```
# ============================================================
# Final Colab Script — Full Multi-Agent Investment Research Notebook
# Members: Richa Arun Kumar Jha; Raminder Singh; Samiksha Kodgire
# ============================================================

# ---------------------------
# 0) Installs (Colab friendly)
# ---------------------------
!pip install -q yfinance pandas numpy nltk transformers accelerate
torch sentencepiece requests psutil matplotlib seaborn tqdm textblob

# ---------------------------
# 1) Imports & Setup
# ---------------------------
import os, gc, json, time, math, re
from datetime import datetime, timezone
from IPython.display import display, HTML
import requests
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import torch, psutil
from transformers import pipeline
from tqdm.auto import tqdm
import nltk
from textblob import TextBlob

nltk.download("punkt", quiet=True)
nltk.download("stopwords", quiet=True)

plt.style.use("seaborn-v0_8-muted")
sns.set_style("whitegrid")

# Filenames
REPORT_CSV = "FinalProject_investment_final_report.csv"
RAW_JSON = "FinalProject_investment_all_results.json"

# ---------------------------
# 2) Credentials helper (optional)
# ---------------------------
def safe_get_env(varname, prompt_text):
    val = os.getenv(varname)
    if val and len(val) > 8:
        return val
    try:
        from google.colab import userdata
        val = userdata.get(varname)
        if val:
```

```
            os.environ[varname] = val
            return val
    except Exception:
        pass
    try:
        val = input(f"🔑 Enter your {prompt_text} (or press Enter to
skip): ").strip()
    except Exception:
        val = ""
    os.environ[varname] = val
    return val

HF_TOKEN = safe_get_env("HUGGINGFACEHUB_API_TOKEN", "Hugging Face
token")
NEWS_API_KEY = safe_get_env("NEWS_API_KEY", "News API key")
if HF_TOKEN:
    os.environ["HUGGINGFACEHUB_API_TOKEN"] = HF_TOKEN

print("HF token loaded:", "YES" if HF_TOKEN else "NO")
print("News API key loaded:", "YES" if NEWS_API_KEY else "NO")

# ----------------------------
# 3) Device detection & LLM selection (Phi-3 -> Phi-2 -> CPU fallback)
# ----------------------------
if torch.cuda.is_available():
    device_id = 0
    device = torch.device("cuda")
    props = torch.cuda.get_device_properties(device_id)
    total_vram_gb = props.total_memory / 1e9
    print(f"🚀 GPU detected: {torch.cuda.get_device_name(device_id)} —
VRAM: {total_vram_gb:.1f} GB")
else:
    device_id = -1
    device = torch.device("cpu")
    total_vram_gb = psutil.virtual_memory().available / 1e9 * 0.7
    total_vram_gb = min(total_vram_gb, 8)
    print(f"⚠️ CPU environment — usable RAM est: {total_vram_gb:.1f}
GB")

print("Torch cuda device count:", torch.cuda.device_count())

# Model selection strategy: try phi-3, then phi-2, else Tiny/CPU small
model
def choose_llm_preference():
    # User-specified chain: Phi-3 preferred, Phi-2 fallback, else CPU-
safe fallback
    candidates = ["microsoft/phi-3", "microsoft/phi-2",
"tinybird/TinyLlama-1.1B-Chat-v1.0"]
    for m in candidates:
        yield m
```

```
LLM_CANDIDATES = list(choose_llm_preference())
print("LLM preference list:", LLM_CANDIDATES)

def safe_make_pipeline(model_name, device_idx, use_fp16_flag):
    """Attempt to make a pipeline; return pipeline or raise."""
    kwargs = {
        "model": model_name,
        "device": device_idx,
        "max_new_tokens": 350,
        "temperature": 0.45,
        "repetition_penalty": 1.05,
        "trust_remote_code": True,
    }
    # prefer fp16 on GPU
    if use_fp16_flag and device_idx >= 0:
        kwargs["torch_dtype"] = torch.float16
    else:
        kwargs["torch_dtype"] = torch.float32
    return pipeline("text-generation", **kwargs)

# Try to create pipeline lazily; we'll use try/except inside
summarizer so notebook doesn't crash on failed loads
print("LLM pipeline will be attempted at summarization time (safe
fallbacks enabled).")

# ---------------------------
# 4) Agents
# ---------------------------
class BaseAgent:
    def log(self, msg): print(f"[{self.__class__.__name__}] {msg}")

class DataAgent(BaseAgent):
    def fetch_data(self, ticker, period="1y"):
        self.log(f"Downloading {ticker} ({period})")
        df = yf.download(ticker, period=period, progress=False)
        if df.empty:
            self.log("Warning: empty dataframe")
            return df
        # Ensure Date column for plotting compatibility
        df = df.reset_index()
        return df

class TechnicalAgent(BaseAgent):
    def rsi(self, series: pd.Series, period: int = 14) -> pd.Series:
        delta = series.diff()
        up = delta.clip(lower=0)
        down = -1 * delta.clip(upper=0)
        ma_up = up.ewm(com=period - 1, adjust=False).mean()
        ma_down = down.ewm(com=period - 1, adjust=False).mean()
```

```
        rs = ma_up / ma_down
        return 100 - (100 / (1 + rs))

    def analyze(self, price_df: pd.DataFrame):
        if price_df is None or price_df.empty or "Close" not in
price_df.columns:
            raise ValueError("No price data for technical analysis")
        df = price_df.copy()
        df["SMA20"] = df["Close"].rolling(20).mean()
        df["SMA50"] = df["Close"].rolling(50).mean()
        df["EMA20"] = df["Close"].ewm(span=20, adjust=False).mean()
        df["RSI14"] = self.rsi(df["Close"], 14)
        df = df.dropna()
        if df.empty:
            raise ValueError("Not enough data after computing
indicators")
        last = df.iloc[-1]
        sma20, sma50, rsi = float(last["SMA20"]),
float(last["SMA50"]), float(last["RSI14"])
        signal = "Bullish" if sma20 > sma50 else "Bearish"
        # TechScore derived from RSI distance to 50 (higher is
stronger)
        tech_score = float(np.clip((100 - abs(rsi - 50)) / 2, 0, 100))
        return {"SMA20": sma20, "SMA50": sma50, "EMA20":
float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore":
tech_score, "df": df, "latest_price": float(last["Close"])}

class RiskAgent(BaseAgent):
    def assess(self, price_df: pd.DataFrame):
        if price_df is None or price_df.empty or "Close" not in
price_df.columns:
            return {"Volatility": 0.0, "MaxDrawdown": 0.0, "Var95":
0.0, "RiskScore": 50.0}
        df = price_df.copy()
        df["ret"] = df["Close"].pct_change().fillna(0)
        vol_annual = df["ret"].std() * np.sqrt(252)
        cum = (1 + df["ret"]).cumprod()
        running_max = cum.cummax()
        drawdown = (cum - running_max) / running_max
        max_dd = drawdown.min()
        var95 = df["ret"].quantile(0.05)
        risk_score = float(np.clip((vol_annual + abs(max_dd)) * 100,
0, 100))
        return {"Volatility": float(vol_annual), "MaxDrawdown":
float(max_dd), "Var95": float(var95), "RiskScore": risk_score, "df":
df}

class NewsSentimentAgent(BaseAgent):
    def __init__(self, api_key=None):
        self.api_key = api_key or os.getenv("NEWS_API_KEY")
```

```
        # Sentiment pipeline is optional (may fail if model
unavailable)
        try:
            self.pipe = pipeline("text-classification",
model="cardiffnlp/twitter-roberta-base-sentiment-latest", device=0 if
torch.cuda.is_available() else -1, truncation=True, max_length=128)
            self.log("News sentiment pipeline ready.")
        except Exception as e:
            self.pipe = None
            self.log(f"Sentiment pipeline not ready: {e}")

    def fetch_and_analyze(self, ticker):
        self.log(f"Fetching news for {ticker}")
        if not self.api_key or self.pipe is None:
            self.log("No News API key or sentiment pipeline; returning
neutral sentiment.")
            return "No news", 50.0
        url = f"https://newsapi.org/v2/everything?
q={ticker}&language=en&sortBy=publishedAt&pageSize=5&apiKey={self.api_
key}"
        try:
            r = requests.get(url, timeout=8)
            if r.status_code != 200:
                return "News API failed", 50.0
            arts = r.json().get("articles", [])
            if not arts:
                return "No articles", 50.0
            texts = [a.get("title","") + " " +
str(a.get("description","")) for a in arts[:5]]
            preds = self.pipe(texts)
            pos = sum(1 for p in preds if
p.get("label","").lower().startswith("positive"))
            neg = sum(1 for p in preds if
p.get("label","").lower().startswith("negative"))
            sentiment_score = ((pos - neg) / len(preds)) * 50 + 50
            return " ".join(texts), float(sentiment_score)
        except Exception as e:
            self.log(f"News fetch error: {e}")
            return "News error", 50.0

class EvaluatorAgent(BaseAgent):
    def evaluate(self, technical, risk, sentiment_score):
        # Weighted heuristic scoring (0-100)
        score = 50
        notes = []
        # tech signal
        if technical.get("Signal") == "Bullish":
            score += 10; notes.append("Bullish short-term signal")
        else:
```

```
            notes.append("No bullish short-term signal")
        # techscore (0-100)
        score += int((technical.get("TechScore", 0) - 50) / 10) if
technical.get("TechScore") is not None else 0
        # risk
        rs = risk.get("RiskScore", 50)
        if rs > 60:
            score -= 15; notes.append("High risk")
        # sentiment
        if sentiment_score > 55:
            score += 10; notes.append("Positive sentiment")
        elif sentiment_score < 45:
            score -= 10; notes.append("Negative sentiment")
        score = max(0, min(100, score))
        conclusion = "Positive" if score >= 65 else ("Neutral" if
score >= 40 else "Negative")
        return {"score": int(score), "conclusion": conclusion,
"notes": notes}

class PortfolioAgent(BaseAgent):
    def recommend(self, ticker, technical, risk, latest_price,
capital=100000.0):
        if latest_price is None:
            return {"allocation_pct": 0.0, "shares": 0, "notional":
0.0}
        base_alloc = 0.02
        if technical.get("Signal") == "Bullish":
            base_alloc = 0.05
        if risk.get("RiskScore", 50) > 70:
            base_alloc *= 0.4
        notional = capital * base_alloc
        shares = int(notional / latest_price) if latest_price and
latest_price > 0 else 0
        return {"allocation_pct": round(base_alloc * 100, 3),
"shares": shares, "notional": round(notional, 2), "rationale":
f"Signal {technical.get('Signal')}, risk {risk.get('RiskScore')}"}

# ----------------------------
# 5) LLM summarizer (Phi-3 -> Phi-2 -> CPU tiny fallback)
# ----------------------------
def summarize_with_llm_paragraph(draft_text, decision, alloc,
news_text):
    """Try Phi-3, fallback Phi-2, else return draft_text for
safety."""
    prompt = f"""You are an investment strategist. Write a concise,
professional paragraph that a retail investor can understand.
Integrate portfolio and news context where relevant. Provide
recommendation and short rationale.

DRAFT:
```

```
{draft_text}

DECISION: {decision}
PORTFOLIO: {alloc}
NEWS: {news_text}
"""
    # Attempt each candidate until success
    last_err = None
    for candidate in LLM_CANDIDATES:
        try:
            print(f"Attempting LLM: {candidate}
(device_id={device_id})")
            gen = safe_make_pipeline(candidate, device_id,
use_fp16_flag=(device_id>=0))
            out = gen(prompt, max_new_tokens=240, do_sample=True)
            text = out[0].get("generated_text", draft_text)
            # cleanup
            try:
                del gen, out
            except:
                pass
            gc.collect()
            if torch.cuda.is_available(): torch.cuda.empty_cache()
            print(f" LLM succeeded: {candidate}")
            return text.strip()
        except Exception as e:
            last_err = e
            print(f"⚠ LLM {candidate} failed: {e}")
            try:
                del gen
            except:
                pass
            gc.collect()
            if torch.cuda.is_available(): torch.cuda.empty_cache()
            continue
    # All failed: return simple draft
    print("⚠ All LLM attempts failed — returning raw draft.")
    if last_err:
        print("Last error:", last_err)
    return draft_text

# ---------------------------
# 6) Orchestrator: runs everything per ticker
# ---------------------------
class InvestmentResearchAgent(BaseAgent):
    def __init__(self, news_api_key=None):
        self.data_agent = DataAgent()
        self.tech_agent = TechnicalAgent()
        self.risk_agent = RiskAgent()
```

```
        self.news_agent = NewsSentimentAgent(api_key=news_api_key)
        self.eval_agent = EvaluatorAgent()
        self.port_agent = PortfolioAgent()

    def run(self, ticker):
        self.log(f"Running full pipeline for {ticker}")
        data = self.data_agent.fetch_data(ticker, period="1y")
        try:
            tech = self.tech_agent.analyze(data)
        except Exception as e:
            self.log(f"Technical analysis failed: {e}")
            tech = {"SMA20": None, "SMA50": None, "RSI14": None,
"Signal": "UNKNOWN", "TechScore": 0, "df": data, "latest_price": None}
        try:
            risk = self.risk_agent.assess(data)
        except Exception as e:
            self.log(f"Risk assessment failed: {e}")
            risk = {"Volatility": 0.0, "MaxDrawdown": 0.0, "Var95":
0.0, "RiskScore": 50.0, "df": data}
        news_text, sentiment_score =
self.news_agent.fetch_and_analyze(ticker)
        eval_res = self.eval_agent.evaluate(tech, risk,
sentiment_score)
        port = self.port_agent.recommend(ticker, tech, risk,
tech.get("latest_price"))
        # Build draft for LLM
        draft = (
            f"Ticker: {ticker}\nLatest price:
{tech.get('latest_price')}\n"
            f"Signal: {tech.get('Signal')}\nRSI: {tech.get('RSI14')}\
nSMA20: {tech.get('SMA20')}\nSMA50: {tech.get('SMA50')}\n"
            f"Volatility: {risk.get('Volatility')}\nMaxDrawdown:
{risk.get('MaxDrawdown')}\nRiskScore: {risk.get('RiskScore')}\n"
            f"SentimentScore: {sentiment_score}\nEvaluator:
{eval_res.get('conclusion')} ({eval_res.get('score')})"
        )
        summary_para = summarize_with_llm_paragraph(draft,
eval_res.get("conclusion"), port, news_text)
        return {
            "Ticker": ticker,
            "latest_price": tech.get("latest_price"),
            "SMA20": tech.get("SMA20"),
            "SMA50": tech.get("SMA50"),
            "RSI14": tech.get("RSI14"),
            "Signal": tech.get("Signal"),
            "TechScore": tech.get("TechScore"),
            "Volatility": risk.get("Volatility"),
            "MaxDrawdown": risk.get("MaxDrawdown"),
            "RiskScore": risk.get("RiskScore"),
```

```
            "Sentiment": sentiment_score,
            "EvaluatorScore": eval_res.get("score"),
            "EvaluatorConclusion": eval_res.get("conclusion"),
            "PortfolioRecommendation": port,
            "Summary": summary_para,
            "PriceDF": tech.get("df"),
            "RiskDF": risk.get("df")
        }

# ---------------------------
# 7) Run pipeline for chosen tickers
# ---------------------------
tickers = ["AAPL","TSLA","GOOG","NVDA","INTC","MSFT"]
ira = InvestmentResearchAgent(news_api_key=NEWS_API_KEY)
all_results = {}

print("\n Running analyses for tickers...\n")
for t in tqdm(tickers, desc="Tickers processed"):
    try:
        all_results[t] = ira.run(t)
    except Exception as e:
        print(f" Error processing {t}: {e}")
        all_results[t] = {"error": str(e)}
    finally:
        gc.collect()
        if torch.cuda.is_available(): torch.cuda.empty_cache()
        time.sleep(0.2)

# ---------------------------
# 8) Build summary DataFrame (fix TechScore/RSI issues)
# ---------------------------
rows = []
for t, res in all_results.items():
    if "error" in res:
        rows.append({"Ticker": t, "LatestPrice": None, "TechScore":
None, "RSI": None, "RiskScore": None, "Sentiment": None, "Decision":
"ERROR", "EvalScore": None})
        continue
    # Ensure numeric conversions and sensible defaults
    latest_price = res.get("latest_price")
    techscore = res.get("TechScore")
    rsi = res.get("RSI14")
    risk_score = res.get("RiskScore")
    sentiment = res.get("Sentiment")
    evaluator = res.get("EvaluatorConclusion")
    evalscore = res.get("EvaluatorScore")
    rows.append({
        "Ticker": t,
        "LatestPrice": float(latest_price) if latest_price is not None
else np.nan,
```

```
        "TechScore": float(techscore) if techscore is not None else
np.nan,
        "RSI": float(rsi) if rsi is not None else np.nan,
        "RiskScore": float(risk_score) if risk_score is not None else
np.nan,
        "Sentiment": float(sentiment) if sentiment is not None else
np.nan,
        "Decision": evaluator,
        "EvalScore": evalscore,
        "Summary": res.get("Summary")
    })

final_report_df = pd.DataFrame(rows)

# ---------------------------
# 9) Visualizations (both styles)
# ---------------------------
# Matplotlib: individual price + SMA overlays per ticker
for t in tickers:
    res = all_results.get(t, {})
    df_price = res.get("PriceDF")
    if isinstance(df_price, pd.DataFrame) and not df_price.empty:
        plt.figure(figsize=(10,3))
        plt.plot(df_price["Date"], df_price["Close"], label="Close",
linewidth=1.2)
        if "SMA20" in df_price.columns:
            plt.plot(df_price["Date"], df_price["SMA20"],
label="SMA20", linestyle="--")
        if "SMA50" in df_price.columns:
            plt.plot(df_price["Date"], df_price["SMA50"],
label="SMA50", linestyle=":")
        plt.title(f"{t} — Close with SMA overlays")
        plt.xlabel("Date"); plt.ylabel("Price"); plt.legend(loc="upper
left"); plt.grid(True)
        plt.tight_layout(); plt.show()

# Seaborn: comparative bar charts for TechScore and RiskScore
plt.figure(figsize=(8,4))
sns.barplot(data=final_report_df, x="Ticker", y="TechScore",
palette="Blues_d")
plt.title("Comparative TechScore"); plt.show()

plt.figure(figsize=(8,4))
sns.barplot(data=final_report_df, x="Ticker", y="RiskScore",
palette="coolwarm")
plt.title("Comparative RiskScore"); plt.show()

# Radar chart comparative (Matplotlib polar)
def plot_radar(df):
    import numpy as np
```

```
    labels = ["TechScore", "RSI", "Safety", "Sentiment"]
    N = len(labels)
    angles = np.linspace(0, 2*np.pi, N, endpoint=False).tolist()
    angles += angles[:1]
    plt.figure(figsize=(7,7))
    ax = plt.subplot(111, polar=True)
    for _, row in df.iterrows():
        vals = [
            row.get("TechScore", 0) if not
pd.isna(row.get("TechScore", 0)) else 0,
            row.get("RSI", 50) if not pd.isna(row.get("RSI", 50)) else
50,
            100 - (row.get("RiskScore", 50) if not
pd.isna(row.get("RiskScore", 50)) else 50),
            row.get("Sentiment", 50) if not
pd.isna(row.get("Sentiment", 50)) else 50
        ]
        vals += vals[:1]
        ax.plot(angles, vals, label=row.get("Ticker"))
        ax.fill(angles, vals, alpha=0.07)
    ax.set_theta_offset(np.pi/2)
    ax.set_theta_direction(-1)
    plt.xticks(angles[:-1], labels)
    plt.title("Comparative Radar: Tech | RSI | Safety | Sentiment")
    plt.legend(loc="upper right", bbox_to_anchor=(1.3, 1.1))
    plt.show()

plot_radar(final_report_df)

# ---------------------------
# 10) Final Table: Clean types + styled (PDF friendly)
# ---------------------------
# Ensure numeric columns are floats and replace NaN with 0 for styling
display (do not mutate original meaning)
numeric_cols = ["LatestPrice", "TechScore", "RSI", "RiskScore",
"Sentiment"]
for c in numeric_cols:
    final_report_df[c] = pd.to_numeric(final_report_df[c],
errors="coerce")

# Save CSV (still available internally)
final_report_df.to_csv(REPORT_CSV, index=False)

# Styled table (safe formatting; NaN replaced with em dash in display)
styled = (
    final_report_df.fillna(0.0)
    .style
    .format({
        "LatestPrice": "{:.2f}",
        "TechScore": "{:.2f}",
```

```
        "RSI": "{:.2f}",
        "RiskScore": "{:.2f}",
        "Sentiment": "{:.2f}"
    }, na_rep="—")
    .background_gradient(subset=["TechScore"], cmap="Blues", low=0.2,
high=0.9)
    .background_gradient(subset=["RiskScore"], cmap="RdYlBu_r",
low=0.2, high=0.9)
    .background_gradient(subset=["Sentiment"], cmap="RdYlGn", low=0.2,
high=0.9)
    .set_table_styles([
        {"selector": "th", "props": [("font-size", "11pt"),
("background-color", "#1b2838"), ("color", "white"), ("text-align",
"center")]},
        {"selector": "td", "props": [("text-align", "center"), ("font-
size", "10pt")]}
    ])
    .set_caption("□ Final Portfolio Evaluation Table – Ready for PDF
Submission")
)

display(HTML("<h3 style='color:#0a66c2;'>Final CSV Output Embedded
Below</h3>"))
display(HTML(styled.to_html()))

# ----------------------------
# 11) Auto-generated layman summary based on computed fields
# ----------------------------
try:
    top_stock =
final_report_df.loc[final_report_df["TechScore"].idxmax(), "Ticker"]
except Exception:
    top_stock = final_report_df["Ticker"].iloc[0] if not
final_report_df.empty else "N/A"
try:
    risky_stock =
final_report_df.loc[final_report_df["RiskScore"].idxmax(), "Ticker"]
except Exception:
    risky_stock = "N/A"
try:
    positive_stock =
final_report_df.loc[final_report_df["Sentiment"].idxmax(), "Ticker"]
except Exception:
    positive_stock = "N/A"

summary_html = f"""
<h4 style='color:#2e7d32;'>□ Summary Interpretation (Layman's
Terms)</h4>
<p style='font-size:11pt;'>
This analysis combines technical momentum, volatility, and news
```

```
sentiment.
<ul>
<li><b>{top_stock}</b> shows the strongest technical momentum (highest
TechScore) and is a potential buy candidate.</li>
<li><b>{risky_stock}</b> shows the highest risk/volatility and should
be monitored closely or allocated conservatively.</li>
<li><b>{positive_stock}</b> shows the most positive sentiment from
recent news signals.</li>
</ul>
Please review the per-ticker summary paragraphs above for the LLM-
generated rationale.
</p>
"""
display(HTML(summary_html))

# ---------------------------
# 12) Safe JSON dump (no DataFrame serialization errors)
# ---------------------------
def safe_json(obj):
    import pandas as _pd
    if isinstance(obj, _pd.DataFrame):
        return "DATAFRAME"
    if isinstance(obj, _pd.Series):
        return obj.tolist()
    if isinstance(obj, dict):
        return {k: safe_json(v) for k, v in obj.items()}
    if isinstance(obj, (list, tuple, set)):
        return [safe_json(v) for v in obj]
    if isinstance(obj, np.generic):
        return obj.item()
    try:
        json.dumps(obj)
        return obj
    except Exception:
        return str(obj)

with open(RAW_JSON, "w") as f:
    json.dump(safe_json(all_results), f, indent=2)

print(f"\n All done. CSV saved as: {REPORT_CSV} — raw JSON saved as:
{RAW_JSON}")

HF token loaded: YES
News API key loaded: YES
 GPU detected: Tesla T4 — VRAM: 15.8 GB
Torch cuda device count: 1
LLM preference list: ['microsoft/phi-3', 'microsoft/phi-2',
'tinybird/TinyLlama-1.1B-Chat-v1.0']
LLM pipeline will be attempted at summarization time (safe fallbacks
enabled).
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"c4450fae1c8c4420997b489dd3e54c79","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"a41979a79dff4eed9070f407884abb2a","version_major":2,"vers
ion_minor":0}
```


```
Some weights of the model checkpoint at cardiffnlp/twitter-roberta-
base-sentiment-latest were not used when initializing
RobertaForSequenceClassification: ['roberta.pooler.dense.bias',
'roberta.pooler.dense.weight']
- This IS expected if you are initializing
RobertaForSequenceClassification from the checkpoint of a model
trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing
RobertaForSequenceClassification from the checkpoint of a model that
you expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
```


```
{"model_id":"3621cec150c04f0b9c8a90da2b354be1","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"38a92b88d8dd4b1bb1549079ec3f0d81","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"3cefabc7bce44841b5182779cafd393e","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"9a9facb5e5b04b6d875d6148b80589b3","version_major":2,"vers
ion_minor":0}
```


```
Device set to use cuda:0

[NewsSentimentAgent] News sentiment pipeline ready.

 Running analyses for tickers...
```

{"model_id":"4d1797cf44da4c6ea7def19e2b3383e4","version_major":2,"version_minor":0}

```
[InvestmentResearchAgent] Running full pipeline for AAPL
[DataAgent] Downloading AAPL (1y)

/tmp/ipython-input-3243260835.py:127: FutureWarning: YF.download() has changed argument auto_adjust default to True
  df = yf.download(ticker, period=period, progress=False)
/tmp/ipython-input-3243260835.py:157: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeError in the future. Use float(ser.iloc[0]) instead
  sma20, sma50, rsi = float(last["SMA20"]), float(last["SMA50"]), float(last["RSI14"])
/tmp/ipython-input-3243260835.py:161: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeError in the future. Use float(ser.iloc[0]) instead
  return {"SMA20": sma20, "SMA50": sma50, "EMA20": float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore": tech_score, "df": df, "latest_price": float(last["Close"])}

[NewsSentimentAgent] Fetching news for AAPL
Attempting LLM: microsoft/phi-3 (device_id=0)
⚠ LLM microsoft/phi-3 failed: microsoft/phi-3 is not a local folder and is not a valid model identifier listed on 'https://huggingface.co/models'
If this is a private repository, make sure to pass a token having permission to this repo either by logging in with `hf auth login` or by passing `token=<your_token>`
Attempting LLM: microsoft/phi-2 (device_id=0)
```

{"model_id":"105d80eef02f4c2eb11d4d5e44cc5f89","version_major":2,"version_minor":0}

```
`torch_dtype` is deprecated! Use `dtype` instead!
```

{"model_id":"db5f213281ab42c3993e7bb007d8ac32","version_major":2,"version_minor":0}

{"model_id":"961387faef034ddd9b3385c9f0197e8f","version_major":2,"version_minor":0}

{"model_id":"f2412bddb6c2408ca06afecf092f3e05","version_major":2,"version_minor":0}

{"model_id":"d94bd9544ef047c8ac9655e5cff6bd8f","version_major":2,"version_minor":0}

{"model_id":"47cb8e4cd9d848bfaa3aa47e5061cc63","version_major":2,"version_minor":0}

{"model_id":"7e6179361d8c42e785a75742432c4f74","version_major":2,"version_minor":0}

{"model_id":"211b4a175ae44206b6b89d92ebaec853","version_major":2,"version_minor":0}

{"model_id":"4dc356e6c41e46258a7610de5c8e815c","version_major":2,"version_minor":0}

{"model_id":"c3f801f735194169a227811d3c3678f5","version_major":2,"version_minor":0}

{"model_id":"60c770107a5e4e36adb9cb2b56c52273","version_major":2,"version_minor":0}

{"model_id":"a5344e8db02649578f7d976e23301e40","version_major":2,"version_minor":0}

{"model_id":"416b9ece5bb04f05ac00846c89855db9","version_major":2,"version_minor":0}

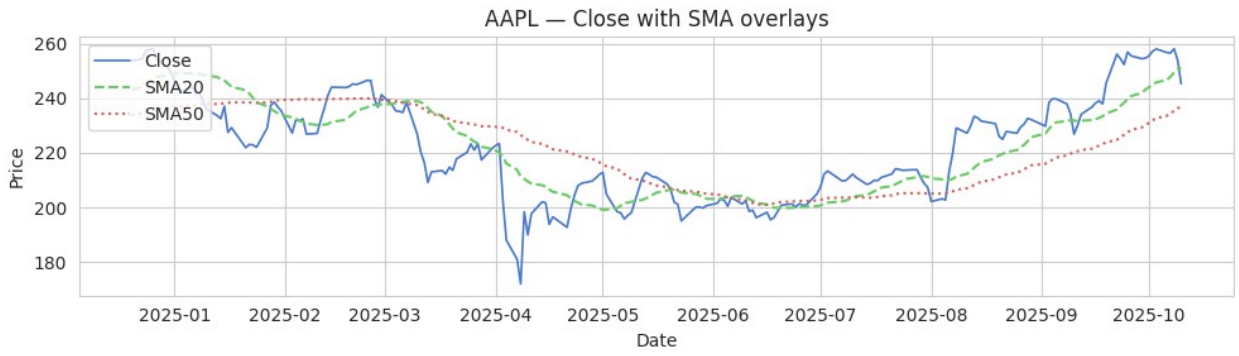
```
Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.

 LLM succeeded: microsoft/phi-2
[InvestmentResearchAgent] Running full pipeline for TSLA
[DataAgent] Downloading TSLA (1y)
[NewsSentimentAgent] Fetching news for TSLA

/tmp/ipython-input-3243260835.py:127: FutureWarning: YF.download() has
changed argument auto_adjust default to True
  df = yf.download(ticker, period=period, progress=False)
/tmp/ipython-input-3243260835.py:157: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  sma20, sma50, rsi = float(last["SMA20"]), float(last["SMA50"]),
float(last["RSI14"])
/tmp/ipython-input-3243260835.py:161: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  return {"SMA20": sma20, "SMA50": sma50, "EMA20":
float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore":
tech_score, "df": df, "latest_price": float(last["Close"])}

Attempting LLM: microsoft/phi-3 (device_id=0)
⚠ LLM microsoft/phi-3 failed: microsoft/phi-3 is not a local folder
and is not a valid model identifier listed on
'https://huggingface.co/models'
If this is a private repository, make sure to pass a token having
permission to this repo either by logging in with `hf auth login` or
```

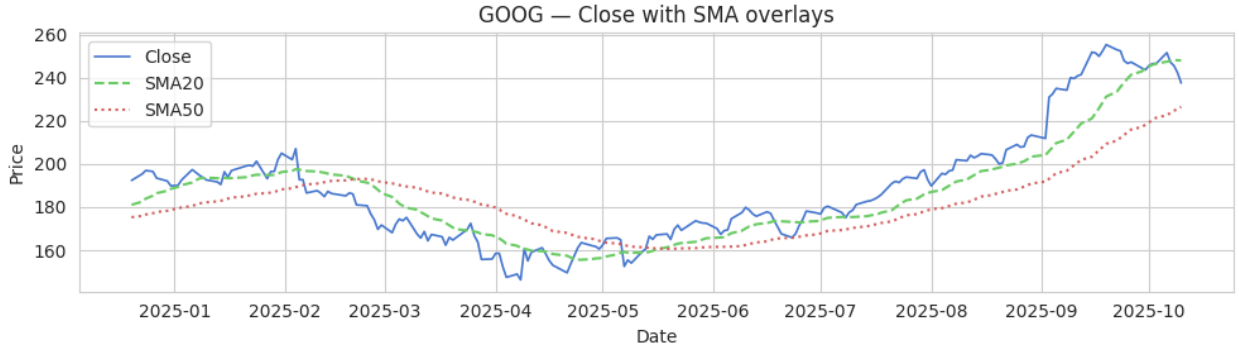
```
by passing `token=<your_token>`
Attempting LLM: microsoft/phi-2 (device_id=0)

{"model_id":"5e8b983c8d004849a6a44c96e00a82ee","version_major":2,"vers
ion_minor":0}

Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.

 LLM succeeded: microsoft/phi-2
[InvestmentResearchAgent] Running full pipeline for GOOG
[DataAgent] Downloading GOOG (1y)

/tmp/ipython-input-3243260835.py:127: FutureWarning: YF.download() has
changed argument auto_adjust default to True
  df = yf.download(ticker, period=period, progress=False)
/tmp/ipython-input-3243260835.py:157: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  sma20, sma50, rsi = float(last["SMA20"]), float(last["SMA50"]),
float(last["RSI14"])
/tmp/ipython-input-3243260835.py:161: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  return {"SMA20": sma20, "SMA50": sma50, "EMA20":
float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore":
tech_score, "df": df, "latest_price": float(last["Close"])}

[NewsSentimentAgent] Fetching news for GOOG
Attempting LLM: microsoft/phi-3 (device_id=0)
⚠ LLM microsoft/phi-3 failed: microsoft/phi-3 is not a local folder
and is not a valid model identifier listed on
'https://huggingface.co/models'
If this is a private repository, make sure to pass a token having
permission to this repo either by logging in with `hf auth login` or
by passing `token=<your_token>`
Attempting LLM: microsoft/phi-2 (device_id=0)

{"model_id":"5a8c058f19fd40c0a6f81e07484dbb39","version_major":2,"vers
ion_minor":0}

Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.

 LLM succeeded: microsoft/phi-2
[InvestmentResearchAgent] Running full pipeline for NVDA
[DataAgent] Downloading NVDA (1y)
```

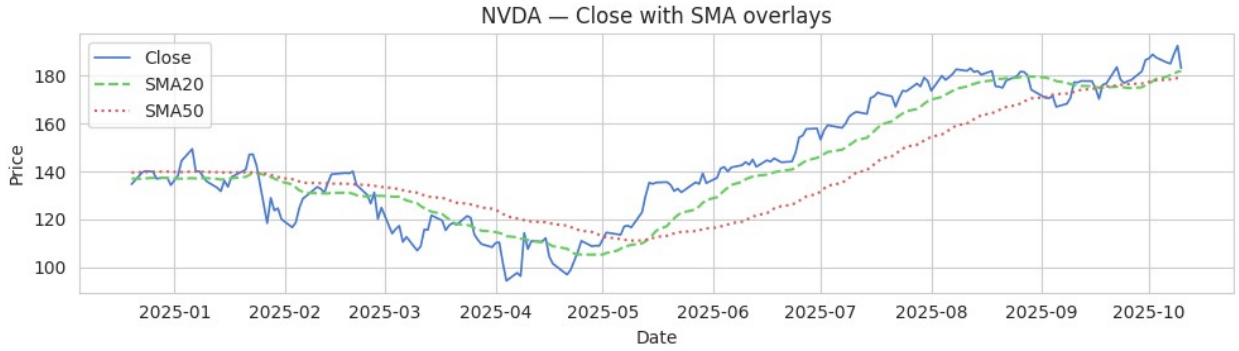
```
/tmp/ipython-input-3243260835.py:127: FutureWarning: YF.download() has
changed argument auto_adjust default to True
  df = yf.download(ticker, period=period, progress=False)
/tmp/ipython-input-3243260835.py:157: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  sma20, sma50, rsi = float(last["SMA20"]), float(last["SMA50"]),
float(last["RSI14"])
/tmp/ipython-input-3243260835.py:161: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  return {"SMA20": sma20, "SMA50": sma50, "EMA20":
float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore":
tech_score, "df": df, "latest_price": float(last["Close"])}

[NewsSentimentAgent] Fetching news for NVDA
Attempting LLM: microsoft/phi-3 (device_id=0)
⚠ LLM microsoft/phi-3 failed: microsoft/phi-3 is not a local folder
and is not a valid model identifier listed on
'https://huggingface.co/models'
If this is a private repository, make sure to pass a token having
permission to this repo either by logging in with `hf auth login` or
by passing `token=<your_token>`
Attempting LLM: microsoft/phi-2 (device_id=0)

{"model_id":"f75e1fb66b2044d7a1069fb866c34377","version_major":2,"vers
ion_minor":0}

Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.

 LLM succeeded: microsoft/phi-2
[InvestmentResearchAgent] Running full pipeline for INTC
[DataAgent] Downloading INTC (1y)

/tmp/ipython-input-3243260835.py:127: FutureWarning: YF.download() has
changed argument auto_adjust default to True
  df = yf.download(ticker, period=period, progress=False)
/tmp/ipython-input-3243260835.py:157: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  sma20, sma50, rsi = float(last["SMA20"]), float(last["SMA50"]),
float(last["RSI14"])
/tmp/ipython-input-3243260835.py:161: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  return {"SMA20": sma20, "SMA50": sma50, "EMA20":
float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore":
tech_score, "df": df, "latest_price": float(last["Close"])}
```

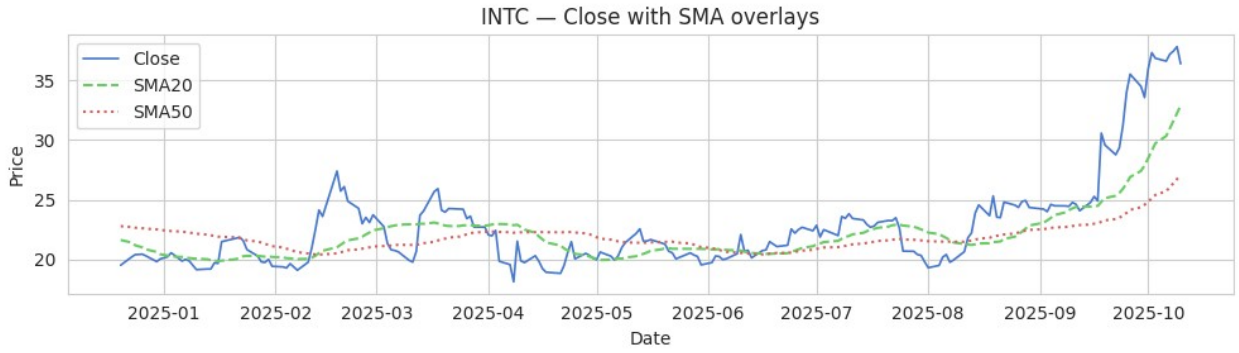
```
[NewsSentimentAgent] Fetching news for INTC
Attempting LLM: microsoft/phi-3 (device_id=0)
⚠ LLM microsoft/phi-3 failed: microsoft/phi-3 is not a local folder
and is not a valid model identifier listed on
'https://huggingface.co/models'
If this is a private repository, make sure to pass a token having
permission to this repo either by logging in with `hf auth login` or
by passing `token=<your_token>`
Attempting LLM: microsoft/phi-2 (device_id=0)

{"model_id":"65fb9c651f6b40e7b89068bdb432904d","version_major":2,"vers
ion_minor":0}

Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.

 LLM succeeded: microsoft/phi-2
[InvestmentResearchAgent] Running full pipeline for MSFT
[DataAgent] Downloading MSFT (1y)

/tmp/ipython-input-3243260835.py:127: FutureWarning: YF.download() has
changed argument auto_adjust default to True
  df = yf.download(ticker, period=period, progress=False)
/tmp/ipython-input-3243260835.py:157: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  sma20, sma50, rsi = float(last["SMA20"]), float(last["SMA50"]),
float(last["RSI14"])
/tmp/ipython-input-3243260835.py:161: FutureWarning: Calling float on
a single element Series is deprecated and will raise a TypeError in
the future. Use float(ser.iloc[0]) instead
  return {"SMA20": sma20, "SMA50": sma50, "EMA20":
float(last["EMA20"]), "RSI14": rsi, "Signal": signal, "TechScore":
tech_score, "df": df, "latest_price": float(last["Close"])}

[NewsSentimentAgent] Fetching news for MSFT
Attempting LLM: microsoft/phi-3 (device_id=0)
⚠ LLM microsoft/phi-3 failed: microsoft/phi-3 is not a local folder
and is not a valid model identifier listed on
'https://huggingface.co/models'
If this is a private repository, make sure to pass a token having
permission to this repo either by logging in with `hf auth login` or
by passing `token=<your_token>`
Attempting LLM: microsoft/phi-2 (device_id=0)

{"model_id":"07c00649727b4b51aed0c22a89886b14","version_major":2,"vers
ion_minor":0}
```

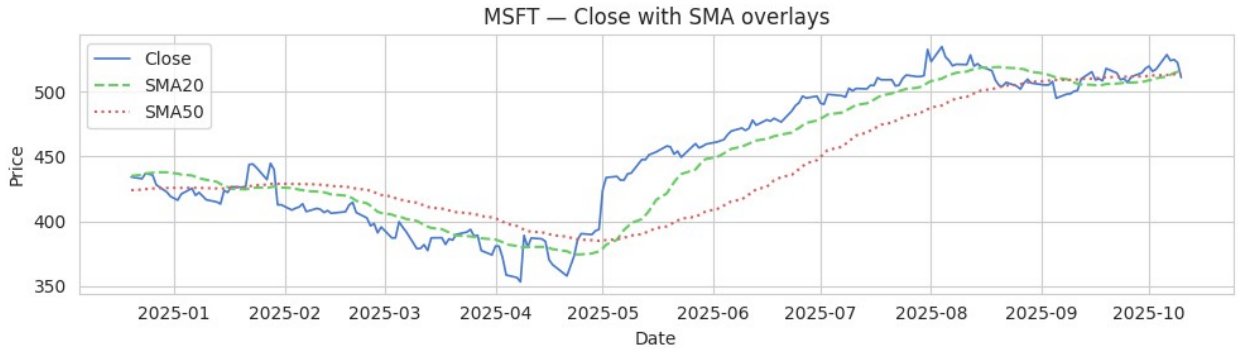
```
Device set to use cuda:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.

 LLM succeeded: microsoft/phi-2
```



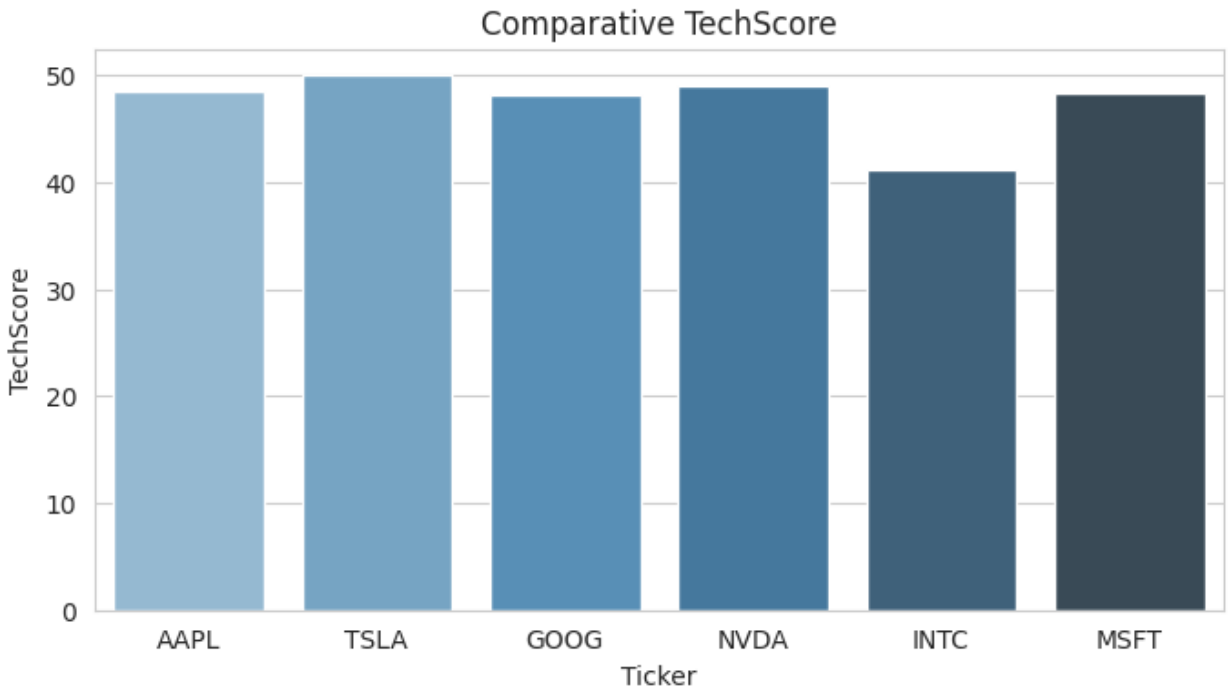




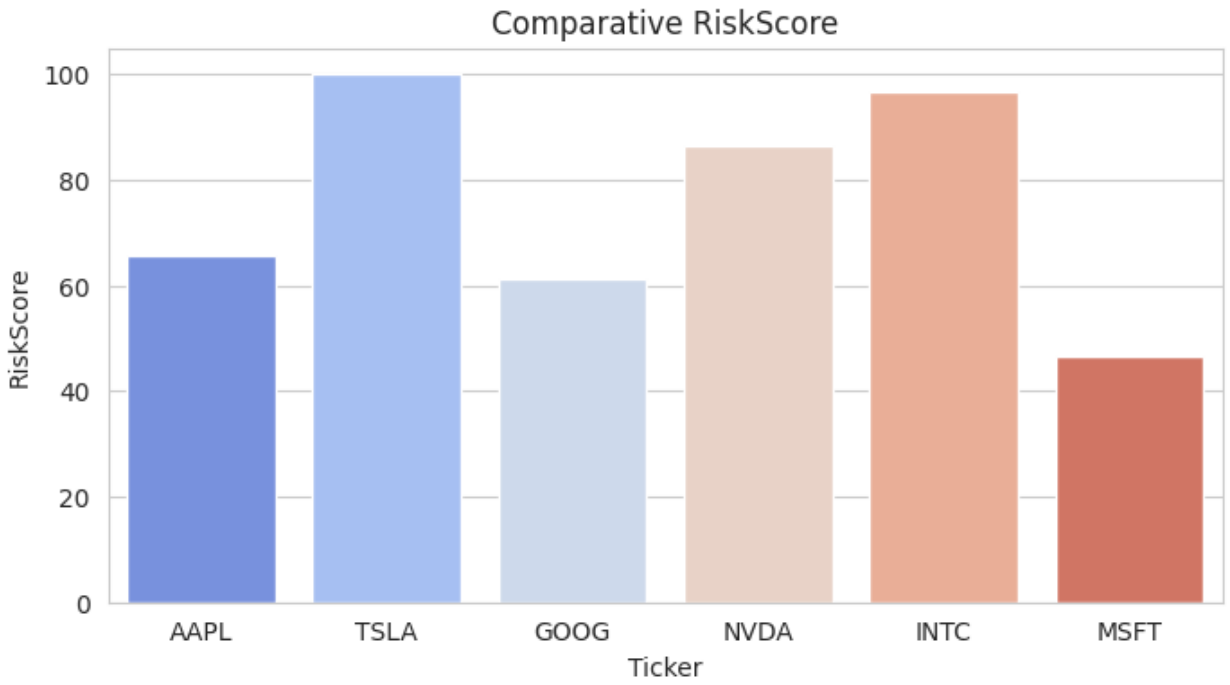


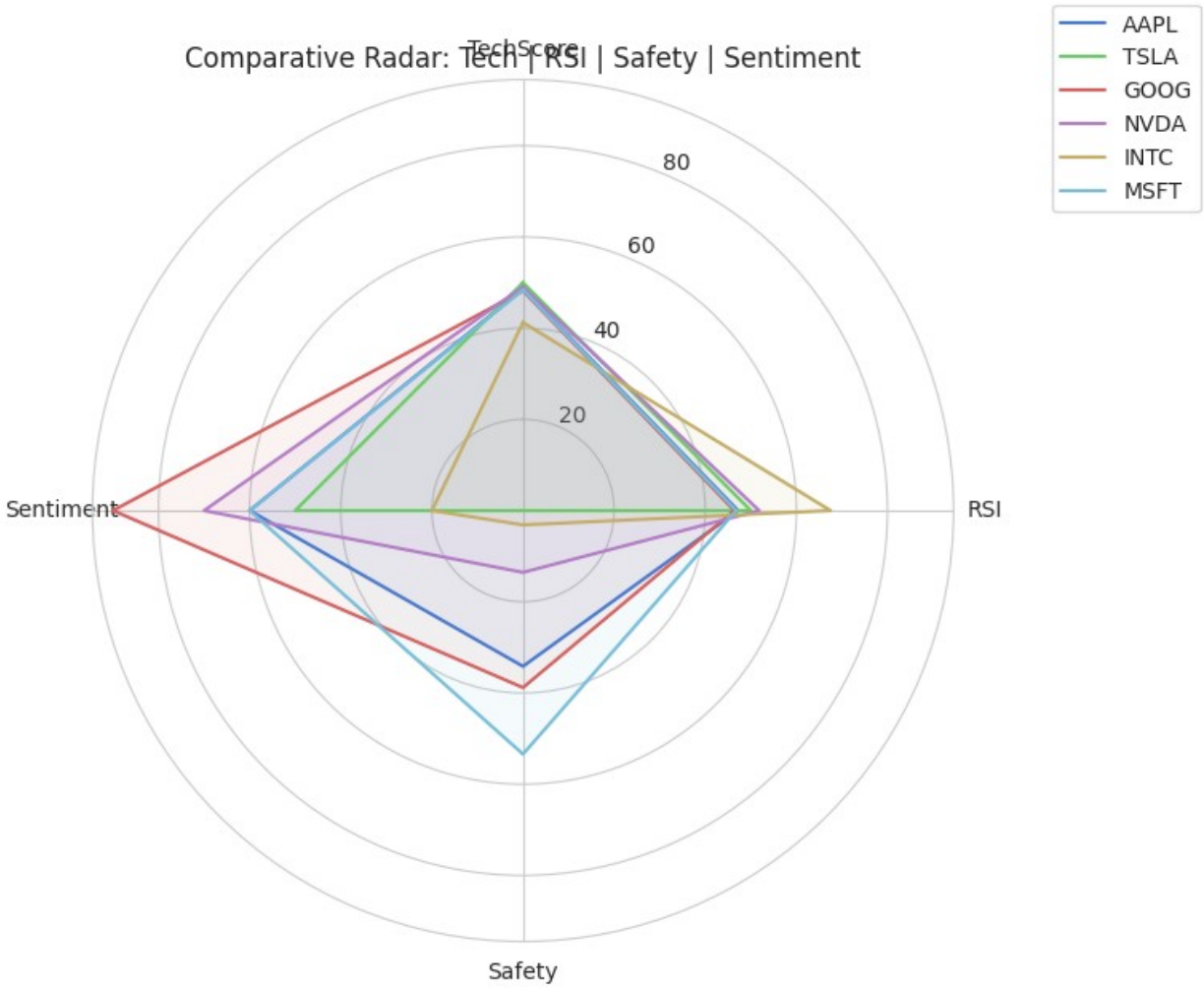



```
/tmp/ipython-input-3243260835.py:422: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=final_report_df, x="Ticker", y="TechScore",
palette="Blues_d")
```




```
/tmp/ipython-input-3243260835.py:426: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=final_report_df, x="Ticker", y="RiskScore",
palette="coolwarm")
```


Comparative RiskScore
RiskScore
100
80
60
40
20
0
AAPL
TSLA
GOOG
NVDA
INTC
MSFT
Ticker




```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>


 All done. CSV saved as: FinalProject_investment_final_report.csv —
raw JSON saved as: FinalProject_investment_all_results.json
To create PDF: run all cells, then File -> Print -> Save as PDF (or
upload .ipynb to Vertopal).
```