headline_upload = files.upload()
headline_file = next(iter(headline_upload))

# STEP 3: Load and merge
price_df = pd.read_csv(price_file, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_file, parse_dates=['timestamp'])

# Merge on company and timestamp
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])

# STEP 4: Feature Engineering
merged_df['next_close_1h'] = merged_df.groupby('company')['close'].shift(-1)
merged_df['next_close_4h'] = merged_df.groupby('company')['close'].shift(-4)
merged_df['next_close_1d'] = merged_df.groupby('company')['close'].shift(-7)

merged_df['return_1h'] = (merged_df['next_close_1h'] - merged_df['close']) / merged_df['close']
merged_df['return_4h'] = (merged_df['next_close_4h'] - merged_df['close']) / merged_df['close']
merged_df['return_1d'] = (merged_df['next_close_1d'] - merged_df['close']) / merged_df['close']

merged_df['lagged_return'] = merged_df.groupby('company')['close'].pct_change().shift(1)
merged_df['volatility_3h'] = merged_df.groupby('company')['close'].pct_change().rolling(window=3).std().shift(1)
```

```
# Clean data
merged_df = merged_df.dropna(subset=[
    'sentiment_score', 'volatility_3h', 'lagged_return',
    'return_1h', 'return_4h', 'return_1d'
])

# STEP 5: Define Granger function (1 predictor → 1 target)
def single_granger_test(df, cause, effect, max_lag=4):
    print(f"\n➡️ Granger Causality Test: Does {cause} → {effect}?")
    try:
        test_df = df[[effect, cause]].dropna()
        grangercausalitytests(test_df, maxlag=max_lag, verbose=True)
    except Exception as e:
        print(f"⚠️ Skipping {cause} → {effect}: {e}")

# STEP 6: Run tests
predictors = ['sentiment_score', 'volatility_3h', 'lagged_return']
returns = ['return_1h', 'return_4h', 'return_1d']

for r in returns:
    for p in predictors:
        single_granger_test(merged_df, p, r)

# Plot
plt.figure(figsize=(10, 6))
for feature in granger_df["Feature"].unique():
    sub = granger_df[granger_df["Feature"] == feature]
    plt.plot(sub["Lag"], sub["p_value"], marker="o", label=feature)

plt.axhline(0.05, color="red", linestyle="--", label="Significance Threshold (0.05)")
plt.title("Granger Causality Test: Sentiment Features → 1H Returns")
plt.xlabel("Lag")
plt.ylabel("P-Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

enable.

```
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (2).csv

📁 Upload HEADLINES WITH SENTIMENT:
```

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

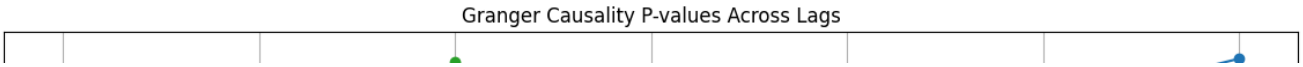
```
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (2).csv

➡️ Granger Causality Test: Does sentiment_score → return_1h?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=2.8328  , p=0.0929  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=2.8490  , p=0.0914  , df=1
likelihood ratio test: chi2=2.8413  , p=0.0919  , df=1
parameter F test:         F=2.8328  , p=0.0929  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.5138  , p=0.5985  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=1.0374  , p=0.5953  , df=2
likelihood ratio test: chi2=1.0364  , p=0.5956  , df=2
parameter F test:         F=0.5138  , p=0.5985  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.4921  , p=0.6879  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=1.4960  , p=0.6832  , df=3
likelihood ratio test: chi2=1.4939  , p=0.6837  , df=3
parameter F test:         F=0.4921  , p=0.6879  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.3578  , p=0.8386  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=1.4561  , p=0.8344  , df=4
likelihood ratio test: chi2=1.4541  , p=0.8347  , df=4
parameter F test:         F=0.3578  , p=0.8386  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does volatility_3h → return_1h?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.1982  , p=0.6563  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=0.1994  , p=0.6552  , df=1
likelihood ratio test: chi2=0.1993  , p=0.6553  , df=1
parameter F test:         F=0.1982  , p=0.6563  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=2.0165  , p=0.1341  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=4.0717  , p=0.1306  , df=2
likelihood ratio test: chi2=4.0560  , p=0.1316  , df=2
parameter F test:         F=2.0165  , p=0.1341  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.9270  , p=0.1242  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=5.8589  , p=0.1187  , df=3
likelihood ratio test: chi2=5.8266  , p=0.1204  , df=3
parameter F test:         F=1.9270  , p=0.1242  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=1.6668  , p=0.1563  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=6.7831  , p=0.1478  , df=4
likelihood ratio test: chi2=6.7397  , p=0.1503  , df=4
parameter F test:         F=1.6668  , p=0.1563  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does lagged_return → return_1h?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.3971  , p=0.5289  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=0.3993  , p=0.5274  , df=1
likelihood ratio test: chi2=0.3992  , p=0.5275  , df=1
parameter F test:         F=0.3971  , p=0.5289  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.1823  , p=0.8334  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=0.3680  , p=0.8319  , df=2
likelihood ratio test: chi2=0.3679  , p=0.8320  , df=2
```

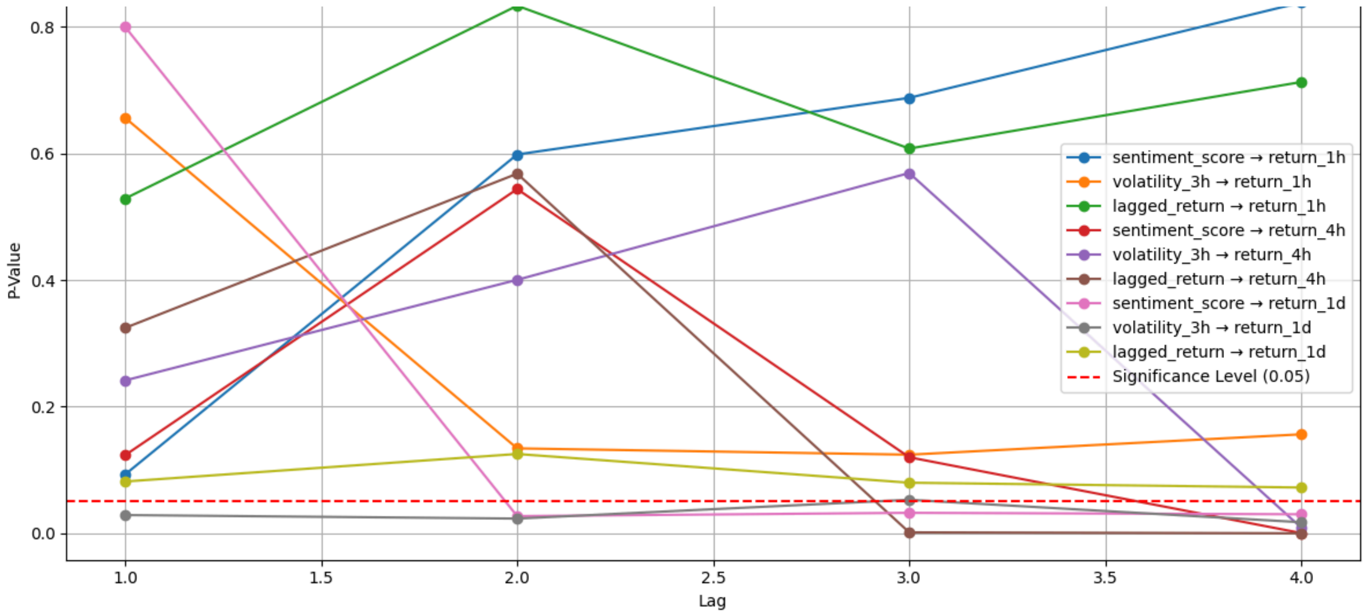
```
parameter F test:         F=0.1823  , p=0.8334  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.6114  , p=0.6078  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=1.8589  , p=0.6022  , df=3
likelihood ratio test: chi2=1.8556  , p=0.6029  , df=3
parameter F test:         F=0.6114  , p=0.6078  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.5311  , p=0.7130  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=2.1613  , p=0.7061  , df=4
likelihood ratio test: chi2=2.1569  , p=0.7069  , df=4
parameter F test:         F=0.5311  , p=0.7130  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does sentiment_score → return_4h?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=2.3808  , p=0.1234  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=2.3943  , p=0.1218  , df=1
likelihood ratio test: chi2=2.3889  , p=0.1222  , df=1
parameter F test:         F=2.3808  , p=0.1234  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.6099  , p=0.5438  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=1.2314  , p=0.5403  , df=2
likelihood ratio test: chi2=1.2299  , p=0.5407  , df=2
parameter F test:         F=0.6099  , p=0.5438  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.9530  , p=0.1201  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=5.9378  , p=0.1147  , df=3
likelihood ratio test: chi2=5.9046  , p=0.1163  , df=3
parameter F test:         F=1.9530  , p=0.1201  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=5.3634  , p=0.0003  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=21.8269 , p=0.0002  , df=4
likelihood ratio test: chi2=21.3862 , p=0.0003  , df=4
parameter F test:         F=5.3634  , p=0.0003  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does volatility_3h → return_4h?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=1.3748  , p=0.2415  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=1.3826  , p=0.2397  , df=1
likelihood ratio test: chi2=1.3808  , p=0.2400  , df=1
parameter F test:         F=1.3748  , p=0.2415  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.9168  , p=0.4004  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=1.8511  , p=0.3963  , df=2
likelihood ratio test: chi2=1.8479  , p=0.3969  , df=2
parameter F test:         F=0.9168  , p=0.4004  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.6725  , p=0.5692  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=2.0447  , p=0.5632  , df=3
likelihood ratio test: chi2=2.0407  , p=0.5640  , df=3
parameter F test:         F=0.6725  , p=0.5692  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=3.4287  , p=0.0088  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=13.9536 , p=0.0074  , df=4
likelihood ratio test: chi2=13.7718 , p=0.0081  , df=4
parameter F test:         F=3.4287  , p=0.0088  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does lagged_return → return_4h?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.9726  , p=0.3245  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=0.9781  , p=0.3227  , df=1
likelihood ratio test: chi2=0.9772  , p=0.3229  , df=1
parameter F test:         F=0.9726  , p=0.3245  , df_denom=526, df_num=1
```

```
Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.5660  , p=0.5682  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=1.1428  , p=0.5647  , df=2
likelihood ratio test: chi2=1.1415  , p=0.5651  , df=2
parameter F test:         F=0.5660  , p=0.5682  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=5.2468  , p=0.0014  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=15.9524 , p=0.0012  , df=3
likelihood ratio test: chi2=15.7157 , p=0.0013  , df=3
parameter F test:         F=5.2468  , p=0.0014  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=13.3393 , p=0.0000  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=54.2860 , p=0.0000  , df=4
likelihood ratio test: chi2=51.6637 , p=0.0000  , df=4
parameter F test:         F=13.3393 , p=0.0000  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does sentiment_score → return_1d?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.0643  , p=0.7999  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=0.0647  , p=0.7992  , df=1
likelihood ratio test: chi2=0.0647  , p=0.7992  , df=1
parameter F test:         F=0.0643  , p=0.7999  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=3.6332  , p=0.0271  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=7.3358  , p=0.0255  , df=2
likelihood ratio test: chi2=7.2853  , p=0.0262  , df=2
parameter F test:         F=3.6332  , p=0.0271  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.9497  , p=0.0323  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=8.9681  , p=0.0297  , df=3
likelihood ratio test: chi2=8.8927  , p=0.0308  , df=3
parameter F test:         F=2.9497  , p=0.0323  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=2.7004  , p=0.0300  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=10.9895 , p=0.0267  , df=4
likelihood ratio test: chi2=10.8763 , p=0.0280  , df=4
parameter F test:         F=2.7004  , p=0.0300  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does volatility_3h → return_1d?

Granger Causality
number of lags (no zero) 1
ssr based F test:         F=4.8064  , p=0.0288  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=4.8338  , p=0.0279  , df=1
likelihood ratio test: chi2=4.8118  , p=0.0283  , df=1
parameter F test:         F=4.8064  , p=0.0288  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=3.7902  , p=0.0232  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=7.6528  , p=0.0218  , df=2
likelihood ratio test: chi2=7.5979  , p=0.0224  , df=2
parameter F test:         F=3.7902  , p=0.0232  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.5794  , p=0.0529  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=7.8425  , p=0.0494  , df=3
likelihood ratio test: chi2=7.7847  , p=0.0507  , df=3
parameter F test:         F=2.5794  , p=0.0529  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=3.0258  , p=0.0174  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=12.3139 , p=0.0152  , df=4
likelihood ratio test: chi2=12.1720 , p=0.0161  , df=4
parameter F test:         F=3.0258  , p=0.0174  , df_denom=517, df_num=4

➡️ Granger Causality Test: Does lagged_return → return_1d?

Granger Causality
number of lags (no zero) 1
number of lags (no zero) 1
ssr based F test:         F=3.0403  , p=0.0818  , df_denom=526, df_num=1
ssr based chi2 test:   chi2=3.0577  , p=0.0804  , df=1
likelihood ratio test: chi2=3.0489  , p=0.0808  , df=1
parameter F test:         F=3.0403  , p=0.0818  , df_denom=526, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=2.0854  , p=0.1253  , df_denom=523, df_num=2
ssr based chi2 test:   chi2=4.2106  , p=0.1218  , df=2
likelihood ratio test: chi2=4.1939  , p=0.1228  , df=2
parameter F test:         F=2.0854  , p=0.1253  , df_denom=523, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.2665  , p=0.0799  , df_denom=520, df_num=3
ssr based chi2 test:   chi2=6.8909  , p=0.0755  , df=3
likelihood ratio test: chi2=6.8462  , p=0.0770  , df=3
parameter F test:         F=2.2665  , p=0.0799  , df_denom=520, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=2.1604  , p=0.0723  , df_denom=517, df_num=4
ssr based chi2 test:   chi2=8.7922  , p=0.0665  , df=4
likelihood ratio test: chi2=8.7195  , p=0.0685  , df=4
parameter F test:         F=2.1604  , p=0.0723  , df_denom=517, df_num=4
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-1369502691.py in <cell line: 0>()
     62 # Plot
     63 plt.figure(figsize=(10, 6))
---> 64 for feature in granger_df["Feature"].unique():
     65     sub = granger_df[granger_df["Feature"] == feature]
     66     plt.plot(sub["Lag"], sub["p_value"], marker="o", label=feature)

NameError: name 'granger_df' is not defined

<Figure size 1000x600 with 0 Axes>
```

```
# STEP 1: Install required libraries (only in Colab)
!pip install -q pandas numpy matplotlib statsmodels

# STEP 2: Upload Datasets
from google.colab import files
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import grangercausalitytests

print("📁 Upload Price Data File:")
price_upload = files.upload()
price_file = next(iter(price_upload))

print("\n📁 Upload Headline Sentiment File:")
headline_upload = files.upload()
headline_file = next(iter(headline_upload))

# STEP 3: Read and Merge Data
price_df = pd.read_csv(price_file, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_file, parse_dates=['timestamp'])

# Merge on timestamp and company
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df = merged_df.sort_values(['company', 'timestamp'])

# STEP 4: Feature Engineering
merged_df['next_close_1h'] = merged_df.groupby('company')['close'].shift(-1)
merged_df['next_close_4h'] = merged_df.groupby('company')['close'].shift(-4)
merged_df['next_close_1d'] = merged_df.groupby('company')['close'].shift(-7)

merged_df['return_1h'] = (merged_df['next_close_1h'] - merged_df['close']) / merged_df['close']
merged_df['return_4h'] = (merged_df['next_close_4h'] - merged_df['close']) / merged_df['close']
merged_df['return_1d'] = (merged_df['next_close_1d'] - merged_df['close']) / merged_df['close']

merged_df['lagged_return'] = merged_df.groupby('company')['close'].pct_change().shift(1)
merged_df['volatility_3h'] = merged_df.groupby('company')['close'].pct_change().rolling(window=3).std().shift(1)

# Clean the dataset
merged_df = merged_df.dropna(subset=[
    'sentiment_score', 'volatility_3h', 'lagged_return',
    'return_1h', 'return_4h', 'return_1d'
])

# STEP 5: Granger Causality Test & Results Collection
def granger_results(df, cause, effect, max_lag=4):
    results = []
    for lag in range(1, max_lag+1):
        try:
            test_df = df[[effect, cause]].dropna()
            gc_result = grangercausalitytests(test_df, maxlag=lag, verbose=False)
            p_val = gc_result[lag][0]['ssr_ftest'][1]
            results.append({'Feature': cause, 'Target': effect, 'Lag': lag, 'p_value': p_val})
        except:
            continue
    return results

# STEP 6: Run all combinations and collect results
predictors = ['sentiment_score', 'volatility_3h', 'lagged_return']
targets = ['return_1h', 'return_4h', 'return_1d']

all_results = []
for t in targets:
    for p in predictors:
        all_results.extend(granger_results(merged_df, cause=p, effect=t))

granger_df = pd.DataFrame(all_results)

# STEP 7: Plot Granger Causality P-values
plt.figure(figsize=(12, 6))
for target in granger_df['Target'].unique():
    subset = granger_df[granger_df['Target'] == target]
    for feature in subset['Feature'].unique():
        sub = subset[subset['Feature'] == feature]
        plt.plot(sub['Lag'], sub['p_value'], marker='o', label=f"{feature} → {target}")
plt.axhline(0.05, color='red', linestyle='--', label='Significance Level (0.05)')
plt.title("Granger Causality P-values Across Lags")
plt.xlabel("Lag")
plt.ylabel("P-Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
📁 Upload Price Data File:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (3).csv

📁 Upload Headline Sentiment File:
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (3).csv
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is deprecated since functions should
  warnings.warn(
```





```
import pandas as pd
from google.colab import files

# STEP 1: Upload both files
print("📁 Please upload your Cleaned Price Data (1h data)")
uploaded_price = files.upload()

print("\n📁 Now upload your Sentiment Headline file")
uploaded_headlines = files.upload()

# STEP 2: Read files
price_filename = next(iter(uploaded_price))
headline_filename = next(iter(uploaded_headlines))

price_df = pd.read_csv(price_filename, parse_dates=['timestamp'])
headline_df = pd.read_csv(headline_filename, parse_dates=['timestamp'])

# STEP 3: Merge on company and timestamp
merged_df = pd.merge(price_df, headline_df, on=['company', 'timestamp'], how='inner')
merged_df.sort_values(['company', 'timestamp'], inplace=True)

# STEP 4: Calculate returns
merged_df['return_1h'] = merged_df.groupby('company')['close'].pct_change(periods=-1)
merged_df['return_4h'] = merged_df.groupby('company')['close'].pct_change(periods=-4)
merged_df['return_1d'] = merged_df.groupby('company')['close'].pct_change(periods=-24)

# STEP 5: Add lagged return and 3-hour rolling volatility
merged_df['lagged_return'] = merged_df.groupby('company')['close'].pct_change()
merged_df['volatility_3h'] = (
    merged_df.groupby('company')['close'].pct_change().rolling(3).std().reset_index(level=0, drop=True)
)

# STEP 6: Drop missing values
merged_df.dropna(subset=[
    'sentiment_score', 'confidence', 'return_1h', 'return_4h', 'return_1d',
    'lagged_return', 'volatility_3h'
], inplace=True)

# STEP 7: Save merged file
output_filename = "merged_df_with_features.csv"
merged_df.to_csv(output_filename, index=False)

print(f"\n✅ Merged and feature-engineered file saved as: {output_filename}")
files.download(output_filename)
```

```
📁 Please upload your Cleaned Price Data (1h data)
```

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

```
Saving Cleaned_Price_Data.csv to Cleaned_Price_Data (3).csv
```

```
📁 Now upload your Sentiment Headline file
```

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

```
Saving PriceDriven_Headlines_2021_2025.csv to PriceDriven_Headlines_2021_2025 (3).csv
```

```
# STEP 1: Install required package
!pip install arch --quiet

# STEP 2: Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from google.colab import files

# STEP 3: Upload merged dataset
print(" Upload the merged dataset (with returns & sentiment):")
uploaded = files.upload()
filename = next(iter(uploaded))
df = pd.read_csv(filename, parse_dates=['timestamp'])

# STEP 4: Clean and prepare data
df = df.sort_values(by='timestamp')
df = df.dropna(subset=['return_1h', 'confidence'])  # Ensure no NaNs

returns = df['return_1h'] * 100  # Scale for EGARCH
sentiment = df['confidence']    # Exogenous regressor in mean equation
timestamps = df['timestamp']

# STEP 5: Fit EGARCH(1,1) with sentiment in mean equation
model = arch_model(
    returns,
    vol='EGARCH',
    p=1,
    q=1,
    mean='LS',     # Linear regression for mean
    x=sentiment,   # Regressor in mean equation
    dist='normal'
)

results = model.fit(disp='inline')

# STEP 6: Plot GARCH Volatility vs Sentiment Confidence
plt.figure(figsize=(12, 6))
plt.plot(timestamps, results.conditional_volatility, label="EGARCH Volatility", color='blue')
plt.plot(timestamps, sentiment * 10, label="Sentiment Confidence (×10)", color='orange', alpha=0.6)

plt.title("EGARCH Volatility vs. Sentiment Confidence Over Time")
plt.xlabel("Timestamp")
plt.ylabel("Volatility / Scaled Sentiment")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```