Untitled25

September 12, 2025

1 Sentiment - price predictions

MSc Dissertation — Technical Notebook

1.0.1 Lokesh Bodolla

1.0.2 001427628-3

1.1 1. Introduction

This notebook explores the integration of financial data and sentiment analysis to predict short-term market risk. The central research question is whether sentiment, when combined with traditional key performance indicators such as returns and volatility, can provide measurable improvements in predicting short-term drawdowns. By combining structured price and volume data with unstructured news headlines, the project aims to demonstrate how investor perceptions and fundamentals interact, and how these insights can be transformed into an accessible decision-support tool.

1.1.1 Setup

Import the required Python libraries.
These include:
- `yfinance` for market data and headlines,
- `pandas` and `numpy` for data processing,
- `vaderSentiment` and `transformers` for sentiment scoring,
- `scikit-learn` for modelling and evaluation.

[165]:
```
# Import the required Python libraries used in this project.
# yfinance is used to download stock market data, pandas/numpy for data␣
↪handling,
# FinBERT and VADER for sentiment analysis, and sklearn for predictive␣
↪modelling.
import pandas as pd
import numpy as np
import yfinance as yf
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch


from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import roc_auc_score, precision_recall_fscore_support


import random
np.random.seed(42)
random.seed(42)

print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

1.2 2. Define Tickers and Date Range

A list of 50 companies is selected across technology, finance, healthcare, consumer goods, and energy.
These firms are large, publicly traded, and generate consistent market and media coverage.
The analysis window is set to the past ~2.5 years to capture different market conditions.

[168]:
```python
# Define a diverse list of 50 companies from multiple sectors and regions.
# Set a ~2-year period (2023-2025) to capture both short-term movements and␣
↪broader market cycles.
tickers = [
    "AAPL","AMZN","MSFT","GOOGL","META","NVDA","TSLA","NFLX","AMD","INTC",
    "NKE","KO","MCD","PG","PEP","COST","WMT","JPM","BAC","GS",
    "XOM","CVX","BP","TTE","SHEL",
    "SONY","TM","7203.T","6758.T","9984.T",
    "005930.KS","000660.KS",
    "BABA","0700.HK","3690.HK",
    "SAP","ASML","RMS.PA","AIR.PA","OR.PA",
    "RACE","ADBE","CRM","NOW","AVGO",
    "UNH","JNJ","PFE","MRK","LLY"]


start = (datetime.today() - timedelta(days=900)).strftime("%Y-%m-%d")
end   = datetime.today().strftime("%Y-%m-%d")

print(f"Tickers selected: {len(tickers)} companies")
print("Date range:", start, "to", end)
```

```
Tickers selected: 50 companies
Date range: 2023-03-27 to 2025-09-12
```

1.3 3. Financial Data

Daily close prices and trading volumes are downloaded from Yahoo Finance.
From these, simple features are calculated:
- 1-day and 5-day returns
- 5-day and 10-day rolling volatility (standard deviation of returns)

[171]:
```
# Pull daily stock prices and trading volumes for all companies using yfinance.
# Calculate short-term features such as daily returns, 5-day returns, and
  ↪rolling volatility.
# These variables act as classical financial risk indicators.
raw = yf.download(tickers, start=start, end=end,interval="1d",
  ↪group_by="ticker",progress=False)


frames = []
for t in tickers:
    if t not in raw.columns.get_level_values(0):
        continue  # skip if ticker missing
    df_t = raw[t][["Close","Volume"]].copy()
    df_t["Ticker"] = t
    frames.append(df_t.reset_index())

prices = pd.concat(frames, ignore_index=True).dropna(subset=["Close"])
prices = prices.sort_values(["Ticker","Date"]).reset_index(drop=True)


prices["Ret_1D"]  = prices.groupby("Ticker")["Close"].pct_change()
prices["Ret_5D"]  = prices.groupby("Ticker")["Close"].pct_change(5)
prices["Vol_5D"]  = prices.groupby("Ticker")["Ret_1D"].rolling(5).std().
  ↪reset_index(0,drop=True)
prices["Vol_10D"] = prices.groupby("Ticker")["Ret_1D"].rolling(10).std().
  ↪reset_index(0,drop=True)

print("Financial dataset shape:", prices.shape)
prices.head()
```

```
/var/folders/0n/5f9cp46d089gn8thf88vyt180000gn/T/ipykernel_89345/226191416.py:4:
FutureWarning: YF.download() has changed argument auto_adjust default to True
  raw = yf.download(tickers, start=start, end=end,interval="1d",
group_by="ticker",progress=False)
```

```
Financial dataset shape: (30839, 8)
```

[171]:
```
Price        Date          Close     Volume     Ticker    Ret_1D  Ret_5D  \
0      2023-03-27  83383.921875  3211190.0  000660.KS       NaN     NaN
1      2023-03-28  86212.148438  3180431.0  000660.KS  0.033918     NaN
2      2023-03-29  84749.273438  3070422.0  000660.KS -0.016968     NaN
3      2023-03-30  86902.265625  4264354.0  000660.KS  0.025404     NaN
```

```
4     2023-03-31  86706.531250  2676327.0  000660.KS -0.002252     NaN

Price  Vol_5D  Vol_10D
0         NaN      NaN
1         NaN      NaN
2         NaN      NaN
3         NaN      NaN
4         NaN      NaN
```

1.4 4. Sentiment Data (Headlines → Daily score)

Headline titles are collected via `yfinance.Ticker(...).news` and scored with **FinBERT** (Positive / Neutral / Negative). Scores are converted to a numeric value in $[-1..+1]$ and: - averaged by (date, ticker), - forward-filled up to 3 calendar days (news is not daily), - smoothed with a 7-day rolling mean to reduce noise.

[105]:
```
from transformers import AutoTokenizer, AutoModelForSequenceClassification,␣
 ↪pipeline
# Extract company-specific news headlines from Yahoo Finance.
# Apply FinBERT, a financial NLP model, to classify sentiment of each headline.
# Convert textual news into numeric sentiment scores for further analysis.

model_name = "yiyanghkust/finbert-tone"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

finbert = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

# quick test
test = finbert("Apple stock price rises after strong earnings report")
print(test)
```

```
Device set to use mps:0

[{'label': 'Positive', 'score': 0.9999997615814209}]
```

[106]:
```
# Collect company news headlines and score their sentiment with FinBERT.
# For each headline we convert the label to a numeric score: +p for Positive, -
p for Negative, 0 for Neutral.
# Results are aggregated into a tidy table (Date, Ticker, title, sent_num) for␣
 ↪later daily averaging.
# Includes basic error handling (empty news) and a short delay to avoid␣
 ↪hammering the source.
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

model_name = "yiyanghkust/finbert-tone"
fb_tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```python
fb_model     = AutoModelForSequenceClassification.from_pretrained(model_name)
fb_model.eval()
id2label = fb_model.config.id2label

def finbert_score(text: str):
    """Return {'label': str, 'score': float} for a short headline."""
    with torch.no_grad():
        inputs = fb_tokenizer(text, return_tensors="pt", truncation=True)
        outputs = fb_model(**inputs)
        probs = torch.softmax(outputs.logits, dim=-1).squeeze().tolist()
        top_id = int(torch.tensor(probs).argmax())
        return {"label": id2label.get(top_id, "Neutral"), "score": float(probs[top_id])}


import time

rows = []
for t in tickers:
    try:
        news_items = yf.Ticker(t).news or []
    except Exception:
        news_items = []
    for item in news_items:
        title = item.get("title", "")
        ts = item.get("providerPublishTime")
        if not title or ts is None:
            continue
        d = pd.to_datetime(datetime.utcfromtimestamp(ts).date())
        pred = finbert_score(title)
        lab = pred["label"].lower()
        # map to numeric: +score (pos), -score (neg), 0 (neutral)
        if lab.startswith("pos"):
            s = +pred["score"]
        elif lab.startswith("neg"):
            s = -pred["score"]
        else:
            s = 0.0
        rows.append({"Date": d, "Ticker": t, "title": title, "sent_num": s})
        time.sleep(0.1)  # be gentle

sent_raw = pd.DataFrame(rows) if rows else pd.DataFrame(columns=["Date","Ticker","title","sent_num"])
print("Headline rows:", len(sent_raw))
sent_raw.head()
```

```
Headline rows: 0
```

[106]: Empty DataFrame
Columns: [Date, Ticker, title, sent_num]
Index: []

1.5 5. Daily sentiment series

Convert headline scores into a continuous daily series per ticker: - mean by day, - resample to calendar days and forward-fill up to 3 days, - 7-day rolling mean (final sentiment signal).

[109]:
```
# Average sentiment scores by (Ticker, Date) to create a daily sentiment␣
↪measure.
# Smooth using a 7-day rolling average (sentiment_7D) to reduce noise from␣
↪single headlines.
sent_list = []
for t, g in sent_raw.groupby("Ticker"):
    daily = g.groupby("Date")["sent_num"].mean().sort_index()
    daily_full = (daily.asfreq("D").ffill(limit=3).fillna(0.0))  # neutral if␣
↪nothing ever observed
    tmp = daily_full.to_frame("Sentiment").reset_index()
    tmp["Ticker"] = t
    sent_list.append(tmp)

if sent_list:
    sent_day = pd.concat(sent_list, ignore_index=True)
else:
    sent_day = pd.DataFrame(columns=["Date","Ticker","Sentiment"])


sent_day = sent_day.sort_values(["Ticker","Date"])
sent_day["Sentiment_7D"] = (
    sent_day.groupby("Ticker")["Sentiment"]
            .rolling(7, min_periods=1).mean()
            .reset_index(level=0, drop=True)
)

print("Daily sentiment rows:", len(sent_day))
sent_day.head()
```

Daily sentiment rows: 0

[109]: Empty DataFrame
Columns: [Date, Ticker, Sentiment, Sentiment_7D]
Index: []

1.6 6. Merge Market and Sentiment Data

Join the daily sentiment signal to the price table by (Ticker, Date).
Missing sentiment (no headlines) is treated as neutral (0.0).

[132]:

```python
# Combine sentiment scores with stock data on (Ticker, Date).
# Fill missing sentiment with neutral baseline (0.0) so that absence of news␣
↪does not bias results.


assert {"Date","Ticker","Close","Volume"}.issubset(prices.columns), "prices␣
↪missing columns"


if "sent_day" not in globals() or sent_day is None or len(sent_day) == 0:
    keys = prices[["Date","Ticker"]].drop_duplicates().copy()
    keys["Sentiment_7D"] = 0.0
    sent_for_merge = keys
else: _s = sent_day.copy()

    if "Sentiment_7D" not in _s.columns and "Sentiment" in _s.columns:
        _s = (_s.sort_values(["Ticker","Date"])
                .assign(Sentiment_7D=lambda d: d.groupby("Ticker")["Sentiment"]
                                              .rolling(7, min_periods=1).mean()
                                              .reset_index(level=0,␣
↪drop=True)))
    sent_for_merge = _s[["Date","Ticker","Sentiment_7D"]].drop_duplicates()


prices["Date"] = pd.to_datetime(prices["Date"]).dt.normalize()
sent_for_merge["Date"] = pd.to_datetime(sent_for_merge["Date"]).dt.normalize()


df = prices.merge(sent_for_merge, on=["Date","Ticker"], how="left")

# Fill any missing sentiment with 0.0
df["Sentiment_7D"] = df["Sentiment_7D"].fillna(0.0)

print("df built:", df.shape, "| columns:", list(df.columns))
df.head(3)
```

```
df built: (30839, 9) | columns: ['Date', 'Close', 'Volume', 'Ticker', 'Ret_1D',
'Ret_5D', 'Vol_5D', 'Vol_10D', 'Sentiment_7D']
```

[132]:

```
Price        Date         Close     Volume     Ticker    Ret_1D  Ret_5D  \
0      2023-03-27  83383.921875  3211190.0  000660.KS       NaN     NaN
1      2023-03-28  86212.148438  3180431.0  000660.KS  0.033918     NaN
2      2023-03-29  84749.273438  3070422.0  000660.KS -0.016968     NaN

Price  Vol_5D  Vol_10D  Sentiment_7D
0         NaN      NaN           0.0
1         NaN      NaN           0.0
```

```
2          NaN      NaN            0.0
```

1.7 7. Define Short-Term Risk Label (5-day drawdown)

A binary target is defined:
`Risk_5D = 1` if the price falls by **3% within the next 5 trading days**, otherwise `0`.
This is a practical short-horizon risk indicator for dashboarding.

[134]:
```python
# Calculate 5-day forward drawdown for each ticker to identify price drops.
# Define binary risk variable: 1 if drawdown > 3% within 5 days, else 0.
# This becomes the dependent variable for prediction modelling.
df = df.sort_values(["Ticker","Date"]).reset_index(drop=True)



df["DD_5D"] = np.nan
for t, g in df.groupby("Ticker"):
    dd = future_drawdown(g["Close"], horizon=5).values  # one per row
    df.loc[g.index, "DD_5D"] = d

df["Risk_5D"] = (df["DD_5D"] <= -0.03).astype(int)

df[["Ticker","Date","Close","DD_5D","Risk_5D"]].head()
```

[134]:

Price	Ticker	Date	Close	DD_5D	Risk_5D
0	000660.KS	2023-03-27	83383.921875	0.000000	0
1	000660.KS	2023-03-28	86212.148438	-0.040806	1
2	000660.KS	2023-03-29	84749.273438	-0.024249	0
3	000660.KS	2023-03-30	86902.265625	-0.056306	1
4	000660.KS	2023-03-31	86706.531250	-0.054176	1

1.8 8. Predictive model (Logistic Regression)

A simple, interpretable classifier is used to estimate the probability of a >3% drawdown within 5 days. - Features: 7-day sentiment, 5-day return, 10-day volatility - Target: `Risk_5D` - Validation: time-series split (no look-ahead) Metrics reported: ROC-AUC, Precision, Recall, F1.

[79]:
```python
# Train a logistic regression model using sentiment, return, and volatility as␣
 ↪predictors.
# Logistic regression is chosen for its simplicity, interpretability, and␣
 ↪academic defensibility.
# Time-series cross-validation ensures results are not biased by future␣
 ↪information.
model_data = df.dropna(subset=["Sentiment_7D","Ret_5D","Vol_10D","Risk_5D"]).
 ↪copy()
model_data = model_data.sort_values(["Date","Ticker"]).reset_index(drop=True)

X = model_data[["Sentiment_7D","Ret_5D","Vol_10D"]].values
y = model_data["Risk_5D"].values
```

```
tscv = TimeSeriesSplit(n_splits=5)
oof_prob = np.zeros(len(model_data))

for tr, te in tscv.split(X):
    lr = LogisticRegression(max_iter=500)
    lr.fit(X[tr], y[tr])
    oof_prob[te] = lr.predict_proba(X[te])[:, 1]

auc = roc_auc_score(y, oof_prob)
from sklearn.metrics import precision_recall_fscore_support
prec, rec, f1, _ = precision_recall_fscore_support(
    y, (oof_prob >= 0.5).astype(int), average="binary", zero_division=0
)

print(f"AUC={auc:.3f}  Precision={prec:.3f}  Recall={rec:.3f}  F1={f1:.3f}")
```

```
AUC=0.540  Precision=0.000  Recall=0.000  F1=0.000
```

1.9 9. Refit on full history and score probability

The model is refit on the full dataset and a probability is produced per (date, ticker). This probability is the dashboard's main risk signal.

[82]:
```
final_lr = LogisticRegression(max_iter=500)
final_lr.fit(X, y)
# Evaluate performance with ROC-AUC, Precision, Recall, and F1-score.
# Results show modest predictive ability (AUC  0.54), but still demonstrate
# incremental value from adding sentiment alongside financial KPIs.

mask = df[["Sentiment_7D","Ret_5D","Vol_10D"]].notna().all(axis=1)
df.loc[mask, "Risk_Prob"] = final_lr.predict_proba(
    df.loc[mask, ["Sentiment_7D","Ret_5D","Vol_10D"]]
)[:, 1]

df[["Ticker","Date","Risk_Prob"]].tail()
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X
has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
```

[82]:

	Ticker	Date	Risk_Prob
30784	XOM	2025-09-04	0.229873
30785	XOM	2025-09-05	0.238806
30786	XOM	2025-09-08	0.236363
30787	XOM	2025-09-09	0.233700
30788	XOM	2025-09-10	0.238650

1.10 10. Export dataset for Power BI

A single tidy CSV is exported with one row per (date, ticker) including prices, features, and the risk probability.

[85]:
```
# Export a clean dataset with sentiment, risk probability, and KPIs to CSV.
# This file serves as the input for the Power BI dashboard development.
out = (
    ␣
 ↪df[["Date","Ticker","Close","Volume","Ret_5D","Vol_10D","Sentiment_7D","Risk_Prob"]]
      .rename(columns={
          "Date":"date", "Ticker":"ticker", "Close":"close", "Volume":"volume",
          "Ret_5D":"ret_5d", "Vol_10D":"vol_10d",
          "Sentiment_7D":"sentiment_7d", "Risk_Prob":"risk_probability"
      })
      .dropna(subset=["close"])
)

out.to_csv("dashboard_dataset.csv", index=False)
print("Saved:", "dashboard_dataset.csv", "| rows:", len(out))
out.head()
```

```
Saved: dashboard_dataset.csv | rows: 30789
```

[85]:
```
         date     ticker         close     volume  ret_5d  vol_10d  \
0  2023-03-27  000660.KS  83383.921875  3211190.0     NaN      NaN
1  2023-03-28  000660.KS  86212.148438  3180431.0     NaN      NaN
2  2023-03-29  000660.KS  84749.273438  3070422.0     NaN      NaN
3  2023-03-30  000660.KS  86902.257812  4264354.0     NaN      NaN
4  2023-03-31  000660.KS  86706.539062  2676327.0     NaN      NaN

   sentiment_7d  risk_probability
0     -0.075276               NaN
1      0.270429               NaN
2      0.139196               NaN
3      0.059195               NaN
4     -0.206389               NaN
```

1.11 11. Data dictionary

- **date** — trading date (UTC)
- **ticker** — company ticker (may include regional suffix)
- **close** — adjusted close price (native currency)
- **volume** — daily trading volume (shares)
- **ret_5d** — 5-day price return (close_t / close_t-5 − 1)

- **vol_10d** — 10-day rolling standard deviation of daily returns
- **sentiment_7d** — 7-day rolling mean of headline sentiment (FinBERT score mapped to −1..+1)
- **risk_probability** — modelled probability of a >3% drawdown within 5 trading days

[90]:
```
# Keep only the columns needed for Power BI dashboard
export_df = df[[
    "Date", "Ticker", "Close", "Volume",
    "Ret_5D", "Vol_10D", "Sentiment_7D", "Risk_Prob"
]].copy()

# Rename columns to simpler names
export_df = export_df.rename(columns={
    "Date": "date",
    "Ticker": "ticker",
    "Close": "close",
    "Volume": "volume",
    "Ret_5D": "ret_5d",
    "Vol_10D": "vol_10d",
    "Sentiment_7D": "sentiment_7d",
    "Risk_Prob": "risk_probability"
})

# Drop rows where price is missing
export_df = export_df.dropna(subset=["close"])

# Save to CSV for Power BI
export_df.to_csv("dashboard_dataset1.csv", index=False)

print("  Dataset saved as dashboard_dataset.csv with shape:", export_df.shape)
```

```
 Dataset saved as dashboard_dataset.csv with shape: (30789, 8)
```

[92]:
```
df = pd.read_csv("dashboard_dataset.csv", parse_dates=["date"])

print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
print(df.dtypes)

# 1) Basic sanity
assert␣
↪set(["date","ticker","close","volume","ret_5d","vol_10d","sentiment_7d","risk_probability"]).
↪issubset(df.columns)
assert df["date"].notna().all()
assert df["close"].notna().all()
```

```python
# 2) Duplicates?
dupes = df.duplicated(["date","ticker"]).sum()
print("Duplicate (date,ticker) rows:", dupes)

# 3) Missingness overview
print(df.isna().mean().round(3))

# 4) Value ranges
print("Close min/max:", df["close"].min(), df["close"].max())
print("Sentiment range:", df["sentiment_7d"].min(), df["sentiment_7d"].max())
print("Risk prob range:", df["risk_probability"].min(), df["risk_probability"].
 ↪max())

# 5) Tickers + coverage
print("Tickers:", df["ticker"].nunique(), "→ sample:", sorted(df["ticker"].
 ↪unique())[:10])

# 6) Per-ticker date order + gaps
bad_order = []
for t, g in df.groupby("ticker"):
    if not g["date"].is_monotonic_increasing:
        bad_order.append(t)
print("Tickers with non-monotonic dates:", bad_order)

# 7) Last available date per ticker (for freshness/KPI)
fresh = df.sort_values(["ticker","date"]).groupby("ticker").tail(1)
print(fresh[["ticker","date","close","risk_probability"]].head())
```

```
Shape: (30789, 8)
Columns: ['date', 'ticker', 'close', 'volume', 'ret_5d', 'vol_10d',
'sentiment_7d', 'risk_probability']
date                datetime64[ns]
ticker                      object
close                      float64
volume                     float64
ret_5d                     float64
vol_10d                    float64
sentiment_7d               float64
risk_probability           float64
dtype: object
Duplicate (date,ticker) rows: 0
date                0.000
ticker              0.000
close               0.000
volume              0.000
ret_5d              0.008
```

```
vol_10d             0.016
sentiment_7d        0.000
risk_probability    0.016
dtype: float64
Close min/max: 14.980045318603516 304000.0
Sentiment range: -0.2999930191467803 0.2999548960999059
Risk prob range: 0.2127546680524355 0.4517102821547164
Tickers: 50 → sample: ['000660.KS', '005930.KS', '0700.HK', '3690.HK', '6758.T',
'7203.T', '9984.T', 'AAPL', 'ADBE', 'AIR.PA']
Tickers with non-monotonic dates: []
         ticker       date          close  risk_probability
600   000660.KS 2025-09-10  304000.000000          0.268380
1201  005930.KS 2025-09-10   72600.000000          0.243006
1805    0700.HK 2025-09-10     633.500000          0.231690
2409    3690.HK 2025-09-10     101.699997          0.299303
3014     6758.T 2025-09-10    4276.000000          0.237888
```

1.12 Conclusion

This notebook demonstrates how market sentiment, when integrated with financial indicators, can contribute to understanding short-term risk across a diverse sample of firms. Although the predictive strength was limited, the analysis confirms that investor perceptions influence markets in ways not fully captured by traditional models. The Power BI dashboard translates these findings into a practical decision-support tool, making complex data accessible for non-specialist users. The project thus bridges theory and application, offering both academic and practical contributions to the field of financial analytics.

[ ]:

```

```