Project 1

Man United stock

Step 1 — Raw data

```
import yfinance as yf

manu = yf.download("MANU", start="2026-01-01")
print(manu.head())
```

Price	Close	High	Low	Open	Volume
Ticker	MANU	MANU	MANU	MANU	MANU
Date					
2026-01-02	15.780000	16.129999	15.71	15.94	213200
2026-01-05	16.000000	16.020000	15.68	15.75	350400
2026-01-06	15.970000	16.070000	15.78	16.00	295700
2026-01-07	16.340000	16.389999	15.92	15.96	351200
2026-01-08	16.389999	16.490000	16.15	16.34	213500

Close: The final price the stock traded at when the market closed
Importance: This is the most important price. Most analysis uses closing price

Open: This is the price in which the stock is first traded when the market opens for the day.
Importance: Shows where investor sentiment started the day
Example: If man uniteds stock yesterday closed at $15 but then opens today at $16 this shows overnight the demand was strong overnight

High: The highest price in which the stock reached on the day
Importance: This shows the maximum buying pressure for the day

Low: The lowest price in which the stock reached on the day
Importance: This shows the maximum selling pressure for the day

Volume: The total number of shares traded that day
Importance: High volume = strong investor interest
Low volume = weak conviction

Step 2 — Calculate Daily Returns

```
import yfinance as yf
import pandas as pd

#This is the data for the Manchester united Stock and the sp500 from
2026-01-01
manu = yf.download("MANU", start="2026-01-01")
sp500 = yf.download("^GSPC", start="2026-01-01")

#calculates the daily returns
manu['Returns'] = manu['Close'].pct_change()
sp500['Returns'] = sp500['Close'].pct_change()

#prints the data

print(manu[['Close', 'Returns']].head())
print(sp500[['Close', 'Returns']].head())
```

Price	Close	Returns
Ticker	MANU	
Date		
2026-01-02	15.780000	NaN
2026-01-05	16.000000	0.013942
2026-01-06	15.970000	-0.001875
2026-01-07	16.340000	0.023168
2026-01-08	16.389999	0.003060

Price	Close	Returns
Ticker	^GSPC	
Date		
2026-01-02	6858.470215	NaN
2026-01-05	6902.049805	0.006354
2026-01-06	6944.819824	0.006197
2026-01-07	6920.930176	-0.003440
2026-01-08	6921.459961	0.000077

'Returns': Calculates the day-to-day percentage change for each stock.
Importance: Returns standardize stock price changes so we can compare performance across days or against the market. Most financial analysis is done on returns rather than raw prices.

Daily return:

$$Return_t = \frac{Price_t - Price_{t-1}}{Price_{t-1}}$$

Where:

- $Price_t$ = today's closing price
- $Price_{t-1}$ = yesterday's closing price

Example: Now if the Manchester united stock goes from 15.78000 to 16.000000 the return would be 0.013942 %
If S&P 500 goes from 6858.470215 → 6902.049805 the return would be 0.006354%

Why this is useful:

- Allows comparison with the market (S&P 500) to calculate abnormal returns.
- Standardizes data for statistical analysis.
- Essential for understanding how the stock reacted relative to normal market movement.

Step 1–2 Mini Quiz: MANU Stock Project

Part A — Conceptual Questions

1. Why do we use `pandas` (`pd`) when analyzing stock data in Python?

2. What is the difference between `Close` and `Adj Close`?

3. Why do we calculate returns instead of using raw closing prices?

4. What does the `NaN` in the first row of `pct_change()` indicate?

5. Why do we also download S&P 500 data along with MANU stock?

Part B — Python / DataFrame Questions

6. What type of object is returned by `yf.download("MANU")`?

7. How would you check all the column names in a DataFrame called `manu`?

8. Suppose `manu` DataFrame exists. Write Python code to calculate daily returns for the column `'Close'` safely, in case `'Adj Close'` does not exist.

9. If the stock goes from 100 → 105 → 102, what would the daily returns be? (Show decimals)

10. How would you remove rows with `NaN` in the `'Returns'` column?

Model answers

1.**Def:** A Python library for working with structured data (tables) called DataFrames.
Importance: Allows us to store, manipulate, and analyze stock data efficiently. Most data analysis in Python uses pandas.

2.`Close`: The final price the stock traded at when the market closed.

- `Adj Close`: Closing price adjusted for dividends, splits, or corporate actions.

Importance: Adjusted Close reflects the true investor return over time.

What it does: Adjusts historical prices for corporate events to make analysis accurate.

How it works: Yahoo Finance calculates it automatically.

Example:

- If a stock closes at $100 but pays a $2 dividend, Adj Close = $98 (approx.).

Why we add it: Use Adj Close when calculating returns to avoid misleading results due to splits/dividends.

3.**Def:** Returns show **percentage change** from one day to the next.

Importance: Standardizes price movements and allows comparison across time and against the market.

What it does: Converts raw prices into daily percent changes.

How it works: `(Price_today - Price_yesterday) / Price_yesterday`

Example:

- Price goes 100 → 102 → return = 0.02 = 2%

Why we add it: Analysts use returns to detect abnormal movements, trends, or compare with the market.

4.**Def:** `NaN` = Not a Number, meaning no value exists.

Importance: Shows that no calculation can be done without a previous day's data.

What it does: Marks missing data in a DataFrame.

How it works: `.pct_change()` needs a previous row; first row has none → NaN.

Example:

Close	Returns
100	NaN
102	0.02

Why we add it: We remove NaN before regression or plotting to avoid errors.

5.**Def:** S&P 500 is a market benchmark.

Importance: Stocks move with the market; we need a benchmark to see abnormal performance.

What it does: Provides daily returns of the overall market.

How it works: Download S&P 500 prices and calculate daily returns like MANU.

Example:

```
sp500 = yf.download("^GSPC", start="2021-01-01")

sp500['Returns'] = sp500['Adj Close'].pct_change()
```

Why we add it: Comparing MANU to the market isolates **event-specific effects**.

6.**Def:** A pandas DataFrame — a 2D table with rows and columns.

Importance: Lets us manipulate stock data programmatically.

What it does: Stores Open, High, Low, Close, Adj Close, Volume, and Dates.

How it works: Each row = one trading day; each column = a stock feature.

Example:

```
type(manu)

# <class 'pandas.core.frame.DataFrame'>
```

Why we add it: DataFrames are the standard structure for time series and financial analysis.

7.**Def:** Columns are the names of each feature in the table.

Importance: Knowing column names prevents errors when referencing them.

What it does: Shows all available features like Close, Adj Close, Volume.

How it works:

```
print(manu.columns)
```

Example Output:

```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

Why we add it: Ensures you reference the correct column before calculations.

8.**Def:** Adds a column `'Returns'` with daily percentage change.

Importance: Standardizes stock changes for analysis.

What it does: Checks if `Adj Close` exists, otherwise uses `Close`.

How it works:

```
if 'Adj Close' in manu.columns:

    price_col = 'Adj Close'

else:

    price_col = 'Close'
```

```
manu['Returns'] = manu[price_col].pct_change()

print(manu[[price_col, 'Returns']].head())
```

Example:

- Price 100 → 102 → return = 0.02

Why we add it: Prevents KeyErrors and ensures calculations use the correct column.

9. If the stock goes from 100 → 105 → 102, what are the daily returns?

Def: Percentage change for each day.

Importance: Shows daily stock performance.

What it does: `(Price_today - Price_yesterday)/Price_yesterday`

How it works:

- Day 1: 100 → 105 → `(105-100)/100 = 0.05` → 5%
- Day 2: 105 → 102 → `(102-105)/105 = -0.02857` → -2.857%

Example Table:

Day	Price	Return
1	100	NaN
2	105	0.05
3	102	-0.02857

Why we add it: Helps detect trends, volatility, and abnormal events.

10.Removes incomplete data before analysis.

Importance: NaN values can break regression or plotting.

What it does: Drops rows where the Returns column is missing.

How it works:

```
manu = manu.dropna(subset=['Returns'])
```

Example:

Close	Returns
100	NaN
102	0.02

Why we add it: Ensures calculations like regression, CAR, and plots run without errors.

Next Step: Step 3 — Market Model (Alpha & Beta)

Concepts

1. Alpha (α)

Def: The part of the stock return **not explained by the market** — the stock's "extra" return.

Importance: Shows if MANU performed better or worse than expected **even if the market was flat**.

What it does: It's like a baseline adjustment — gives the average abnormal return.

How it works: Calculated via regression; α is the intercept.

Example:

- If S&P 500 return = 0% and MANU return = 1%, then α ≈ 1% for that period.

Why we add it: Needed to measure abnormal returns for your event study.Perfect. I'll give you a clean **study-ready version** with:

1. The Beta interpretation table
2. Proper Def / Importance / What it does / How it works / Example / Why we add it
3. Written clearly so you can paste into your notes

📊 Beta Interpretation Table

Beta (β)	Meaning	What It Tells You
2.0	Moves twice the market	Very volatile compared to market
1.0	Moves same as market	Average market sensitivity
0.5	Moves half as much	More stable than market
0	No relationship	Market doesn't affect it
-1	Moves opposite market	Inverse relationship

Beta (β)

Def:

Beta measures how sensitive a stock's returns are to movements in the overall market.

Importance:

It tells us how much of the stock's movement is explained by general market changes.

What it does:

Scales market returns to estimate how much the stock should normally move.

How it works:

It is the **slope** of the regression line between:

- Market returns (X-axis)
- Stock returns (Y-axis)

Mathematically:

Expected Return = Alpha + Beta × Market Return

Beta is the multiplier on Market Return.

Example:

If:

- Beta = 1.5
- Market return = 2%

Expected stock return (ignoring alpha):

= 1.5 × 2%
= 3%

Market Model (Full Formula)

$$R_{Stock} = \alpha + \beta R_{Market} + \varepsilon$$

Def:

A linear regression model that explains stock returns using market returns.

Importance:

Provides the baseline “normal” expected return.

What it does:

Predicts what the stock *should* return given market movement.

How it works:

- Alpha = intercept
- Beta = slope
- Epsilon (ε) = unexplained randomness

Example:

If:

- Alpha = 0.001
- Beta = 1.2
- Market return = 3%

Expected stock return:

= 0.001 + (1.2 × 0.03)
= 0.001 + 0.036
= 0.037 (3.7%)

Why we add it:

This allows us to calculate:

Abnormal Return = Actual − Expected

Which is the core of event study analysis.

```
import yfinance as yf

import pandas as pd

import statsmodels.api as sm  # for regression
```

```python
# Step 1: Download data

manu = yf.download("MANU", start="2026-01-01")

sp500 = yf.download("^GSPC", start="2026-01-01")



# Step 2: Calculate daily returns

manu['Returns'] = manu['Close'].pct_change()

sp500['Returns'] = sp500['Close'].pct_change()



# Step 2: Quick check

print(manu[['Close', 'Returns']].head())

print(sp500[['Close', 'Returns']].head())



# Step 3: Alpha & Beta (Regression)

window = 5  # example for testing, use more days for real analysis



# Take last N rows

y = manu['Returns'].iloc[-window:]

X = sp500['Returns'].iloc[-window:]



# Align indices so regression works

y, X = y.align(X, join='inner')



# Add constant for alpha

X = sm.add_constant(X)
```

```
# Run regression

model = sm.OLS(y, X).fit()

alpha, beta = model.params

print(f"\nAlpha: {alpha}, Beta: {beta}")



# Step 3: Expected and abnormal returns

manu['Expected'] = alpha + beta * sp500['Returns']

manu['Abnormal'] = manu['Returns'] - manu['Expected']



# Step 3: View results

print(manu[['Returns', 'Expected', 'Abnormal']].tail())
```

Output

Daily Returns

Date	MANU Close	MANU Returns	S&P 500 Close	S&P 500 Returns
2026-01-02	15.78	NaN	6858.47	NaN
2026-01-05	16.00	0.013942	6902.05	0.006354
2026-01-06	15.97	-0.001875	6944.82	0.006197
2026-01-07	16.34	0.023168	6920.93	-0.003440
2026-01-08	16.39	0.003060	6921.46	0.000077

Regression Output

Metric	Value
Alpha	0.002313 (≈ 0.23%)
Beta	1.261

Explanation:

- **Alpha (α)**: Extra return MANU is independent of the market (~0.23% per day in this window).
- **Beta (β)**: Sensitivity to market movements. Beta > 1 → MANU moves **more strongly than the S&P 500**.

Example: If S&P goes up 1%, MANU is expected to go up 1.26%.

Expected & Abnormal Returns

Date	MANU Returns	Expected Return	Abnormal Return
2026-02-09	0.010957	0.008228	0.002729
2026-02-10	0.014261	-0.001853	0.016114
2026-02-11	-0.014061	0.002251	-0.016312
2026-02-12	-0.019966	-0.017435	-0.002531
2026-02-13	0.022701	NaN	NaN

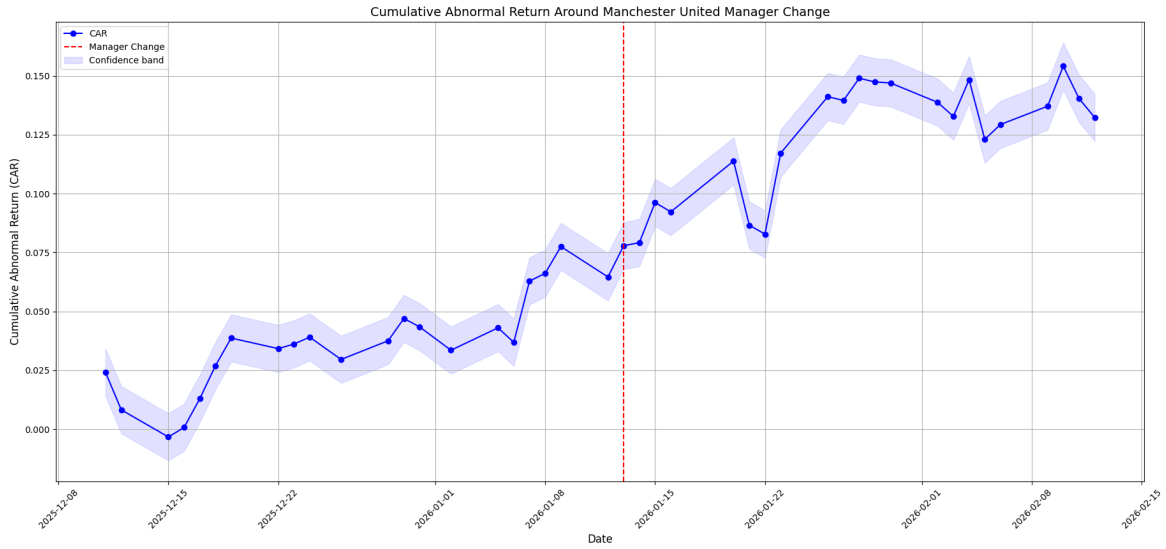
Explanation:

(Step 4): Event Window Analysis

```
import yfinance as yf
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt  # for plotting

# ----------------------------
# Step 1: Download data
# ----------------------------
manu = yf.download("MANU", start="2025-01-01")    # start earlier for
12-month estimation window
sp500 = yf.download("^GSPC", start="2025-01-01")

# ----------------------------
```

```
# Step 2: Calculate daily returns
# ------------------------------
manu['Returns'] = manu['Close'].pct_change()
sp500['Returns'] = sp500['Close'].pct_change()

print(manu[['Close', 'Returns']].head())
print(sp500[['Close', 'Returns']].head())

# ------------------------------
# Step 3: Alpha & Beta (Estimation Window)
# ------------------------------
event_date = pd.to_datetime("2026-01-13")

# Make sure index is datetime
manu.index = pd.to_datetime(manu.index)
sp500.index = pd.to_datetime(sp500.index)

# 252 trading days (~12 months) BEFORE the event
estimation_window = manu.loc[:event_date].iloc[-253:-1]

y = estimation_window['Returns']
X = sp500.loc[estimation_window.index]['Returns']

# Align dates and drop NaNs
y, X = y.align(X, join='inner')
data = pd.concat([y, X], axis=1).dropna()
y = data.iloc[:, 0]
X = data.iloc[:, 1]

# Add constant for alpha
X = sm.add_constant(X)

# Run regression
model = sm.OLS(y, X).fit()
alpha, beta = model.params
print(f"\nAlpha: {alpha}, Beta: {beta}")

# ------------------------------
# Step 4: Expected & Abnormal Returns (Full Sample)
# ------------------------------
manu['Expected'] = alpha + beta * sp500['Returns']
manu['Abnormal'] = manu['Returns'] - manu['Expected']
```

```
# ----------------------------
# Step 5: Event Window (1 Month Before & After)
# ----------------------------
event_index = manu.index.get_loc(event_date)

# 21 trading days before and after (~1 month)
event_window = manu.iloc[event_index-21 : event_index+22].copy()

print("\nEvent Window:")
print(event_window[['Returns', 'Expected', 'Abnormal']])

# ----------------------------
# Step 6: Cumulative Abnormal Return (CAR)
# ----------------------------
event_window['CAR'] = event_window['Abnormal'].cumsum()

print("\nEvent Window with CAR:")
print(event_window[['Abnormal', 'CAR']])

# ----------------------------
# Step 7: Professional CAR Plot
# ----------------------------
plt.figure(figsize=(12,6))
plt.plot(event_window.index, event_window['CAR'], marker='o',
color='blue', label='CAR')
plt.axvline(event_date, linestyle='--', color='red', label='Manager
Change')

# Optional: add a confidence band (+/-1% as example)
plt.fill_between(event_window.index,
                 event_window['CAR'] - 0.01,
                 event_window['CAR'] + 0.01,
                 color='blue', alpha=0.1, label='Confidence band')

plt.title("Cumulative Abnormal Return Around Manchester United Manager
Change", fontsize=14)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Cumulative Abnormal Return (CAR)", fontsize=12)
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



State the event

> On 13 January 2026, Carrick was appointed manager of Manchester United.

State the method

> Using the market model (CAPM-style regression), expected returns were calculated based on S&P 500 returns. Abnormal returns were then derived and accumulated into CAR.

State the result

- CAR increased after the event → positive market reaction

Economic meaning

This is the part that separates beginners from serious analysts.

Ask:

- Did investors believe Carrick improves future performance?
- Was the appointment already priced in?
- Was it neutral news?

Highlight Significant Days & Market Overlay

```
import yfinance as yf

import pandas as pd

import statsmodels.api as sm

import matplotlib.pyplot as plt

import os

import numpy as np



# ----------------------------

# Step 1: Download and Align Data

# ----------------------------

manu = yf.download("MANU", start="2025-01-01")

sp500 = yf.download("^GSPC", start="2025-01-01")



# Keep only Close prices

manu = manu[['Close']].rename(columns={'Close': 'MANU_Close'})

sp500 = sp500[['Close']].rename(columns={'Close': 'SP500_Close'})



# Merge to ensure perfect date alignment

data = manu.merge(sp500, left_index=True, right_index=True,
how='inner')



# ----------------------------

# Step 2: Calculate Returns

# ----------------------------
```

```
data['MANU_Returns'] = data['MANU_Close'].pct_change()

data['SP500_Returns'] = data['SP500_Close'].pct_change()



data.dropna(inplace=True)



# ----------------------------

# Step 3: Alpha & Beta (252-day Estimation Window)

# ----------------------------

event_date = pd.to_datetime("2026-01-13")



estimation_window = data.loc[:event_date].iloc[-253:-1]



y = estimation_window['MANU_Returns']

X = estimation_window['SP500_Returns']



X = sm.add_constant(X)

model = sm.OLS(y, X).fit()



alpha = model.params['const']

beta = model.params['SP500_Returns']



print(f"\nAlpha: {alpha:.6f}")

print(f"Beta:  {beta:.6f}")



# ----------------------------
```

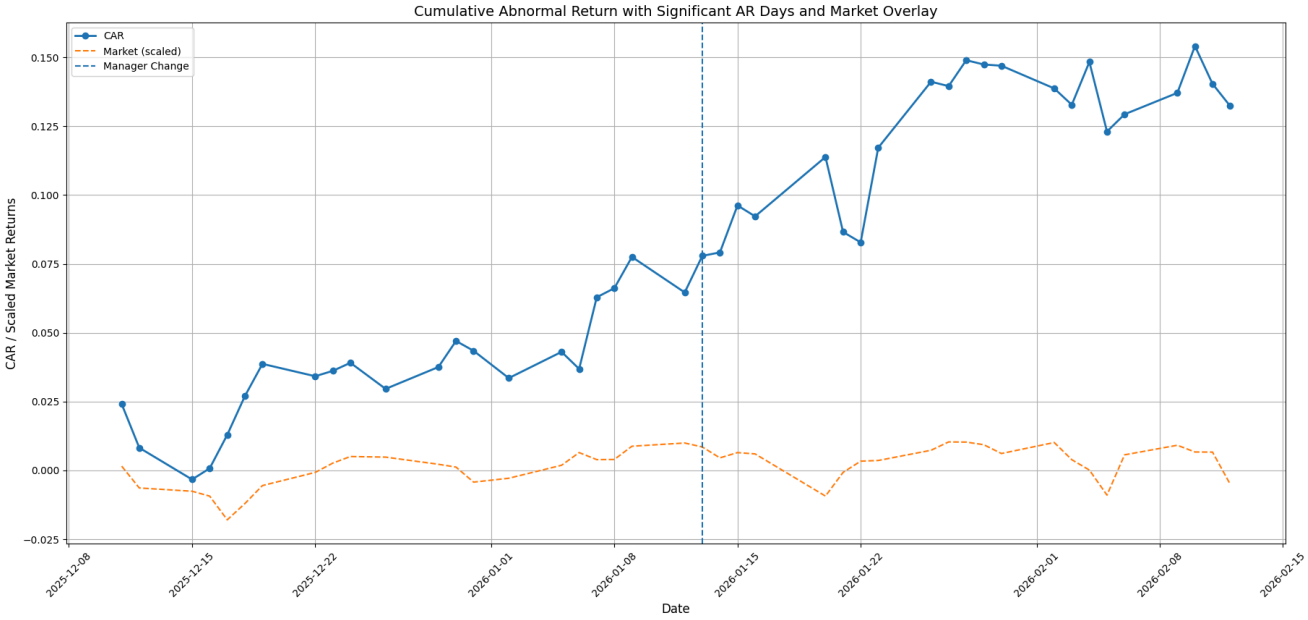
```
# Step 4: Expected & Abnormal Returns
# ----------------------------
data['Expected'] = alpha + beta * data['SP500_Returns']
data['Abnormal'] = data['MANU_Returns'] - data['Expected']


# ----------------------------
# Step 5: Event Window (±21 Days)
# ----------------------------
if event_date not in data.index:
    raise ValueError("Event date not in trading calendar.")


event_index = data.index.get_loc(event_date)
event_window = data.iloc[event_index-21:event_index+22].copy()


# ----------------------------
# Step 6: CAR
# ----------------------------
event_window['CAR'] = event_window['Abnormal'].cumsum()


# ----------------------------
# Step 7: Significance Testing
# ----------------------------
est_std = estimation_window['MANU_Returns'].std()


event_window['t_AR'] = event_window['Abnormal'] / est_std
```

```python
N = len(event_window)
CAR_std = est_std * np.sqrt(N)
t_CAR = event_window['CAR'].iloc[-1] / CAR_std

print(f"Cumulative Abnormal Return t-statistic: {t_CAR:.3f}")

# Identify significant days
sig_pos = event_window[event_window['t_AR'] > 2]
sig_neg = event_window[event_window['t_AR'] < -2]

# ----------------------------
# Step 8: Professional CAR Plot
# ----------------------------
output_folder = "manu_event_analysis"
os.makedirs(output_folder, exist_ok=True)

plt.figure(figsize=(12,6))

# CAR line
plt.plot(event_window.index,
         event_window['CAR'],
         marker='o',
         linewidth=2,
         label='CAR')
```

```
# Significant AR markers (plotted at CAR level)
if not sig_pos.empty:
    plt.scatter(sig_pos.index,
                sig_pos['CAR'],
                s=120,
                zorder=5,
                label='Significant Positive AR')

if not sig_neg.empty:
    plt.scatter(sig_neg.index,
                sig_neg['CAR'],
                s=120,
                zorder=5,
                label='Significant Negative AR')

# Market overlay (scaled cumulative)
market_overlay = event_window['SP500_Returns'].cumsum() * beta
plt.plot(event_window.index,
         market_overlay,
         linestyle='--',
         label='Market (scaled)')

# Event line
plt.axvline(event_date,
```

```
            linestyle='--',
            linewidth=1.5,
            label='Manager Change')

# Formatting
plt.title("Cumulative Abnormal Return with Significant AR Days and Market Overlay", fontsize=14)
plt.xlabel("Date", fontsize=12)
plt.ylabel("CAR / Scaled Market Returns", fontsize=12)
plt.grid(True)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()

# Save plot
plot_file = os.path.join(output_folder, "CAR_plot.png")
plt.savefig(plot_file)
plt.show()

print(f"Professional CAR plot saved to {plot_file}")

# ----------------------------
# Step 9: Export Event Window Table
# ----------------------------
table_file = os.path.join(output_folder, "event_window.csv")
```

```
event_window.to_csv(table_file)



print(f"Event window table saved to {table_file}")
```



Professional CAR with Significance & Market Overlay:
The second graph improves on the first by adding **statistical and comparative context**, making it suitable for professional presentation. The **orange dashed line** represents the scaled cumulative returns of the S&P 500 over the same period, allowing comparison between Manchester United's stock and overall market movement. The vertical red dashed line again marks the event day. The plot features a grid, rotated x-axis labels, and a tight layout for clarity. The y-axis now represents **CAR alongside scaled market returns**, giving an immediate visual impression of how abnormal returns relate to the broader market trend. This enhanced plot communicates both the **magnitude and significance** of stock movements.