coursework-2

January 13, 2026

1 ECS764P Applied Statistics - Coursework 2

- **Professors:** Dr. Frederik Dahlqvist
- **Due Date:** Thursday, 15. January 2026 (12:00 GMT)
- **Student:** Philipp Schmidt
- **Python Version:** 3.11.14

[218]:
```
# Download the required dependencies
import os

if os.path.exists('requirements.txt'):
    %pip install -q -r requirements.txt
else:
    print("requirements.txt not found. Installing packages manually...")
    %pip install -q matplotlib numpy pandas requests scipy tqdm yfinance
```

```
Note: you may need to restart the kernel to use updated packages.
```

1.1 Task 1: Explain Dataset Choice

We want to perform a test to check for the correlation between a stock's investor attention and its volatility at the London Stock Exchange.

We assume there is a decent volatility observable as usually news drives both attention (visible in Wikipedia visits) and stock price moves. Unusual attention spikes might indicate news events, earnings announcements, or sentiment shifts that drive trading activity.

We use the absolute value of log returns as a standard measure of realized volatility and the daily percentage change in Wikipedia pageviews to capture sudden spikes or drops in public interest that may correlate with market movements.

1.2 Task 2: Fetch and Preprocess Data from External API

For this we are taking the biggest stocks traded at the London Stock Exchange by market capitalisation (based on Dec. 2025). The task only asks for one stock but, out of personal interest, we're doing more (if you only want one then set `use_multiple_stocks = False`).

We fetch the data using the `yfinance` library.

[219]:
```
# Flag
use_multiple_stocks = True

if not(use_multiple_stocks):
    stocks = {
        "RR.L": "Rolls-Royce_Holdings"
    }
```

To make sure that the dataset is available during the grading process and the risk that Wikimedia API requests are rate-limited/refused, as every date has to get fetched and this might trigger the API's bots policy we chose the way of uploading the data beforehand as a `.csv` file to GitHub and now fetching it from there. If you want to directly fetch the Wikimedia API set `use_wikimedia_api = True`.

[220]:
```
# Flag
use_wikimedia_api = False
```

For each stock entry we want to retrieve the page views of their Wikipedia website and calculate a percentage change of the days before.

[221]:
```
# Define the stocks we want to analyze with their Wikipedia article names
# These are the largest stocks on the London Stock Exchange by market cap (Dec␣
 ↪2025)
if use_multiple_stocks:
    stocks = {
        "AZN.L": "AstraZeneca",
        "HSBA.L": "HSBC",
        "SHEL.L": "Shell_plc",
        "ULVR.L": "Unilever",
        "RR.L": "Rolls-Royce_Holdings",
        "BATS.L": "British_American_Tobacco",
        "RIO.L": "Rio_Tinto_(corporation)",
        "GSK.L": "GSK_plc",
        "BP.L": "BP",
        "BARC.L": "Barclays",
    }
```

[222]:
```
from pathlib import Path
from typing import Tuple
import pandas as pd
import numpy as np
import requests
import urllib.parse
import yfinance as yf
from tqdm import tqdm

def fetch_stock_data_from_external_apis(stocks: dict) -> Tuple[pd.DataFrame,␣
 ↪Path]:
```

```
    """
    Fetch stock price data and Wikipedia pageview statistics for given stock␣
↪tickers.

    This function retrieves historical stock closing prices using yfinance,␣
↪calculates
    logarithmic returns, and fetches Wikipedia pageview counts for the␣
↪corresponding
    company articles. It also computes the daily percentage change in pageviews.

    Args
    ----
    stocks : dict
        Dictionary mapping stock tickers (str) to Wikipedia article names (str).

    Returns
    -------
    stock_information_df: pd.DataFrame
        Multi-index DataFrame with columns:
            - ('Log_Returns', ticker): Daily logarithmic returns.
            - ('Views', ticker): Wikipedia pageviews count.
            - ('Views_Change', ticker): Daily percentage change in pageviews.

    stock_information_csv: Path
        Path pointing to the CSV file where the DataFrame is saved.

    Raises
    ------
    RuntimeError
        If yfinance returns no data for the provided tickers.
    """

    # Extract ticker symbols from the input dictionary
    stock_tickers = list(stocks.keys())

    # Download daily stock price data from yfinance
    dl = yf.download(stock_tickers, start='2023-11-29', end='2025-12-02',␣
↪interval="1d", auto_adjust=True)
    if dl is None or (hasattr(dl, "empty") and dl.empty):
        raise RuntimeError(f"yfinance returned no data for tickers:␣
↪{stock_tickers}")

    try:
        # Extract the 'Close' price column and remove any missing values
        stock_information_df = dl["Close"].dropna()
    except Exception:
        # Handle MultiIndex columns that yfinance sometimes returns
```

```
        stock_information_df = dl.xs("Close", axis=1, level=0, drop_level=True).
↪dropna()

    # Dictionary to hold all computed columns for the final DataFrame
    df_dict = {}

    # Compute logarithmic returns for each stock
    # Log returns = ln(P_t / P_{t-1}) where P_t is the closing price at time t
    for ticker in stock_tickers:
        df_dict[('Log_Returns', ticker)] = np.log(stock_information_df[ticker] /
↪ stock_information_df[ticker].shift(1))

    # Initialise Wikipedia pageviews column with NaN values
    for ticker in stock_tickers:
        df_dict[('Views', ticker)] = np.nan

    # Initialise Wikipedia pageviews change column with NaN values
    for ticker in stock_tickers:
        df_dict[('Views_Change', ticker)] = np.nan

    # Create a new DataFrame with multi-index columns containing all metrics
    stock_information_df = pd.DataFrame(df_dict, index=stock_information_df.
↪index)

    # Delete first row as it doesn't contain a valid log return (requires␣
↪previous day)
    stock_information_df = stock_information_df.iloc[1:]

    # Set up HTTP headers for Wikimedia API requests
    # User-Agent is required by Wikimedia's bot policy
    headers = {
        "Accept": "application/json",
        "User-Agent": "qmul-student-agent/1.0 (mailto:student@qmul.ac.uk)",
    }

    # Create a session to reuse connections and headers across requests
    session = requests.Session()
    session.headers.update(headers)

    # Get all dates for which we need to fetch Wikipedia data
    search_dates = stock_information_df.index

    # Calculate total API requests needed for progress tracking
    total_requests = len(stocks) * len(search_dates)
    pbar = tqdm(total=total_requests, desc="Fetching Wikipedia pageviews")

    # Iterate through each stock and fetch Wikipedia pageview data
```

```
    for ticker, article in stocks.items():
        # URL-encode the Wikipedia article name to handle special characters
        article_enc = urllib.parse.quote(str(article), safe="")

        # Fetch pageview data for each date
        for date in search_dates:
            # Format date as YYYYMMDD for the Wikimedia API
            d = pd.Timestamp(date).strftime("%Y%m%d")

            # Construct the Wikimedia Pageviews API URL
            # Fetches daily pageviews for a specific article on a specific date
            url = (
                "https://wikimedia.org/api/rest_v1/metrics/pageviews/
↪per-article/"
                f"en.wikipedia.org/all-access/all-agents/{article_enc}/daily/
↪{d}/{d}"
            )

            # Make the API request
            response = session.get(url)

            # Parse the response if successful
            if response.ok:
                data = response.json()
                items = data.get("items", [])
                # Extract views count from the first item, or None if no data
                views = items[0]["views"] if items else None
            else:
                views = None
                print(f"\nError fetching {ticker} on {date.
↪strftime('%Y-%m-%d')}: {response.status_code}")

            # Store the pageview count in the DataFrame
            stock_information_df.loc[date, ('Views', ticker)] = views

            # Update progress bar with current stock and date
            pbar.set_postfix_str(f"{ticker} - {date.strftime('%Y-%m-%d')}")
            pbar.update(1)

    # Close the progress bar
    pbar.close()

    # Calculate the daily percentage change in Wikipedia pageviews
    # Formula: (Views_t - Views_{t-1}) / Views_{t-1}
    for ticker in stock_tickers:
        views_col = ('Views', ticker)
        views_change_col = ('Views_Change', ticker)
```

```
        stock_information_df[views_change_col] = (
            (stock_information_df[views_col] - stock_information_df[views_col].
↪shift(1))
            / stock_information_df[views_col].shift(1)
        )

    # Delete first row as it doesn't contain a valid views change (requires␣
↪previous day)
    stock_information_df = stock_information_df.iloc[1:]

    # Write CSV to a Path and return that Path (to_csv returns None when␣
↪writing to a file)
    csv_path = Path('ecs764p-applied-statistics-coursework-2-dataset.csv')
    stock_information_df.to_csv(csv_path, index=True)

    return stock_information_df, csv_path
```

[223]:
```
import pandas as pd

def fetch_data_from_github() -> pd.DataFrame:
    """
    Fetch pre-computed stock and Wikipedia data from GitHub Gist.

    Returns
    -------
    pd.DataFrame
        DataFrame containing stock returns and Wikipedia pageview metrics.
    """
    path = 'https://gist.githubusercontent.com/PhilippXXY/
↪4511ed662bc78c847d9d65adf610d591/raw/
↪d4dec838c9caba906b1ebd3c1ad1f25774f5f53f/
↪ecs764p-applied-statistics-coursework-2-dataset.csv'
    df = pd.read_csv(
        path,
        index_col=0,
        parse_dates=True,
        date_format='%Y-%m-%d',
        header=[0, 1],
        skiprows=[2],
    )

    df.index.name = 'Date'

    # Clean up column names by stripping whitespace
    df.columns = pd.MultiIndex.from_tuples([
    (level0.strip(), level1.strip())
```

```
    for level0, level1 in df.columns
    ])

    # Convert all columns to numeric types
    for col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')

    return df
```

[224]:
```
if use_wikimedia_api:
    df, _ = fetch_stock_data_from_external_apis(stocks)
else:
    df = fetch_data_from_github()

# Drop absolute views column
df = df.drop(columns='Views')

# Add absolute returns (volatility proxy) to the dataframe
for ticker in stocks.keys():
    df[('Abs_Log_Returns', ticker)] = df[('Log_Returns', ticker)].abs()
```

[225]:
```
# Reorder columns: Log_Returns, Abs_Log_Returns, Views_Change
new_column_order = []

for ticker in stocks.keys():
    new_column_order.append(('Log_Returns', ticker))
    new_column_order.append(('Abs_Log_Returns', ticker))
    new_column_order.append(('Views_Change', ticker))

# Reindex the DataFrame with the new column order
df = df[new_column_order]
```

[226]:
```
# Display data
display(df)
```

	Log_Returns	Abs_Log_Returns	Views_Change	Log_Returns
	AZN.L	AZN.L	AZN.L	HSBA.L
Date				
2023-12-01	0.007085	0.007085	-0.126561	0.003316
2023-12-04	0.006646	0.006646	0.041945	0.000000
2023-12-05	-0.011955	0.011955	0.014639	-0.001159
2023-12-06	-0.001381	0.001381	0.021641	0.018554
2023-12-07	-0.003164	0.003164	-0.100618	-0.003749
…	…	…	…	…
2025-11-26	0.003398	0.003398	-0.148594	0.013050
2025-11-27	-0.009659	0.009659	0.049057	0.000564
2025-11-28	-0.002716	0.002716	0.046763	0.004871
2025-12-01	-0.009925	0.009925	-0.019759	0.008745

```
2025-12-02   -0.002605        0.002605    -0.073620    0.007567

           Abs_Log_Returns Views_Change Log_Returns Abs_Log_Returns  \
                     HSBA.L       HSBA.L      SHEL.L          SHEL.L
Date
2023-12-01        0.003316    -0.097157    0.003120        0.003120
2023-12-04        0.000000    -0.007643   -0.012342        0.012342
2023-12-05        0.001159     0.042206   -0.004148        0.004148
2023-12-06        0.018554     0.007390   -0.014354        0.014354
2023-12-07        0.003749     0.018779   -0.003621        0.003621
…                    …           …          …               …
2025-11-26        0.013050     0.001220    0.000181        0.000181
2025-11-27        0.000564     0.000000   -0.004892        0.004892
2025-11-28        0.004871     0.025173    0.011018        0.011018
2025-12-01        0.008745    -0.000396    0.006089        0.006089
2025-12-02        0.007567     0.091125   -0.004294        0.004294

           Views_Change Log_Returns  … Views_Change Log_Returns  \
                 SHEL.L      ULVR.L  …         RIO.L       GSK.L
Date                                 …
2023-12-01    -0.057082    0.002914  …    -0.150000    0.011627
2023-12-04     0.055530    0.009609  …     0.521569    0.013419
2023-12-05    -0.019836   -0.003937  …    -0.115979   -0.010915
2023-12-06     0.011879    0.000263  …     0.068027    0.000278
2023-12-07    -0.020870    0.001314  …    -0.106460   -0.005152
…                …          …  …           …          …
2025-11-26     0.047800   -0.006834  …    -0.184795    0.010529
2025-11-27    -0.119233   -0.000221  …    -0.045911   -0.006637
2025-11-28     0.024720    0.005296  …    -0.049624   -0.006123
2025-12-01    -0.042504    0.003296  …     0.123418    0.005013
2025-12-02     0.109778   -0.016811  …     0.087324    0.010500

           Abs_Log_Returns Views_Change Log_Returns Abs_Log_Returns  \
                     GSK.L       GSK.L       BP.L            BP.L
Date
2023-12-01        0.011627    -0.059398   -0.001462        0.001462
2023-12-04        0.013419     0.028547   -0.013574        0.013574
2023-12-05        0.010915     0.106812   -0.000106        0.000106
2023-12-06        0.000278     0.500760   -0.012689        0.012689
2023-12-07        0.005152    -0.281519   -0.012309        0.012309
…                    …           …          …               …
2025-11-26        0.010529    -0.113695    0.007094        0.007094
2025-11-27        0.006637     0.010204   -0.012671        0.012671
2025-11-28        0.006123    -0.150072    0.015979        0.015979
2025-12-01        0.005013     0.275042    0.007348        0.007348
2025-12-02        0.010500     0.021305    0.000874        0.000874

           Views_Change Log_Returns Abs_Log_Returns Views_Change
```

```
                BP.L      BARC.L          BARC.L       BARC.L
Date
2023-12-01  -0.039457    0.009316        0.009316    -0.118083
2023-12-04   0.032542    0.004346        0.004346     0.049806
2023-12-05  -0.018858   -0.024927        0.024927     0.112754
2023-12-06   0.058452    0.005577        0.005577     0.139535
2023-12-07  -0.043532   -0.000713        0.000713    -0.232264
…                 …           …               …            …
2025-11-26   0.000971    0.031097        0.031097    -0.105200
2025-11-27  -0.107951    0.016415        0.016415     0.037408
2025-11-28   0.031884    0.000697        0.000697    -0.063104
2025-12-01   0.937149   -0.001512        0.001512     0.424055
2025-12-02  -0.114374    0.015705        0.015705    -0.041988

[505 rows x 30 columns]
```

[227]:
```
# Get basic descriptive statistics
descriptive_stats_df = df.describe(percentiles=[.01, .05, .25, .5, .75, .95, 0.
 ↪99])
descriptive_stats_df.loc['skew'] = df.skew()
descriptive_stats_df.loc['kurtosis'] = df.kurtosis()

print("*** Display descriptive statistics for all stocks ***")
display(descriptive_stats_df)
```

```
*** Display descriptive statistics for all stocks ***
```

	Log_Returns AZN.L	Abs_Log_Returns AZN.L	Views_Change AZN.L	Log_Returns HSBA.L	Abs_Log_Returns HSBA.L	\
count	505.000000	505.000000	505.000000	505.000000	505.000000	
mean	0.000614	0.010270	0.028234	0.001174	0.009748	
std	0.015105	0.011083	0.335532	0.014466	0.010745	
min	-0.087786	0.000000	-0.576711	-0.092856	0.000000	
1%	-0.039410	0.000167	-0.453647	-0.051883	0.000005	
5%	-0.019222	0.000523	-0.251317	-0.022639	0.000585	
25%	-0.006576	0.003720	-0.077844	-0.004531	0.003231	
50%	0.000518	0.007451	-0.009346	0.002536	0.006797	
75%	0.008029	0.013177	0.084104	0.008745	0.011902	
95%	0.022075	0.028783	0.242038	0.019551	0.029499	
99%	0.033467	0.057675	1.380536	0.035979	0.054488	
max	0.106290	0.106290	4.277303	0.054587	0.092856	
skew	-0.114261	3.620662	7.592235	-1.604481	3.227485	
kurtosis	9.536589	20.771961	79.836753	8.536110	15.866315	

	Views_Change HSBA.L	Log_Returns SHEL.L	Abs_Log_Returns SHEL.L	Views_Change SHEL.L	Log_Returns ULVR.L	\
count	505.000000	505.000000	505.000000	505.000000	505.000000	
mean	0.009836	0.000171	0.008722	0.006960	0.000345	

std	0.160802	0.012019	0.008262	0.136615	0.010410
min	-0.417422	-0.072278	0.000000	-0.490213	-0.058063
1%	-0.306036	-0.037632	0.000179	-0.238276	-0.024985
5%	-0.182383	-0.018520	0.000567	-0.143833	-0.014674
25%	-0.066174	-0.005575	0.003096	-0.058824	-0.005138
50%	-0.004980	0.000598	0.006291	-0.005319	0.000201
75%	0.051072	0.007004	0.012390	0.055530	0.005356
95%	0.236081	0.016865	0.023746	0.169144	0.015437
99%	0.603242	0.026221	0.037685	0.376476	0.030336
max	1.411674	0.035490	0.072278	1.738618	0.060451
skew	3.517138	-0.905643	2.334170	4.915579	0.324995
kurtosis	25.350637	3.845365	9.533993	55.618122	6.208731

	…	Views_Change	Log_Returns	Abs_Log_Returns	Views_Change	\
	…	RIO.L	GSK.L	GSK.L	GSK.L	
count	…	505.000000	505.000000	505.000000	505.000000	
mean	…	0.018270	0.000493	0.010318	0.034970	
std	…	0.214974	0.014887	0.010733	0.659400	
min	…	-0.568644	-0.100247	0.000000	-0.799713	
1%	…	-0.366679	-0.046742	0.000000	-0.344835	
5%	…	-0.220350	-0.021742	0.000419	-0.199423	
25%	…	-0.092031	-0.006598	0.003423	-0.073718	
50%	…	-0.011887	0.000981	0.007281	-0.002381	
75%	…	0.086356	0.007970	0.014706	0.074671	
95%	…	0.343627	0.021129	0.028393	0.236878	
99%	…	0.875574	0.035296	0.058419	0.500413	
max	…	1.518298	0.073331	0.100247	14.378190	
skew	…	2.649534	-0.547966	3.035128	20.516444	
kurtosis	…	13.409011	6.971745	15.470637	446.539786	

	Log_Returns	Abs_Log_Returns	Views_Change	Log_Returns	Abs_Log_Returns	\
	BP.L	BP.L	BP.L	BARC.L	BARC.L	
count	505.000000	505.000000	505.000000	505.000000	505.000000	
mean	-0.000087	0.010720	0.013856	0.002238	0.013972	
std	0.015305	0.010913	0.182515	0.019083	0.013175	
min	-0.078380	0.000000	-0.489202	-0.091140	0.000000	
1%	-0.050933	0.000102	-0.411219	-0.049987	0.000220	
5%	-0.024312	0.000766	-0.163557	-0.026795	0.001084	
25%	-0.006931	0.003467	-0.062709	-0.007327	0.004814	
50%	0.001062	0.007686	-0.005400	0.003202	0.010661	
75%	0.008218	0.013828	0.056019	0.012443	0.018768	
95%	0.019017	0.029745	0.242286	0.030975	0.039717	
99%	0.035975	0.058664	0.694487	0.052552	0.063874	
max	0.071045	0.078380	1.817283	0.082415	0.091140	
skew	-0.760640	2.609639	3.380681	-0.224677	2.195870	
kurtosis	4.754088	9.978367	25.106985	3.214735	6.816099	

	Views_Change

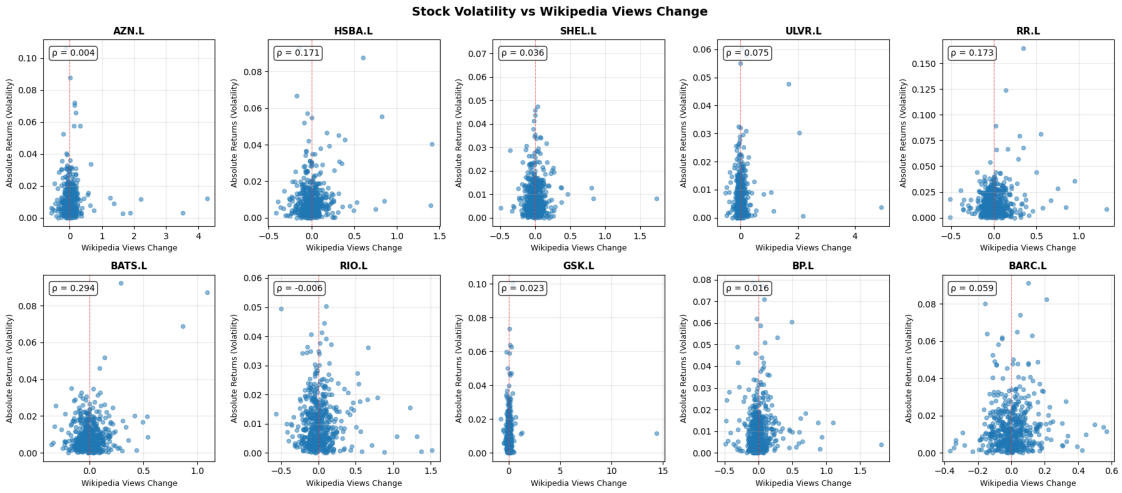
```
              BARC.L
count     505.000000
mean        0.006095
std         0.110582
min        -0.366844
1%         -0.280005
5%         -0.136140
25%        -0.056983
50%         0.000565
75%         0.055757
95%         0.178307
99%         0.392181
max         0.568243
skew        0.964210
kurtosis    4.327879

[14 rows x 30 columns]
```

[228]:
```
# Select all columns where the second level of the MultiIndex is 'AZN.L'
print("*** Display descriptive statistics for AstraZeneca only ***")
display(descriptive_stats_df.xs('RR.L', level=1, axis=1))
```

```
*** Display descriptive statistics for AstraZeneca only ***
          Log_Returns  Abs_Log_Returns  Views_Change
count      505.000000       505.000000    505.000000
mean         0.002691         0.014685      0.010619
std          0.020778         0.014930      0.159948
min         -0.124002         0.000000     -0.511819
1%          -0.040124         0.000251     -0.356633
5%          -0.028683         0.000960     -0.160858
25%         -0.008351         0.005307     -0.070392
50%          0.001295         0.011080     -0.012830
75%          0.013806         0.019997      0.070890
95%          0.030927         0.036232      0.268747
99%          0.056716         0.067682      0.577686
max          0.164735         0.164735      1.321646
skew         0.532748         3.875026      2.319337
kurtosis    10.713650        27.728711     14.104520
```

[229]:
```
import matplotlib.pyplot as plt

# Scatter plots: Views Change vs Volatility (Absolute Returns)
fig, axes = plt.subplots(2, 5, figsize=(18, 8))
axes = axes.flatten()

for i, ticker in enumerate(stocks.keys()):
    x = df[('Views_Change', ticker)]
```

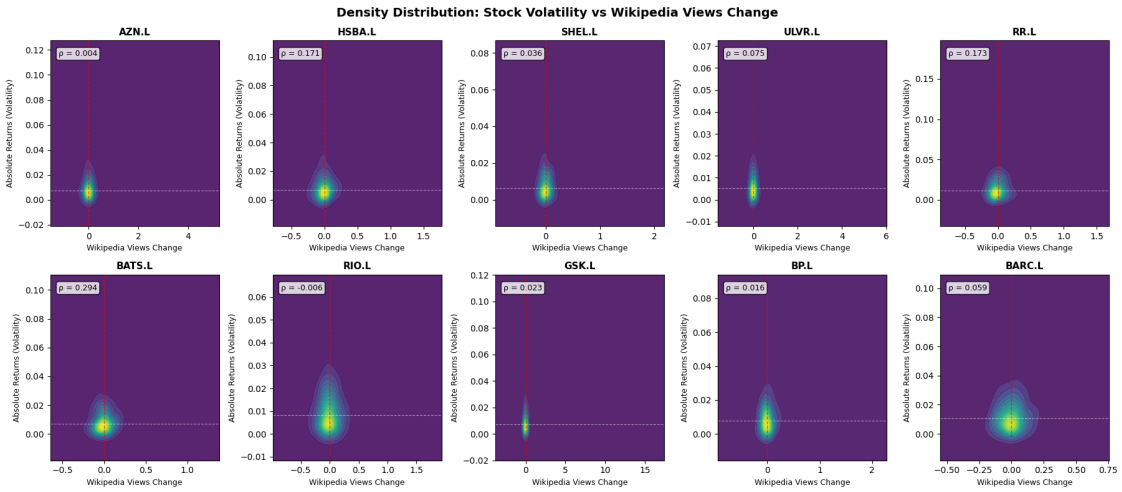
```
    y = df[('Abs_Log_Returns', ticker)]

    axes[i].scatter(x, y, alpha=0.5, s=20)
    axes[i].set_xlabel('Wikipedia Views Change', fontsize=9)
    axes[i].set_ylabel('Absolute Returns (Volatility)', fontsize=9)
    axes[i].set_title(ticker, fontsize=11, fontweight='bold')
    axes[i].grid(True, alpha=0.3)
    axes[i].axvline(x=0, color='red', linestyle='--', linewidth=0.8, alpha=0.5)

    corr = x.corr(y)
    axes[i].text(0.05, 0.95, f'  = {corr:.3f}',
                 transform=axes[i].transAxes,
                 verticalalignment='top',
                 bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.suptitle('Stock Volatility vs Wikipedia Views Change', fontsize=14,␣
 ↪fontweight='bold')
plt.tight_layout()
plt.show()
```



[230]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import gaussian_kde

# Density contour plots: Views Change vs Volatility (Absolute Returns)
fig, axes = plt.subplots(2, 5, figsize=(18, 8))
axes = axes.flatten()

for i, ticker in enumerate(stocks.keys()):
    x = df[('Views_Change', ticker)].dropna()
```

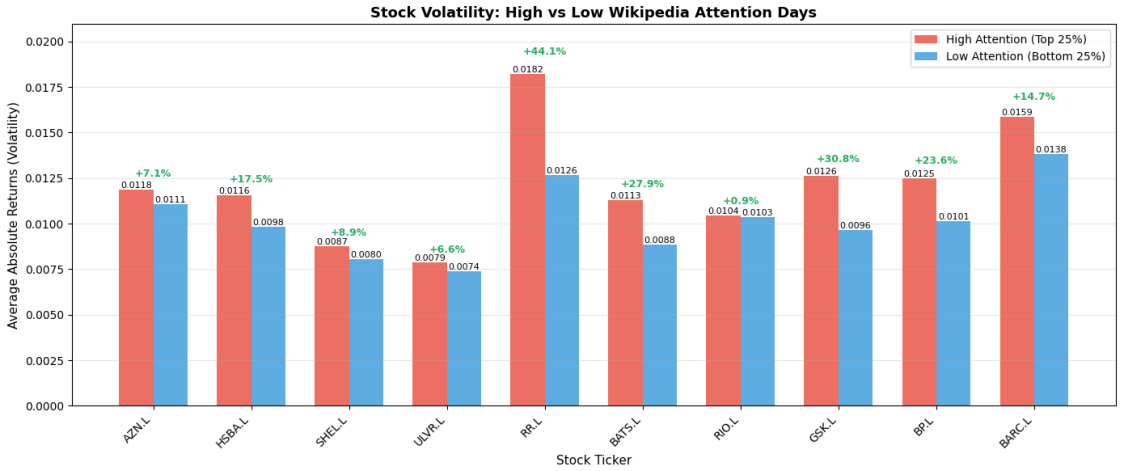
```
    y = df[('Abs_Log_Returns', ticker)].dropna()

    # Align the data (remove rows where either x or y is NaN)
    valid_idx = x.index.intersection(y.index)
    x = x.loc[valid_idx]
    y = y.loc[valid_idx]

    # Calculate kernel density estimate for contours
    # Stack the data
    xy = np.vstack([x, y])

    # Calculate kernel density
    kde = gaussian_kde(xy)

    # Create grid for contour plot
    x_min, x_max = x.min(), x.max()
    y_min, y_max = y.min(), y.max()

    # Add some padding
    x_range = x_max - x_min
    y_range = y_max - y_min
    x_min -= 0.2 * x_range
    x_max += 0.2 * x_range
    y_min -= 0.2 * y_range
    y_max += 0.2 * y_range

    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 200),
        np.linspace(y_min, y_max, 200)
    )

    # Evaluate KDE on grid
    positions = np.vstack([xx.ravel(), yy.ravel()])
    zz = kde(positions).reshape(xx.shape)

    contourf = axes[i].contourf(xx, yy, zz, levels=15, cmap='viridis', alpha=0.
↪9)

    axes[i].set_xlabel('Wikipedia Views Change', fontsize=9)
    axes[i].set_ylabel('Absolute Returns (Volatility)', fontsize=9)
    axes[i].set_title(ticker, fontsize=11, fontweight='bold')
    axes[i].axvline(x=0, color='red', linestyle='--', linewidth=1, alpha=0.7)
    axes[i].axhline(y=y.median(), color='white', linestyle='--', linewidth=0.8,␣
↪alpha=0.5)

    # Add correlation coefficient
    corr = x.corr(y)
```

```
    axes[i].text(0.05, 0.95, f'  = {corr:.3f}',
                 transform=axes[i].transAxes,
                 verticalalignment='top',
                 bbox=dict(boxstyle='round', facecolor='white', alpha=0.8),
                 fontsize=9)

plt.suptitle('Density Distribution: Stock Volatility vs Wikipedia Views Change',
             fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



[231]:
```
import matplotlib.pyplot as plt
import numpy as np

# Calculate high and low attention volatility for each stock
high_vol = []
low_vol = []
ticker_names = []

for ticker in stocks.keys():
    high_attention = df[df[('Views_Change', ticker)] > df[('Views_Change',␣
 ↪ticker)].quantile(0.75)]
    low_attention = df[df[('Views_Change', ticker)] < df[('Views_Change',␣
 ↪ticker)].quantile(0.25)]

    high_vol.append(high_attention[('Abs_Log_Returns', ticker)].mean())
    low_vol.append(low_attention[('Abs_Log_Returns', ticker)].mean())
    ticker_names.append(ticker)

# Calculate percentage differences
```

```
pct_diffs = [((high_vol[i] - low_vol[i]) / low_vol[i]) * 100 for i in␣
 ↪range(len(ticker_names))]

# Create bar plot
x = np.arange(len(ticker_names))
width = 0.35

fig, ax = plt.subplots(figsize=(14, 6))
bars1 = ax.bar(x - width/2, high_vol, width, label='High Attention (Top 25%)',␣
 ↪alpha=0.8, color='#e74c3c')
bars2 = ax.bar(x + width/2, low_vol, width, label='Low Attention (Bottom 25%)',␣
 ↪alpha=0.8, color='#3498db')

# Add value labels on bars
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.4f}',
                ha='center', va='bottom', fontsize=8)

# Add percentage difference labels above each pair of bars
for i, (pct, ticker) in enumerate(zip(pct_diffs, ticker_names)):
    max_height = max(high_vol[i], low_vol[i])
    color = '#27ae60' if pct > 0 else '#c0392b'
    ax.text(i, max_height * 1.05, f'{pct:+.1f}%',
            ha='center', va='bottom', fontsize=9, fontweight='bold',␣
 ↪color=color)

ax.set_xlabel('Stock Ticker', fontsize=11)
ax.set_ylabel('Average Absolute Returns (Volatility)', fontsize=11)
ax.set_title('Stock Volatility: High vs Low Wikipedia Attention Days',␣
 ↪fontsize=13, fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(ticker_names, rotation=45, ha='right')
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

# Add some padding at the top for percentage labels
ax.set_ylim(top=max(max(high_vol), max(low_vol)) * 1.15)

plt.tight_layout()
plt.show()
```



The graphics show that we can expect some observable correlation for some stocks.

1.3 Task 3: Hypothesis Test

1.3.1 Significance level

$$\alpha = 0.05.$$

1.3.2 Hypotheses

We perform a two-sided hypothesis test to assess whether there is a linear association between the two one-dimensional datasets:

$$H_0 : \ \rho = 0$$

$$H_1 : \ \rho \neq 0.$$

1.3.3 Test statistic (Lecture 10: t-test for slope in simple linear regression)

Equivalently, in the simple linear regression model $Y = \beta_0 + \beta_1 X + \varepsilon$, testing for no linear association corresponds to testing whether the slope is zero:

$$H_0 : \ \beta_1 = 0$$

$$H_1 : \ \beta_1 \neq 0.$$

We use the t-test for the regression slope. The test statistic is

$$t = \frac{\hat{\beta}_1 - 0}{\mathrm{se}(\hat{\beta}_1)},$$

and under H_0 it follows a Student t-distribution with $n-2$ degrees of freedom:

$$t \sim t_{n-2}.$$

1.3.4 Critical region

For a two-sided test at level α, the critical value is $t_{1-\alpha/2,\,n-2}$, hence the rejection region is

$$|t| > t_{1-\alpha/2,\,n-2},$$

equivalently,

$$t \notin \left[-t_{1-\alpha/2,\,n-2},\; t_{1-\alpha/2,\,n-2}\right].$$

1.3.5 p-value

Given an observed value t_{obs}, the two-sided p-value is

$$p = 2\left(1 - F_{t_{n-2}}(|t_{\mathrm{obs}}|)\right),$$

where $F_{t_{n-2}}$ denotes the CDF of the Student t-distribution with $n-2$ degrees of freedom.

[232]:
```
from scipy import stats
import numpy as np

def perform_hypothesis_test(x, y, alpha=0.05):
    """
    Perform t-test for regression slope to test correlation.

    H0:  1 = 0 (no linear association)
    H1:  1   0 (linear association exists)

    Parameters
    ----------
    x : array-like
        Independent variable (Views_Change)
    y : array-like
        Dependent variable (Abs_Log_Returns)
    alpha : float
        Significance level (default 0.05)

    Returns
    -------
```

```
    dict
        Dictionary containing test results
    """
    # Remove NaN values
    valid_idx = ~(np.isnan(x) | np.isnan(y))
    x_clean = x[valid_idx]
    y_clean = y[valid_idx]

    n = len(x_clean)

    # Calculate regression slope and intercept
    x_mean = np.mean(x_clean)
    y_mean = np.mean(y_clean)

    #  1 = Σ[(xi - x̄)(yi - ȳ)] / Σ[(xi - x̄)²]
    numerator = np.sum((x_clean - x_mean) * (y_clean - y_mean))
    denominator = np.sum((x_clean - x_mean) ** 2)
    beta_1 = numerator / denominator

    #  0 = ȳ -  1 * x̄
    beta_0 = y_mean - beta_1 * x_mean

    # Calculate fitted values and residuals
    y_fitted = beta_0 + beta_1 * x_clean
    residuals = y_clean - y_fitted

    # Calculate residual standard error
    SSE = np.sum(residuals ** 2)
    sigma_squared = SSE / (n - 2)

    # Calculate standard error of  1
    se_beta_1 = np.sqrt(sigma_squared / denominator)

    # Calculate t-statistic
    t_stat = beta_1 / se_beta_1

    # Degrees of freedom
    dof = n - 2

    # Calculate p-value (two-sided)
    p_value = 2 * (1 - stats.t.cdf(np.abs(t_stat), dof))

    # Critical value
    t_critical = stats.t.ppf(1 - alpha/2, dof)

    # Decision
    reject_null = np.abs(t_stat) > t_critical
```

```
    # Calculate correlation coefficient
    correlation = np.corrcoef(x_clean, y_clean)[0, 1]

    return {
        'n': n,
        'beta_0': beta_0,
        'beta_1': beta_1,
        'se_beta_1': se_beta_1,
        't_statistic': t_stat,
        'dof': dof,
        'p_value': p_value,
        't_critical': t_critical,
        'reject_null': reject_null,
        'correlation': correlation,
        'alpha': alpha
    }


# Perform hypothesis test for each stock
results = {}

alpha = 0.05
for ticker in stocks.keys():
    x = df[('Views_Change', ticker)].values
    y = df[('Abs_Log_Returns', ticker)].values

    result = perform_hypothesis_test(x, y, alpha)
    results[ticker] = result
```

[233]:
```
# Create a summary table of results
import pandas as pd

summary_data = []
for ticker in stocks.keys():
    res = results[ticker]
    summary_data.append({
        'Stock': ticker,
        'Company': stocks[ticker].replace('_', ' '),
        'n': res['n'],
        'Correlation  ': res['correlation'],
        'Slope  ': res['beta_1'],
        't-stat': res['t_statistic'],
        'p-value': res['p_value'],
        'Reject H ': 'yes' if res['reject_null'] else 'no'
    })
```

```
summary_df = pd.DataFrame(summary_data)

tickers_list = list(stocks.keys())
significant_count = sum(1 for t in tickers_list if results[t]['reject_null'])

print("\nSUMMARY TABLE")
print("=" * 100)
display(summary_df)
print(f"For {significant_count} out of {len(tickers_list)} stocks H  has to be␣
 ↪rejected at a significance level of   = {alpha}.")
print("=" * 100)
```

```
SUMMARY TABLE
====================================================================================
====================
```

	Stock	Company	n	Correlation	Slope	t-stat	p-value	Reject H
0	AZN.L	AstraZeneca	505	0.004366	0.000144	0.097921	9.220340e-01	no
1	HSBA.L	HSBC	505	0.171366	0.011450	3.901053	1.087901e-04	yes
2	SHEL.L	Shell plc	505	0.035519	0.002148	0.797120	4.257573e-01	no
3	ULVR.L	Unilever	505	0.074742	0.001841	1.680981	9.338747e-02	no
4	RR.L	Rolls-Royce Holdings	505	0.173035	0.016152	3.940205	9.292439e-05	yes
5	BATS.L	British American Tobacco	505	0.293625	0.021899	6.888973	1.685363e-11	yes
6	RIO.L	Rio Tinto (corporation)	505	-0.006177	-0.000255	-0.138540	8.898691e-01	no
7	GSK.L	GSK plc	505	0.023184	0.000377	0.520097	6.032250e-01	no
8	BP.L	BP	505	0.016185	0.000968	0.363047	7.167225e-01	no
9	BARC.L	Barclays	505	0.059305	0.007066	1.332424	1.833242e-01	no

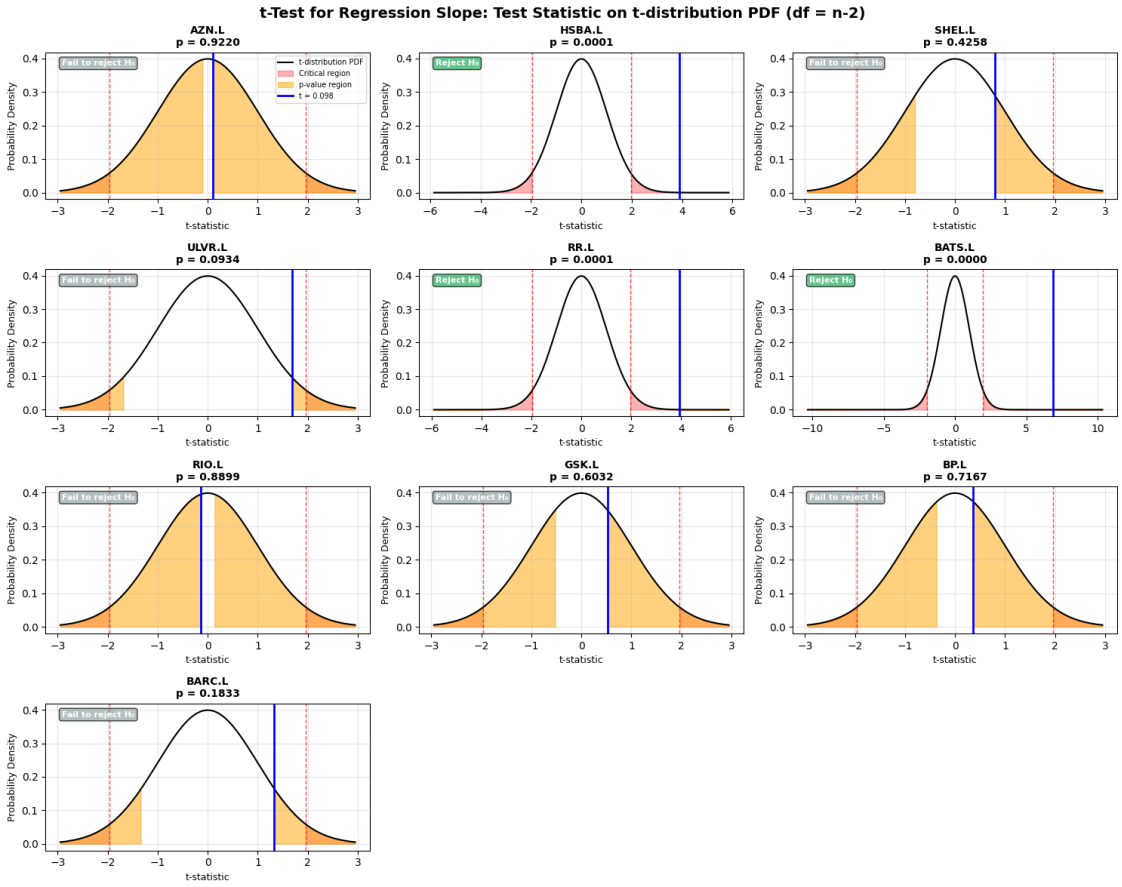
```
For 3 out of 10 stocks H  has to be rejected at a significance level of   =
0.05.
====================================================================================
====================
```

[234]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
from scipy import stats

fig, axes = plt.subplots(4, 3, figsize=(15, 12))
axes = axes.flatten()

for i, ticker in enumerate(stocks.keys()):
    res = results[ticker]

    # Get test statistics
    t_obs = res['t_statistic']
    t_crit = res['t_critical']
    dof = res['dof']
    p_val = res['p_value']

    # Create x-axis values for the t-distribution
    x_max = max(abs(t_obs), t_crit) * 1.5
    x = np.linspace(-x_max, x_max, 1000)

    # Calculate the t-distribution PDF
    y = stats.t.pdf(x, dof)

    # Plot the PDF
    axes[i].plot(x, y, 'k-', linewidth=1.5, label='t-distribution PDF')

    # Shade critical regions (rejection regions)
    x_left_crit = x[x <= -t_crit]
    y_left_crit = stats.t.pdf(x_left_crit, dof)
    axes[i].fill_between(x_left_crit, y_left_crit, alpha=0.3, color='red',␣
↪label='Critical region')

    x_right_crit = x[x >= t_crit]
    y_right_crit = stats.t.pdf(x_right_crit, dof)
    axes[i].fill_between(x_right_crit, y_right_crit, alpha=0.3, color='red')

    # Shade p-value regions
    if t_obs > 0:
        x_p_right = x[x >= abs(t_obs)]
        y_p_right = stats.t.pdf(x_p_right, dof)
        axes[i].fill_between(x_p_right, y_p_right, alpha=0.5, color='orange',␣
↪label='p-value region')

        x_p_left = x[x <= -abs(t_obs)]
        y_p_left = stats.t.pdf(x_p_left, dof)
        axes[i].fill_between(x_p_left, y_p_left, alpha=0.5, color='orange')
    else:
        x_p_left = x[x <= -abs(t_obs)]
        y_p_left = stats.t.pdf(x_p_left, dof)
```

```
        axes[i].fill_between(x_p_left, y_p_left, alpha=0.5, color='orange',␣
↪label='p-value region')

        x_p_right = x[x >= abs(t_obs)]
        y_p_right = stats.t.pdf(x_p_right, dof)
        axes[i].fill_between(x_p_right, y_p_right, alpha=0.5, color='orange')

    # Mark the observed t-statistic
    axes[i].axvline(x=t_obs, color='blue', linestyle='-', linewidth=2,␣
↪label=f't = {t_obs:.3f}')

    # Mark critical values
    axes[i].axvline(x=-t_crit, color='red', linestyle='--', linewidth=1,␣
↪alpha=0.7)
    axes[i].axvline(x=t_crit, color='red', linestyle='--', linewidth=1, alpha=0.
↪7)

    # Labels and title
    axes[i].set_xlabel('t-statistic', fontsize=9)
    axes[i].set_ylabel('Probability Density', fontsize=9)
    axes[i].set_title(f'{ticker}\np = {p_val:.4f}', fontsize=10,␣
↪fontweight='bold')
    axes[i].grid(True, alpha=0.3)

    # Add legend only to first subplot
    if i == 0:
        axes[i].legend(fontsize=7, loc='upper right')

    # Add text box with decision
    decision_text = 'Reject H ' if res['reject_null'] else 'Fail to reject H '
    decision_color = '#27ae60' if res['reject_null'] else '#95a5a6'
    axes[i].text(0.05, 0.95, decision_text,
                 transform=axes[i].transAxes,
                 verticalalignment='top',
                 bbox=dict(boxstyle='round', facecolor=decision_color, alpha=0.
↪7),
                 fontsize=8, color='white', fontweight='bold')

# Hide unused subplots
for j in range(len(stocks), len(axes)):
    axes[j].set_visible(False)

plt.suptitle(f't-Test for Regression Slope: Test Statistic on t-distribution␣
↪PDF (df = n-2)',
             fontsize=14, fontweight='bold')
plt.tight_layout()
```

```
plt.show()
```



We can see that for some stocks, the null hypothesis must be rejected. This means there's no evidence that they're not linearly dependent.

For the others we fail to reject H_0.

1.4 Task 4: Linear Regression

Having found in the previous step that we can't reject the null hypothesis for all stocks, and observing no obvious non-linear correlation in the scatterplots, we'll proceed with option a) and perform a linear regression on these few stocks where we rejected H_0.

[235]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

# Get stocks with significant linear relationship
```

```
significant_stocks = [ticker for ticker in stocks.keys() if␣
 ↪results[ticker]['reject_null']]

print(f"Performing linear regression for {len(significant_stocks)} stocks with␣
 ↪significant correlation:")
print(f"{', '.join(significant_stocks)}\n")

# Store regression results
regression_results = {}
regression_summary = []

for ticker in significant_stocks:
    # Get data
    x = df[('Views_Change', ticker)].to_numpy(dtype=float)
    y = df[('Abs_Log_Returns', ticker)].to_numpy(dtype=float)

    # Remove NaN values
    valid_idx = np.isfinite(x) & np.isfinite(y)
    x_clean = x[valid_idx]
    y_clean = y[valid_idx]

    # Perform linear regression
    slope, intercept, r_value, p_value, std_err = stats.linregress(x_clean,␣
 ↪y_clean)

    # Calculate fitted values and residuals
    y_fitted = intercept + slope * x_clean
    residuals = y_clean - y_fitted

    n = len(x_clean)
    RSS = np.sum(residuals**2)
    residuals_std = np.sqrt(RSS / (n - 2))

    # Store results
    regression_results[ticker] = {
        'x': x_clean,
        'y': y_clean,
        'slope': slope,
        'intercept': intercept,
        'r_value': r_value,
        'p_value': p_value,
        'std_err': std_err,
        'y_fitted': y_fitted,
        'residuals': residuals,
        'residuals_std': residuals_std
    }
```

```
    # Add to summary table
    regression_summary.append({
        'Stock': ticker,
        'n': n,
        'Intercept ( )': intercept,
        'Slope ( )': slope,
        'R²': r_value**2,
        'Residuals Std (s)': residuals_std,
        'Regression Equation': f'y = {intercept:.4f} + {slope:.4f}x'
    })

# Display as table
regression_summary_df = pd.DataFrame(regression_summary)
display(regression_summary_df)
```

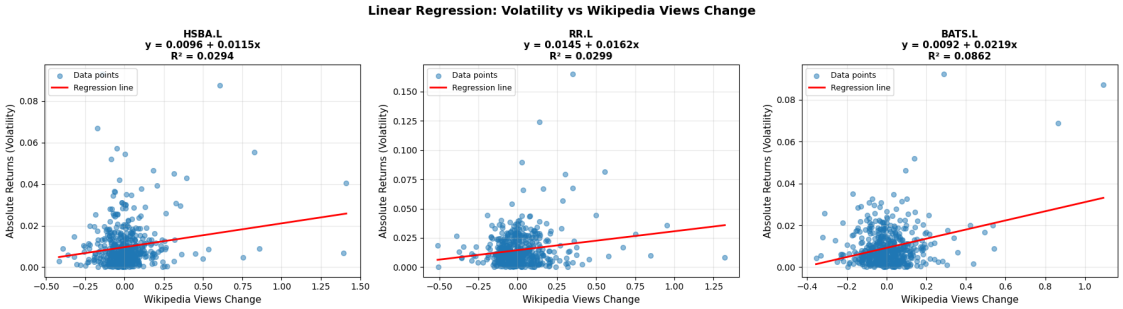
```
Performing linear regression for 3 stocks with significant correlation:
HSBA.L, RR.L, BATS.L
```

	Stock	n	Intercept ( )	Slope ( )	R^2	Residuals Std (s)	\
0	HSBA.L	505	0.009636	0.011450	0.029366	0.010596	
1	RR.L	505	0.014513	0.016152	0.029941	0.014719	
2	BATS.L	505	0.009202	0.021899	0.086215	0.009068	

	Regression Equation
0	y = 0.0096 + 0.0115x
1	y = 0.0145 + 0.0162x
2	y = 0.0092 + 0.0219x

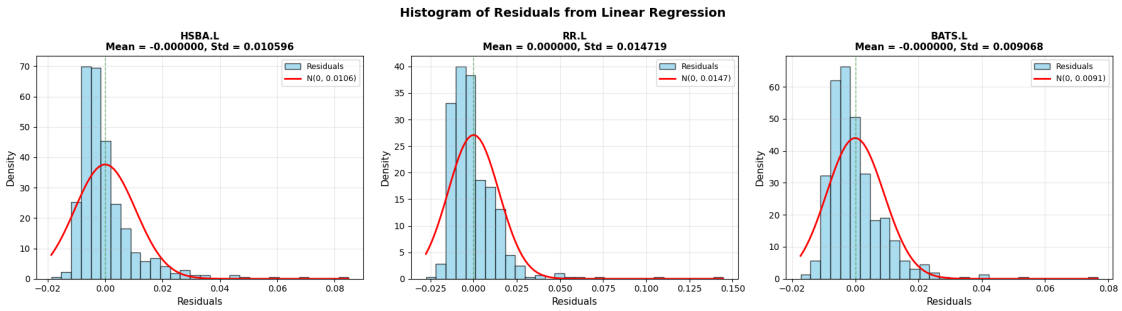
[236]:
```
# Plot regression results: scatter plots with regression lines
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

for i, ticker in enumerate(significant_stocks):
    res = regression_results[ticker]

    # Scatter plot
    axes[i].scatter(res['x'], res['y'], alpha=0.5, s=30, label='Data points')

    # Regression line
    x_line = np.array([res['x'].min(), res['x'].max()])
    y_line = res['intercept'] + res['slope'] * x_line
    axes[i].plot(x_line, y_line, 'r-', linewidth=2, label='Regression line')

    # Labels and title
    axes[i].set_xlabel('Wikipedia Views Change', fontsize=11)
    axes[i].set_ylabel('Absolute Returns (Volatility)', fontsize=11)
    axes[i].set_title(f'{ticker}\ny = {res["intercept"]:.4f} + {res["slope"]:.
 ↪4f}x\nR² = {res["r_value"]**2:.4f}',
```

```
                      fontsize=11, fontweight='bold')
    axes[i].grid(True, alpha=0.3)
    axes[i].legend(loc='upper left', fontsize=9)

plt.suptitle('Linear Regression: Volatility vs Wikipedia Views Change',␣
 ↪fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



[237]:
```
# Plot histograms of residuals
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

for i, ticker in enumerate(significant_stocks):
    res = regression_results[ticker]
    residuals = res['residuals']

    # Histogram
    n, bins, patches = axes[i].hist(residuals, bins=30, density=True, alpha=0.7,
                                    color='skyblue', edgecolor='black',␣
 ↪label='Residuals')

    # Overlay normal distribution
    mu = 0
    sigma = res['residuals_std']
    x_norm = np.linspace(residuals.min(), residuals.max(), 100)
    y_norm = stats.norm.pdf(x_norm, mu, sigma)
    axes[i].plot(x_norm, y_norm, 'r-', linewidth=2, label=f'N(0, {sigma:.4f})')

    # Labels and title
    axes[i].set_xlabel('Residuals', fontsize=11)
    axes[i].set_ylabel('Density', fontsize=11)
    axes[i].set_title(f'{ticker}\nMean = {np.mean(residuals):.6f}, Std = {sigma:
 ↪.6f}',
                      fontsize=11, fontweight='bold')
    axes[i].grid(True, alpha=0.3)
```

```
    axes[i].legend(loc='upper right', fontsize=9)
    axes[i].axvline(x=0, color='green', linestyle='--', linewidth=1, alpha=0.5)

plt.suptitle('Histogram of Residuals from Linear Regression', fontsize=14,␣
 ↪fontweight='bold')
plt.tight_layout()
plt.show()
```



Now we perform the Kolmogorov-Smirnov Test for Normality of Residuals with

$$H_0 : \text{Residuals come from N}(0,\, s^2) \text{ distribution}$$

$$H_1 : \text{Residuals do not come from N}(0,\, s^2) \text{ distribution}$$

at a significance level of $\alpha = 0.05$.

[238]:
```
# Kolmogorov-Smirnov test for normality of residuals
from scipy.stats import kstest

ks_results = {}
ks_summary = []

for ticker in significant_stocks:
    res = regression_results[ticker]
    residuals = res['residuals']
    sigma = res['residuals_std']

    # Standardise residuals
    standardized_residuals = residuals / sigma

    # Perform KS test against standard normal distribution N(0,1)
    # Then we test if residuals/s ~ N(0,1), equivalent to residuals ~ N(0, s²)
    ks_statistic, p_value = kstest(standardized_residuals, 'norm', args=(0, 1))
```

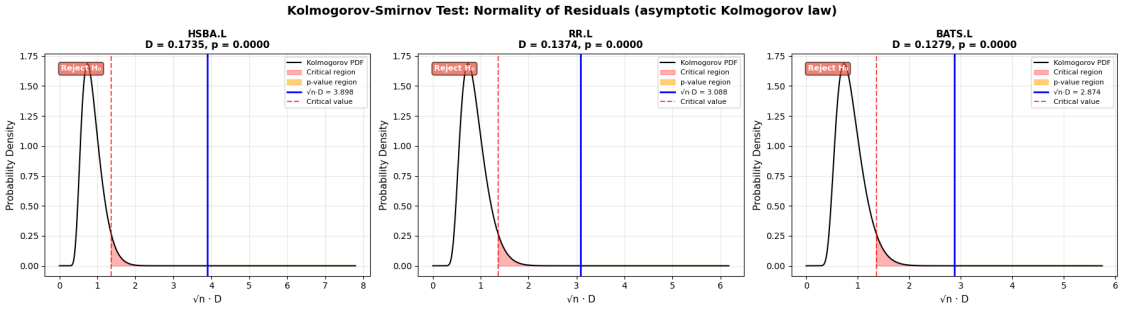
```
    # Critical value for KS test at   = 0.05
    n = len(residuals)
    # Kolmogorov-Smirnov critical value approximation
    critical_value = 1.36 / np.sqrt(n)

    reject_null = ks_statistic > critical_value

    ks_results[ticker] = {
        'ks_statistic': ks_statistic,
        'p_value': p_value,
        'critical_value': critical_value,
        'reject_null': reject_null,
        'n': n
    }

    # Add to summary table
    decision = 'Reject H ' if reject_null else 'Fail to reject H '
    conclusion = 'not Normal' if reject_null else 'Consistent with Normal'

    ks_summary.append({
        'Stock': ticker,
        'n': n,
        'KS Statistic (D)': ks_statistic,
        'Critical Value': critical_value,
        'p-value': p_value,
        'Decision': decision,
        'Conclusion': conclusion
    })

# Display as table
ks_summary_df = pd.DataFrame(ks_summary)
display(ks_summary_df)
```

```
    Stock    n  KS Statistic (D)  Critical Value       p-value   Decision  \
0  HSBA.L  505          0.173450        0.060519  9.336780e-14  Reject H
1    RR.L  505          0.137401        0.060519  8.882314e-09  Reject H
2  BATS.L  505          0.127907        0.060519  1.160031e-07  Reject H

   Conclusion
0  not Normal
1  not Normal
2  not Normal
```

[239]:
```
# Plot KS test statistic on Kolmogorov distribution
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import kstwobign
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

for i, ticker in enumerate(significant_stocks):
    ks_res = ks_results[ticker]

    D_obs = ks_res['ks_statistic']
    D_crit = ks_res['critical_value']
    n = ks_res['n']

    # --- Use asymptotic Kolmogorov distribution of X = sqrt(n) * D ---
    x_obs = np.sqrt(n) * D_obs
    x_crit = np.sqrt(n) * D_crit

    # --- p-value consistent with the Kolmogorov PDF you plot ---
    # (right-tail area beyond observed statistic)
    p_val = kstwobign.sf(x_obs)

    # Create x grid
    x_max = max(x_obs, x_crit) * 2
    x_max = max(x_max, 3.0)  # ensure a reasonable plot range
    x = np.linspace(0, x_max, 2000)

    # PDF for kstwobign is available directly
    pdf_vals = kstwobign.pdf(x)

    # Plot the PDF
    axes[i].plot(x, pdf_vals, 'k-', linewidth=1.5, label='Kolmogorov PDF')

    # Shade critical region: X >= x_crit
    mask_crit = x >= x_crit
    axes[i].fill_between(x[mask_crit], pdf_vals[mask_crit], alpha=0.3,␣
↪color='red', label='Critical region')

    # Shade p-value region: X >= x_obs
    mask_p = x >= x_obs
    axes[i].fill_between(x[mask_p], pdf_vals[mask_p], alpha=0.5,␣
↪color='orange', label='p-value region')

    # Mark observed and critical values
    axes[i].axvline(x=x_obs, color='blue', linestyle='-', linewidth=2,␣
↪label=f'√n·D = {x_obs:.3f}')
    axes[i].axvline(x=x_crit, color='red', linestyle='--', linewidth=1.5,␣
↪alpha=0.7, label='Critical value')

    # Decision
    reject_null = x_obs > x_crit
```

```
    decision_text = 'Reject H ' if reject_null else 'Fail to reject H '
    decision_color = '#e74c3c' if reject_null else '#27ae60'

    axes[i].set_xlabel('√n · D', fontsize=11)
    axes[i].set_ylabel('Probability Density', fontsize=11)
    axes[i].set_title(f'{ticker}\nD = {D_obs:.4f}, p = {p_val:.4f}',␣
 ↪fontsize=11, fontweight='bold')
    axes[i].grid(True, alpha=0.3)
    axes[i].legend(loc='upper right', fontsize=8)

    axes[i].text(
        0.05, 0.95, decision_text,
        transform=axes[i].transAxes,
        verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor=decision_color, alpha=0.7),
        fontsize=9, color='white', fontweight='bold'
    )

plt.suptitle('Kolmogorov-Smirnov Test: Normality of Residuals (asymptotic␣
 ↪Kolmogorov law)', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



1.4.1 Conclusions from Task 4a

For the three stocks that showed significant linear correlation in Task 3 (HSBC, Rolls-Royce, and British American Tobacco), we performed linear regression analysis and tested the normality of residuals.

Linear Regression Results:

- **HSBC (HSBA.L)**: $y = 0.0096 + 0.0115x$, $R^2 = 0.029$
- **Rolls-Royce (RR.L)**: $y = 0.0145 + 0.0162x$, $R^2 = 0.030$
- **British American Tobacco (BATS.L)**: $y = 0.0092 + 0.0219x$, $R^2 = 0.086$

All three stocks show positive slopes, confirming that increased Wikipedia attention is associated with higher stock volatility. However, the R^2 values are quite low ($2.9\% - 8.6\%$), indicating that

Wikipedia views change explains only a small portion of volatility variance or the other way around.

Kolmogorov-Smirnov Test for Normality:

All three stocks **reject the null hypothesis** at $\alpha = 0.05$, with very small p-values ($p < 0.0001$):

- **HSBC**: $D = 0.173$, $p < 0.0001$
- **Rolls-Royce**: $D = 0.137$, $p < 0.0001$
- **British American Tobacco**: $D = 0.128$, $p < 0.0001$

Interpretation:

The residuals from the linear regression do **not** follow a normal distribution with mean zero and standard deviation s. This suggests that:

1. The linear model may not fully capture the relationship between Wikipedia attention and stock volatility
2. The distribution of residuals shows significant departures from normality, visible in the histograms as positive skewness and heavy tails
3. The normality assumption required for standard linear regression inference is violated

This finding suggests that while there is a statistically significant linear association, the relationship may be more complex or contain outliers that violate the classical linear regression assumptions.

1.5 Task 5:

Independence means the joint distribution factorises as a product of its marginals, as there is no "information flow" between the two. But marginals alone are not enough to decide independence because very different joint distributions can share the same marginals. So, to test independence we would compare the empirical joint distribution to the product of the empirical marginals.

For this we perform a Pearson's χ^2 goodness-of-fit test:

1. Split X and Y into bins.
2. Build the contingency table O_{ij} that represents the observed counts in bin (i, j).
3. Under H_0 (independence) the expected counts are $E_{ij} = n\hat{p}_i\hat{q}_j$ where $\hat{p}_i$ and $\hat{q}_j$ are the empirical marginals.
4. Use the Pearson statistic with $T = \sum_{ij} \frac{(O_{ij}-E_{ij})^2}{E_{ij}}$
5. Compute a p-value from a χ^2 distribution and reject if $p < \alpha = 0.05$

[240]:
```
import pandas as pd
from scipy.stats import chi2_contingency, chi2

# Significance level
alpha = 0.05
# Number of bins
kx = ky = 20

independence_summary = []
for i, ticker in enumerate(stocks.keys()):
```

```
    # Get series
    x = df[('Views_Change', ticker)]
    y = df[('Abs_Log_Returns', ticker)]

    # Drop NaNs + align
    x = x.dropna()
    y = y.dropna()
    valid_idx = x.index.intersection(y.index)
    x = x.loc[valid_idx]
    y = y.loc[valid_idx]

    # Create bins
    x_bin = pd.qcut(x, q=kx, duplicates="drop")
    y_bin = pd.qcut(y, q=ky, duplicates="drop")
    # Create crosstab
    obs = pd.crosstab(x_bin, y_bin)

    # Chi-Square test
    chi2_stat, p_value, dof, expected = chi2_contingency(obs.values,␣
 ↪correction=False)

    # Critical value (right-tailed)
    critical_value = chi2.ppf(1 - alpha, dof)

    reject_null = p_value < alpha

    # Add to summary table
    decision = 'Reject H ' if reject_null else 'Fail to reject H '
    conclusion = 'Dependent' if reject_null else 'Insufficient evidence of␣
 ↪dependence'

    independence_summary.append({
        'Stock': ticker,
        'n': len(x),
        'dof': dof,
        'Chi2': chi2_stat,
        'Critical Value': critical_value,
        'p-value': p_value,
        'Decision': decision,
        'Conclusion': conclusion
    })

# Display as table
independence_summary_df = pd.DataFrame(independence_summary)
display(independence_summary_df)
```

	Stock	n	dof	Chi2	Critical Value	p-value	Decision	\
0	AZN.L	505	361	355.916828	406.304801	0.565618	Fail to reject H	
1	HSBA.L	505	361	349.075124	406.304801	0.664180	Fail to reject H	
2	SHEL.L	505	361	350.854333	406.304801	0.639194	Fail to reject H	
3	ULVR.L	505	361	364.318355	406.304801	0.441205	Fail to reject H	
4	RR.L	505	361	369.435290	406.304801	0.368329	Fail to reject H	
5	BATS.L	505	361	336.475822	406.304801	0.818395	Fail to reject H	
6	RIO.L	505	361	363.811562	406.304801	0.448614	Fail to reject H	
7	GSK.L	505	361	402.008686	406.304801	0.067257	Fail to reject H	
8	BP.L	505	361	390.617799	406.304801	0.136074	Fail to reject H	
9	BARC.L	505	361	360.768414	406.304801	0.493539	Fail to reject H	

	Conclusion
0	Insufficient evidence of dependence
1	Insufficient evidence of dependence
2	Insufficient evidence of dependence
3	Insufficient evidence of dependence
4	Insufficient evidence of dependence
5	Insufficient evidence of dependence
6	Insufficient evidence of dependence
7	Insufficient evidence of dependence
8	Insufficient evidence of dependence
9	Insufficient evidence of dependence

We fail to reject independence at $\alpha = 0.05$. The data are consistent with independence under this χ^2 binning approach, but this does not prove independence.