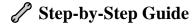
i would like to build a corn and soybean news headline sentiment analyszer

ChatGPT said:

That sounds like a great project—useful for agri-commodity traders, analysts, or even farmers keeping an eye on the market. Here's a breakdown to help get you started on building a **Corn and Soybean News Headline Sentiment Analyzer**:



1. Data Collection

You'll need headlines related to corn and soybeans.

- Sources:
 - Google News RSS feeds
 - o Reuters, Bloomberg, AgWeb, Successful Farming
 - o APIs like NewsAPI or web scraping with BeautifulSoup/Scrapy
- Search Keywords:
 - o "corn prices", "soybean harvest", "corn yield forecast", "soybean USDA report" etc.

2. Preprocessing Headlines

Basic cleaning:

- Lowercasing
- Removing punctuation, stopwords (optional)
- Tokenization
- Lemmatization (optional)

3. Sentiment Analysis Models

Option A: Rule-Based

• Use **VADER** sentiment analyzer (great for short text like headlines)

python CopyEdit

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

```
analyzer = SentimentIntensityAnalyzer()
scores = analyzer.polarity_scores("Soybean prices drop due to export
slowdown")
```

Option B: Machine Learning

• Fine-tune a **transformer model** (like BERT) on labeled agri-news sentiment data, if you have access.

4. Labeling (if ML-based)

• Positive: Good harvest, rising prices, increased demand

• Negative: Drought, yield cuts, falling exports

• Neutral: USDA reports, forecasts

5. Categorization

Separate whether the headline is about:

- Corn
- Soybeans
- Both

You could use keyword matching or an NLP classifier for this.

6. Dashboard (Optional)

Use Streamlit or Dash to show:

- Daily sentiment trends
- Word clouds
- Sentiment by crop

7. Bonus Ideas

- Track how sentiment correlates with **futures prices**
- Alert system for high-impact negative headlines

Want help starting with code for scraping or sentiment scoring? Or building a dataset structure?

40

You said:

is there a bert model suitable for this task that i can use to fine tune with my data

ChatGPT said:

Yes! For a corn and soybean news headline sentiment analyzer, you can fine-tune a **BERT**based model that's already been pretrained on a similar domain (news or finance) for better results than using general-purpose BERT.

Here are some great options you can fine-tune:



BERT Models Suitable for Your Task

1. FinBERT

- Trained on financial news from Reuters.
- Very suitable for economic/agriculture news headlines.
- Already performs sentiment classification (positive, negative, neutral).
- You can fine-tune it on **corn/soybean-specific sentiment labels**.

```
bash
CopyEdit
!pip install transformers
python
CopyEdit
from transformers import AutoTokenizer, AutoModelForSequenceClassification
tokenizer = AutoTokenizer.from pretrained("ProsusAI/finbert")
model =
AutoModelForSequenceClassification.from pretrained("ProsusAI/finbert")
```

2. BERTweet

- Trained on tweets (short texts), so it's good for short headlines.
- If your headlines are similar in length/style to social media snippets, this works well.
- Requires custom fine-tuning for sentiment.

3. DistilBERT (base)

- Lightweight and fast.
- Good baseline if you're starting from scratch and want quick iteration.

✓ Recommended: Use FinBERT

Because your domain is **financial/agricultural news**, and the text is **short headlines**, FinBERT is a great match.

Fine-Tuning Overview

You'll need:

- Labeled dataset: headlines + sentiment label (positive/neutral/negative)
- Format into text, label pairs
- Tokenize using FinBERT tokenizer
- Use Trainer API or custom training loop

Would you like a working example notebook or script to fine-tune FinBERT on your custom data?

You said:

how can i use moving average to observe trend on market price data

ChatGPT said:

Using a **moving average** (**MA**) is a classic and effective way to smooth out short-term fluctuations and highlight trends in market price data—perfect for analyzing corn and soybean price movements.

What is a Moving Average?

A **moving average** takes the average of a set number of previous data points to smooth out price changes.

- **Simple Moving Average (SMA):** Basic average over a window.
- Exponential Moving Average (EMA): Weighs recent data more heavily (better for fast-moving markets).



Example in Python (with Pandas)

Let's say you have daily price data:

```
python
CopyEdit
import pandas as pd
import matplotlib.pyplot as plt
# Sample price data (replace with your own CSV or API data)
df = pd.read csv("corn prices.csv", parse dates=["Date"])
df = df.sort values("Date")
# Add Simple Moving Averages
df['SMA_7'] = df['Price'].rolling(window=7).mean()  # 1-week trend
df['SMA_30'] = df['Price'].rolling(window=30).mean()  # 1-month trend
# Plot it
plt.figure(figsize=(12,6))
plt.plot(df['Date'], df['Price'], label='Daily Price', alpha=0.5)
plt.plot(df['Date'], df['SMA 7'], label='7-Day SMA')
plt.plot(df['Date'], df['SMA 30'], label='30-Day SMA')
plt.legend()
plt.title("Corn Price with Moving Averages")
plt.xlabel("Date")
plt.ylabel("Price")
plt.grid(True)
plt.show()
```

Which is the How to Interpret It

- If the shorter MA crosses above the longer MA \rightarrow bullish signal (trend may be going
- If the shorter MA crosses below the longer $MA \rightarrow bearish$ signal.
- Helps filter noise and makes it easier to see direction.

X Optional Enhancements

- Use EMA with df['EMA 10'] = df['Price'].ewm(span=10).mean()
- Add MACD, Bollinger Bands, or overlay sentiment signals from your NLP model.
- Combine with sentiment to create a "price vs. mood" chart.

Want to integrate this with your sentiment analyzer or fetch real commodity prices (e.g., from Quandl, Yahoo Finance, or USDA)?

You said:

USDA corn

Unable to display visualization I would like to process this:

ChatGPT said:

Your dataset is loaded! It appears to be a USDA corn market data table with monthly reports, starting from January 2014. It includes a lot of columns like:

- Area Planted, Area Harvested
- Yield, Production, Imports, Total Supply
- Feed and Residual, FSI (Food, Seed & Industrial)

•

You said:

prices historical corn

Unable to display visualization

I would like process this to find short term and long term trends using moving average

ChatGPT said:

It looks like the file doesn't contain a column named "Date", which caused an error when trying to parse it as a date.

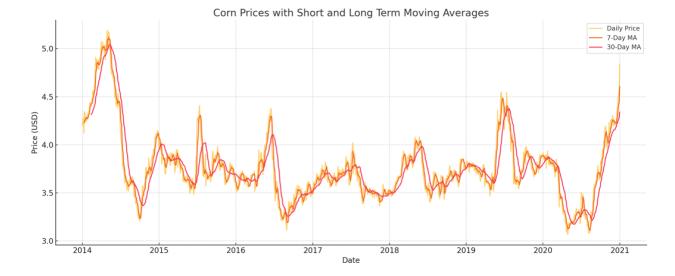
Let me inspect the file structure first to identify the correct date column.

Thanks! The file contains two columns:

- date: the trading date
- value: the corn price on that date

Let's now parse the date properly and compute **short-term and long-term moving averages** (e.g., 7-day and 30-day).

Corn Prices with Short and Long Term Moving Averages



Here's your chart showing:

- Daily corn prices
- A **7-day moving average** (short-term trend)
- A **30-day moving average** (long-term trend)

This helps visualize short-term fluctuations versus the broader trend. Want to explore things like price crossovers (signals), seasonal effects, or overlay sentiment scores?

You said:

how can i interpret the moving average to understand short and long term trends

ChatGPT said:

Great question—interpreting moving averages (MAs) is key to understanding market momentum and trend direction. Here's how to read short-term vs. long-term moving averages in the context of your corn price data:

✓ 1. Trend Direction

- MA is rising → Uptrend
- MA is falling → Downtrend

Short-term MA (e.g., 7-day)

- Reacts quickly to price changes
- Shows immediate momentum
- Useful for short-term trading signals or recent volatility

♦ Long-term MA (e.g., 30-day)

- Smoother curve
- Filters out noise
- Reflects the broader market direction

2. Crossovers: Trend Change Signals

- Bullish Signal ("Golden Cross"):
 - Short-term MA crosses above long-term MA
 - o Suggests a potential upward trend
- Bearish Signal ("Death Cross"):
 - o Short-term MA crosses below long-term MA
 - o Indicates a possible downward trend

You can visually inspect these crossover points on your chart.

3. Support and Resistance

- In an uptrend, the **longer-term MA** often acts as **support** (prices bounce off it).
- In a downtrend, it acts as **resistance**.

4. Confirmation Tool

You shouldn't use MAs alone. Combine them with:

- Volume data
- Sentiment signals (from your NLP analyzer!)
- Fundamental events (e.g., USDA reports)

***** Example Interpretation

Let's say:

- Daily price dips but stays **above the 30-day MA** → Long-term uptrend still intact.
- 7-day MA starts to flatten and drops **below** the 30-day MA → Trend may be weakening or reversing.

Would you like me to:

- Highlight crossover points in your chart?
- Calculate trend direction in a table format?
- Combine this with sentiment signals from your analyzer?

You said:

give me the code to process this file to calculate 7 day and 30 day moving average

ChatGPT said:

Sure! Here's a complete Python snippet using **Pandas** to calculate the 7-day and 30-day moving averages from your corn price file (prices_historical_corn.csv). You can run this in a script, Jupyter notebook, or your data pipeline:

✓ Python Code to Calculate Moving Averages

```
python
CopyEdit
import pandas as pd
import matplotlib.pyplot as plt
# Load the data
df = pd.read csv("prices historical corn.csv")
# Rename and parse dates
df = df.rename(columns={"date": "Date", "value": "Price"})
df["Date"] = pd.to datetime(df["Date"])
df = df.sort values("Date")
# Calculate moving averages
df["MA 7"] = df["Price"].rolling(window=7).mean()
df["MA 30"] = df["Price"].rolling(window=30).mean()
# Save to CSV (optional)
df.to csv("corn prices with moving averages.csv", index=False)
# Plot the results
plt.figure(figsize=(14, 6))
plt.plot(df["Date"], df["Price"], label="Daily Price", alpha=0.6)
plt.plot(df["Date"], df["MA 7"], label="7-Day MA")
plt.plot(df["Date"], df["MA 30"], label="30-Day MA")
plt.title("Corn Prices with 7-Day and 30-Day Moving Averages")
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.legend()
plt.grid(True)
```

```
plt.tight layout()
plt.show()
```

This code will give you both a **visual plot** and a **CSV file** containing the moving averages. Let me know if you want to add automatic signal detection (crossovers) or combine it with sentiment scores.

You said:

using the smoothed 7 and 30 day moving averages how can i infer whether at a specific time the sentiment changed

ChatGPT said:

That's a sharp question! You can absolutely **infer changes in market sentiment** by analyzing the behavior of **short-term vs. long-term moving averages**—like the 7-day and 30-day MAs. Here's how you can interpret sentiment shifts:



1. Crossover Events = Sentiment Shifts

- **Bullish Crossover (Positive Sentiment Shift)**
 - The 7-day MA crosses above the 30-day MA.
 - Implies that **recent prices are rising faster** than the longer-term trend.
 - Often signals **renewed buying interest**, optimism, or reaction to good news (e.g., favorable USDA report, increased exports, lower supply).
- **Bearish Crossover (Negative Sentiment Shift)**
 - The 7-day MA crosses below the 30-day MA.
 - Suggests that **short-term momentum is weakening**.
 - May reflect negative sentiment from bad weather, trade disruptions, or lower demand.

2. Divergence Between 7 & 30 Day MAs

- Widening Gap (7 MA pulling away above 30 MA):
 - o Strengthening **bullish** sentiment
 - o Often follows strong rallies or unexpected positive news
- Converging Gap (7 MA approaching or flattening toward 30 MA):
 - Weakening momentum
 - o Possible sentiment shift toward **neutral** or **bearish**



% Code to Detect Sentiment Shifts from MA Crossovers

Here's how you can calculate crossover points (potential sentiment shifts):

```
python
CopyEdit
\# Create signal column: 1 = bullish crossover, -1 = bearish crossover, 0 = no
crossover
df["Signal"] = 0
df.loc[(df["MA 7"] > df["MA 30"]) & (df["MA 7"].shift(1) <=
df["MA 30"].shift(1)), "Signal"] = 1 # Bullish crossover
df.loc[(df["MA 7"] < df["MA 30"]) & (df["MA 7"].shift(1) >=
df["MA 30"].shift(1)), "Signal"] = -1 # Bearish crossover
# Filter crossover events
crossover events = df[df["Signal"] != 0][["Date", "Price", "Signal"]]
print(crossover events)
```

This will give you a table of dates where sentiment may have shifted.

③ Use Case for Your Analyzer

You can correlate these crossover points with:

- News headlines or your sentiment analysis output (from FinBERT or VADER)
- USDA or geopolitical events
- Futures price reactions

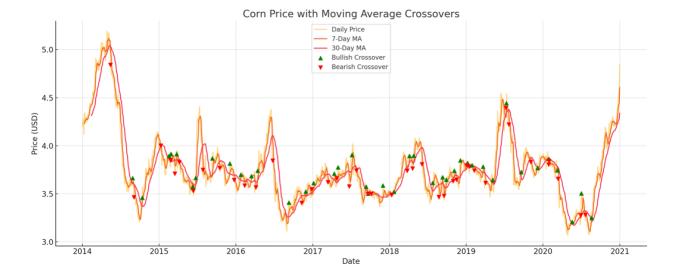
Would you like help visualizing these crossover points on a chart or correlating them with headline sentiment values?

You said:

yes show me the way to visualize crossover points

ChatGPT said:

Corn Price with Moving Average Crossovers



Here's your chart with:

- **Bullish crossovers** marked with green upward arrows (†)
- **Bearish crossovers** marked with red downward arrows (1)

These crossover points highlight where **short-term sentiment** may have shifted relative to the long-term trend.

Want to layer in **headline sentiment scores** or label these points with specific events (e.g., USDA reports)?

You said:

give me the code to visualize crossovers

ChatGPT said:

Here's the complete Python code to visualize **bullish and bearish crossover points** based on the 7-day and 30-day moving averages:

Crossover Visualization Code

```
python
CopyEdit
import pandas as pd
import matplotlib.pyplot as plt

# Load your price data
df = pd.read_csv("prices_historical_corn.csv")
df = df.rename(columns={"date": "Date", "value": "Price"})
df["Date"] = pd.to_datetime(df["Date"])
```

```
df = df.sort values("Date")
# Calculate Moving Averages
df["MA 7"] = df["Price"].rolling(window=7).mean()
df["MA 30"] = df["Price"].rolling(window=30).mean()
# Detect crossover signals
df["Signal"] = 0
df.loc[
    (df["MA 7"] > df["MA_30"]) & (df["MA_7"].shift(1) <=
df["MA 30"].shift(1)),
    "Signal"
] = 1 # Bullish crossover
df.loc[
    (df["MA 7"] < df["MA 30"]) & (df["MA 7"].shift(1) >=
df["MA 30"].shift(1)),
    "Signal"
] = -1 \# Bearish crossover
# Extract crossover points
bullish = df[df["Signal"] == 1]
bearish = df[df["Signal"] == -1]
# Plotting
plt.figure(figsize=(14, 6))
plt.plot(df["Date"], df["Price"], label="Daily Price", alpha=0.5)
plt.plot(df["Date"], df["MA 7"], label="7-Day MA")
plt.plot(df["Date"], df["MA 30"], label="30-Day MA")
# Mark crossovers
plt.scatter(bullish["Date"], bullish["Price"], color="green", marker="^",
s=100, label="Bullish Crossover")
plt.scatter(bearish["Date"], bearish["Price"], color="red", marker="v",
s=100, label="Bearish Crossover")
# Final touches
plt.title("Corn Price with 7-Day and 30-Day Moving Average Crossovers")
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.legend()
plt.grid(True)
plt.tight layout()
plt.show()
```

This will give you a clear visual of:

- Price line
- Short- and long-term moving averages
- Arrows marking points of **trend sentiment shift**

Let me know if you want to **annotate with headline sentiment**, or connect this with FinBERT outputs too!

You said:

I have sentiment crossover points now. I also have news headlines. in order for a change of sentiment to occur some lag must pass after emergence of a headline. how may i determine which headline is responsible for which sentiment crossover

ChatGPT said:

This is a **super insightful and realistic problem**—you're trying to **attribute a sentiment shift** (**crossover**) to one or more news headlines that likely **preceded it by a short time**. The idea is to **estimate causal influence with a lag window**. Here's how to approach it:

Conceptual Approach

Assumptions

- 1. Headlines influence market sentiment with a delay.
- 2. Crossover points reflect aggregated trader response to prior information.
- 3. You want to **match headlines to the nearest future crossover** within a reasonable time window (e.g., 1–5 days).

% Step-by-Step Strategy

1. Define a Lag Window

Pick a lag window—say, 1 to 5 days. That means:

"Any headline occurring within 5 days before a crossover may have contributed to it."

2. Loop Over Crossover Points

For each sentiment crossover, look backward in time by lag_days, and collect all headlines that appeared in that range.

3. Optional: Rank Headlines

If you have **sentiment scores** for headlines:

- Rank them by absolute sentiment (e.g., strongest negative or positive).
- Or use an average of sentiment scores if there are multiple headlines.

Sample Code Structure

Assuming:

- crossovers df: rows with Date and Signal columns
- headlines df: rows with Date, Headline, and optionally Sentiment

```
python
CopyEdit
from datetime import timedelta
# Set lag window
lag days = 5
# Convert to datetime if needed
crossovers df["Date"] = pd.to datetime(crossovers df["Date"])
headlines df["Date"] = pd.to datetime(headlines df["Date"])
# Create a list to store mappings
matches = []
for , crossover in crossovers df.iterrows():
    crossover_date = crossover["Date"]
    window_start = crossover_date - timedelta(days=lag_days)
    # Find headlines within the lag window
    relevant headlines = headlines df[
        (headlines df["Date"] >= window start) &
        (headlines df["Date"] <= crossover date)</pre>
    ].copy()
    relevant headlines["Crossover Date"] = crossover date
    relevant headlines["Signal"] = crossover["Signal"]
   matches.append(relevant headlines)
# Combine all results
matched headlines df = pd.concat(matches).sort values(["Crossover Date",
"Date"])
```

What You Get

Each row now tells you:

- A headline
- Its publish date
- The crossover date it's potentially influencing
- The **signal direction** (bullish/bearish)

Enhancements (Optional)

- Add a "time-to-impact" column (Crossover Date Headline Date)
- Use **sentiment weight** to measure impact probability
- Visualize: build a timeline or annotated chart with headlines near crossover points

Want me to help you plug in your real data and generate this matching DataFrame? You can upload the headline CSV, and I'll help wire it up.

You said:

All News Corn Soybeans

Unable to display visualization

I want to create two dataframes for one containing news with soy and one with news containing corn

ChatGPT said:

Your news dataset is loaded! It has:

• Date: Publication date

• Healline: (misspelled) headline

• News: Full article text

Let's fix the column name and then create two DataFrames:

- 1. One with headlines/articles mentioning "corn"
- 2. One with headlines/articles mentioning "soy" (soybean, soybeans, soy)

Let's do that now.

Corn News

Date	Headline	News
0 ²⁰²⁰ - 12-31	Export Taxes on Argentina Soybeans back to 33%	As 2020 comes to a close, farmers in Argentina have a lot of things to worry about. The weather thus far this growing season has been problematic, a prolonged strike at Argentine ports has disrupted their marketing plans, the Argentine economy is struggling to remain afloat, and now taxes on grain exports will increase the first of the year.Back in October, the Fernandez administration lowered the commodity export taxes somewhat in an effort to spur more farmer sales of their 2019/20 grain production. The soybean export tax was lowered from 33% to 30%, but it has been gradually rising since then. The tax increased to 31.5% in November, 32% in

News

December, and now it will be 33% on January 1st. The soybean meal tax will go from 27% to 30% in January and soybean oil will go from 28% to 31%. Their effort to increase farmer sales generally failed because the temporary tax reduction was not enough of an incentive for farmers to let go of their grain, which they view as a hedge against inflation and a potential devaluation of the Argentine peso. The government has been resisting a significant devaluation of the Argentine peso because a weaker currency would make it even harder to pay back their staggering debts. The prior administration of Mouricio Macri had been gradually reducing the export taxes, but that process was interrupted by the worsening economic situation in the country. The tax had been lowered to 24.7% for soybeans and 6.7% for corn and wheat. Export taxes on agricultural commodities is one of the primary sources of revenue for the government.', "Increasing the export taxes was one of the first actions taken by the Alberto Fernandez and Cristina Kirchner administration in an effort to shore up the country's finances.

Rainfall in Argentina continues to be scattered with less than ideal coverage. This has allowed some areas to maintain adequate soil moisture while other areas are rated short to very short on soil moisture. The best rains over the holiday weekend were across the western areas. The forecast is calling for improved chances of rainfall mid-week and then a return of hotter and dryer conditions later this week and into next week. Two things led me to lower the corn estimate. First, the corn in Argentina was rated only 20% good to excellent as of the middle of last week. Given the weather since then, the percentage rated good to excellent could decline again this week. Obviously, that is a very low number. Secondly, I do not think that all the intended corn acreage will end up being planted. It is already the end of December, the weather continues to be problematic and there are only a couple of weeks left in the planting window. In the face of adverse weather, farmers may opt to not risk planting an expensive corn crop later than normal.Corn in Argentina was 61.2% planted last week compared to 76% last year and 80.2% average. This represented an advance of 6% for the week. In the core production areas, the corn was 90% planted with 75-85% planted in the southern areas and less than 5% planted in far northern Argentina. The early corn was 64.5% planted and the late corn was 57.6% planted. The nationwide corn crop was 20.3% pollinating compared to 43% last year and 39.4% average and 3.7% filling grainThe corn was rated 15% poor to very poor, 64% fair, and 20% good to excellent. The good to excellent compares to 24% last week and 32% last year. The soil moisture for the corn was rated 24% short to very short and 76% favorable to optimum. The favorable to optimum compares to 81% last week and 66% last

2020/21 Argentina Corn Only 61% Planted vs. 80% Average

News

year.', "Dryer than normal conditions continue to slow the corn planting and the corn development. It is already the end of December and the corn planting is about 20% behind the average planting pace. The corn planting window in Argentina will close in 2-3 weeks and unless the weather improves quickly allowing a more rapid planting pace, I don't think the farmers in Argentina will plant all their intended corn acreage.

The prolonged strike by port workers in Argentina is worrisome for Brazilian wheat millers especially if the work stoppage isn't resolved by the end of the year. Currently, there are 22 ports in Argentina have been paralyzed by the strike of over two weeks and as a result, there are more than 100 vessels waiting to load soybean meal, soybeans, corn, and wheat.", 'Brazil needs to import approximately half of its wheat needs and its principal supplier is Argentina. The Brazilian wheat harvest has just concluded so millers are in the process of using Brazilian wheat, but if the strike drags on into early January, some Brazilian wheat millers could run out of wheat. During the first three weeks of December, Brazil imported 200,000 tons of wheat from all sources compared to 650,000 tons for the month of December 2019. From January through November of 2020, Brazil imported 5.87 million tons of wheat with 4.33 million tons from Argentina (73.7% of the total), 733,800 tons from the U.S. (12.5% of the total), 237,600 tons from Russia (4% of the total), 235,000 from Uruguay (4% of the total), and 218,500 tons from Paraguay and others (3.7% of the total). Argentine farmers who just finished harvesting their wheat are not allowed to enter the ports to deliver their wheat due to the strike, but if the strike ends by January 1st, wheat millers in Brazil should be restocked in time to continue operations.

Strike at
6 2020- Argentine Ports
12-24 Worry Brazilian
Wheat Millers

Soy News

Date Headline

News

0 2020-12-31 Export Taxes on Argentina Soybeans back to 33% As 2020 comes to a close, farmers in Argentina have a lot of things to worry about. The weather thus far this growing season has been problematic, a prolonged strike at Argentine ports has disrupted their marketing plans, the Argentine economy is struggling to remain afloat, and now taxes on grain exports will increase the first of the year.Back in October, the Fernandez administration lowered the commodity export taxes somewhat in an effort to spur more farmer sales of their 2019/20 grain production. The soybean export tax was lowered from 33% to 30%, but it has been gradually rising since then. The tax increased to 31.5% in November, 32% in December, and now it will be 33% on January 1st. The soybean meal tax will go from 27% to 30% in January and soybean oil will go from 28% to

News

31%. Their effort to increase farmer sales generally failed because the temporary tax reduction was not enough of an incentive for farmers to let go of their grain, which they view as a hedge against inflation and a potential devaluation of the Argentine peso. The government has been resisting a significant devaluation of the Argentine peso because a weaker currency would make it even harder to pay back their staggering debts. The prior administration of Mouricio Macri had been gradually reducing the export taxes, but that process was interrupted by the worsening economic situation in the country. The tax had been lowered to 24.7% for soybeans and 6.7% for corn and wheat. Export taxes on agricultural commodities is one of the primary sources of revenue for the government.', "Increasing the export taxes was one of the first actions taken by the Alberto Fernandez and Cristina Kirchner administration in an effort to shore up the country's finances.

The weather in Brazil has slowly improved over the past few weeks, but the rainfall is still scattered in nature instead of the widespread rains that are typical this time of the year. Some of the areas where the soil moisture has not been fully recharged include: Rio Grande do Sul, parts of Mato Grosso do Sul, southwestern Goias, isolated pockets in Mato Grosso, and northeastern Brazil. The forecast for this week is calling for improved rainfall across much of central and south-central Brazil. The rainfall over the last few days has been good enough to stabilize the crop, at least for the time being, but if the rainfall starts to disappoint, the soybean estimates could move lower. The earlier planted soybeans are setting pods and filling pods, so the crop cannot afford any prolonged periods of dry weather. The latest developing soybeans are in southern Brazil especially Rio Grande do Sul and also in northeastern Brazil. The soybeans in those two areas are starting to flower.', "Mato Grosso Soybeans-Imea estimates that at least 2.5% of the state's soybeans had to be replanted due to poor plant populations. They estimated that the soybean acreage increased 3% to 10.3 million hectares (25.4) million acres). Due to the irregular weather, Imea lowered the statewide soybean yield estimate to 57.4 sacks per hectare (51.3) bu/ac), which would be down 3% compared to last year. The 2020/21 soybean production in the state was estimated at 35.4 million tons, which is up only 0.2% compared to last year.", 'Imea estimates that farmers in the state have forward contracted 66.4% of their anticipated soybean production compared to 51% last year at this time and 47% average. Farmers only increased their forward sales 2% in November due to concerns about potential lower yields. Imea indicated that the soybean price in

Recent Rains in

2020- Central Brazil Help

12-30 to Stabilize Soybean

Crop

2020 averaged R\$ 111.06 per sack (\$9.70 per bushel), which was a 65% increase over 2019. The high price for the year was approximately R\$ 170.00 per sack (approximately \$14.85 per bushel). It is estimated that about 60% of the forward sales were made when prices were much lower than the current prices. The conclusion of Highway BR-163 between northern Mato Grosso and ports on the Amazon River in early 2020 was an important milestone for the state of Mato Grosso. With the resulting lower transportation costs, there is now a higher volume of Mato Grosso produced soybeans being sent north to ports in northern Brazil than the traditional ports in southern Brazil. Municipality of Sao Jose do Rio Claro- In the municipality of Sao Jose do Rio Claro, which is located in south-central Mato Grosso, many farmers planted their 2020/21 soybeans between mid-October and mid-November, which was 3-4 weeks later than normal. The region had below normal rainfall during October and almost no rain during November and early December. Some of the latest planted soybeans had poor emergence and delayed early growth due to dry conditions and farmers are expecting yield losses in the range of 15% to 20%. Especially hard hit were areas of pastures that were converted to soybeans for the first time. These new areas generally have lower fertility and lower organic matter making them more susceptible to adverse weather. Parana Soybeans- The recent rains in Parana have helped to stabilize the soybean crop in the state. The Department of Rural Economics (Deral) reported earlier last week that the soybean crop was 62% in vegetative development, 28% flowering, and 10% filling pods. The soybeans were rated 4% poor, 19% average, and 77% good. Deral estimates that the 2020/19 sovbean acreage increased 2% to 5.6 million hectares (13.8 million acres), but that the production will be down 2% compared to last year to 20.4 million tons. Dryer weather that delayed the early planting was cited as the reason for the lower production. Farmers in Parana have forward contracted 43% of their anticipated 2020/21 soybean production. The average price during December was R\$ 137.38 per sack (approximately \$11.78 per bushel) compared to December 2019 when the average price was R\$ 77.00 per sack (approximately \$6.60 per bushel). Rio Grande do Sul Soybeans-The soybeans in Rio Grande do Sul were 93% planted last week according to Emater. They are estimating that farmers in the state will plant 6.07 million hectares of soybeans (15.0 million acres). Recent rains have helped the development of the later planted soybeans.

A small portion of Brazil's soybean production is still devoted to the production of conventional soybeans, which are non-GMO.

Conventional Soybeans (non-GMO)

News

Brazil planted approximately 38.3 million hectares of soybeans in 2020/21 (94.6 million acres) with approximately 900,000 hectares devoted to conventional soybeans (2.2 million acres or 2.3% of Brazil's total soybean acreage). Mato Grosso is the main conventional soybean producing state in Brazil with 470,000 hectares (1.16 million acres).", 'The three main products of conventional soybeans are hipro soybean meal with 48% protein, SPC soybean meal with 62% protein, and lectin. The Brazilian Conventional Soybean Institute (ISL) estimates that there is an annual demand for 4 million tons for hipro soybean meal, 750,000 tons for SPC soybean meal, and 70,000 tons for lectin. The five main countries that purchase Brazilian hipro soybean meal made from conventional soybeans, are Germany, France, Italy, Holland, and Switzerland and it is used primarily for poultry rations, but also for hogs and cattle, The main purchasers of Brazilian SPC 62% soybean meal are Norway, Denmark, United Kingdom, Greece, and Australia and it is used mainly for fish rations especially salmon and trout. Lectin is used as an emulsifier for many food products especially chocolate and ice cream. Since commercial seed companies develop soybean varieties to be used in conjunction with their specific herbicides, it has been left to Embrapa to develop conventional soybean varieties. There are currently 17 different conventional soybean varieties adapted for production in the states of Mato Grosso and Goias. The state of Mato Grosso is the main producer of conventional soybeans in Brazil because conventional soybeans produced in the western part of the state can be exported out of the Port of Porto Velho, which is set up to prevent the mixture of conventional sovbeans with GMO soybeans. The Brazilian Conventional Soybean Institute (ISL) feels the future of conventional soybean production in Brazil will depend on demand from the European Union, which is the principal consumer of conventional soybean products.

Here are the two filtered datasets:

- Corn News: Headlines or articles mentioning "corn"
- **Soy News**: Headlines or articles mentioning "soy", "soybeans", etc.

You can now analyze them separately or match them with corn/soy sentiment crossover points. Let me know if you'd like to tag them with sentiment or match them to crossover windows.

40

You said:

give me the python code

ChatGPT said:

Here's the complete Python code to split your news dataset into two separate DataFrames—one for **corn-related** news and one for **soy-related** news:

Y Python Code

```
python
CopyEdit
import pandas as pd
# Load the dataset
df = pd.read csv("All News Corn Soybeans.csv")
# Fix the misspelled column name
df = df.rename(columns={"Healline": "Headline"})
# Convert headline and news text to lowercase for filtering
df["Headline lower"] = df["Headline"].str.lower()
df["News lower"] = df["News"].str.lower()
# Create boolean filters
corn filter = df["Headline lower"].str.contains("corn") |
df["News lower"].str.contains("corn")
soy filter = df["Headline lower"].str.contains("soy") |
df["News lower"].str.contains("soy")
# Create separate DataFrames
corn_news_df = df[corn_filter].drop(columns=["Headline_lower", "News_lower"])
soy news df = df[soy filter].drop(columns=["Headline lower", "News lower"])
# (Optional) Save them to CSV
corn news df.to csv("corn news.csv", index=False)
soy news df.to csv("soy news.csv", index=False)
```

This gives you two clean datasets:

- corn news df all entries related to corn
- soy news df all entries related to soy

Let me know if you'd like to filter more precisely (e.g., excluding ambiguous matches) or want to attach sentiment scores!

You said:

when i try to assign values from an existing dataframe to new one i get dataframe not defined error

ChatGPT said:

Ah, yes — that's a common issue and usually comes down to **scope** or **typos**. Here's a checklist to debug it quickly:

Common Causes of DataFrame not defined Error

♦ 1. Variable not declared yet

Make sure you've actually created the DataFrame before trying to use it.

```
python
CopyEdit
# Will raise error if 'df' is not defined earlier
new_df = df[df["column"] == "some value"]

V Fix:

python
CopyEdit
```

```
df = pd.read_csv("your_file.csv") # define it first
new_df = df[df["column"] == "some value"]
```

♦ 2. Typos / Inconsistent Names

Make sure the variable name is consistent:

```
python
CopyEdit
# X Incorrect
df = pd.read_csv("file.csv")
new_df = dataframe[dataframe["col"] > 0] # 'dataframe' not defined
# V Correct
df = pd.read_csv("file.csv")
new_df = df[df["col"] > 0]
```

♦ 3. Cell Order in Jupyter Notebooks

If you're using a Jupyter notebook:

- You must **run the cell that defines the DataFrame first**, or it won't exist in memory.
- Restarting the kernel clears all variables.

X Example Fix

If you wrote something like this:

```
python
CopyEdit
filtered_df = news_data[news_df["column"].str.contains("corn")]
```

✓ Make sure the news_df is defined before that line:

```
python
CopyEdit
news_df = pd.read_csv("news.csv")
filtered_df = news_df[news_df["column"].str.contains("corn")]
```

If you're still running into it, feel free to paste the **exact code snippet**, and I'll point out the issue directly.

40

You said:

I get Unknown datetime string format for the date values in my previous datasets

ChatGPT said:

Got it — you're getting an Unknown datetime string format error when trying to parse dates, most likely with pd.to_datetime().

This usually means that:

- Your Date column has inconsistent formats, like "2020-12-31", "12/31/2020", or "31-Dec-2020" mixed together.
- Or, it might contain missing or corrupted values.

✓ How to Fix It Safely

Here's a **robust way** to handle date parsing issues:

```
python
CopyEdit
import pandas as pd

# Let's assume your column is called 'Date'
df["Date"] = pd.to_datetime(df["Date"], errors="coerce",
infer datetime format=True)
```

What this does:

- errors="coerce": Converts unparseable values to NaT (missing datetime)
- infer datetime format=True: Tries to intelligently guess the format

Debug: See Which Dates Failed

You can then find rows where the date failed to parse:

```
python
CopyEdit
invalid dates = df[df["Date"].isna()]
print(invalid dates)
```

Optional: Clean and Reparse If Needed

Sometimes it's just a character or formatting issue:

```
python
CopyEdit
df["Date"] = df["Date"].str.strip() # Remove leading/trailing spaces
df["Date"] = pd.to datetime(df["Date"], errors="coerce")
```

Would you like me to run this fix on your news or price data and show you the cleaned version?

2/2

40

You said:

in a dataframe called corn_news_df with fields Date, Headline and News for each day I would like create a joined field wich aggregates previous n number of days Headline and News seperately

ChatGPT said:

Perfect — you're looking to create **rolling/joined aggregations** of past n days' worth of headlines and news articles, grouped by date. This is super useful for modeling how sentiment builds up before price or trend changes.



For each date:

- Joined Headline: headlines from previous n days
- Joined News: news bodies from previous n days

★ Code Example (Using n=3 Days)

```
python
CopyEdit
import pandas as pd
# Convert Date to datetime and sort
corn news df["Date"] = pd.to datetime(corn news df["Date"])
corn news df = corn news df.sort values("Date")
# Set the number of days to look back
n days = 3
# Group by date to avoid duplicate headlines on the same date
daily_grouped = corn_news_df.groupby("Date").agg({
    "Headline": lambda x: " | | | ".join(x),
    "News": lambda x: " ||| ".join(x)
}).reset index()
# Create rolling aggregations
daily grouped["Joined Headline"] =
daily grouped["Headline"].rolling(window=n days, min periods=1).apply(
    lambda x: " ||| ".join(x.dropna()), raw=False
)
daily grouped["Joined News"] = daily grouped["News"].rolling(window=n days,
min periods=1).apply(
    lambda x: " ||| ".join(x.dropna()), raw=False
# Final output
result df = daily grouped[["Date", "Joined Headline", "Joined News"]]
```

Explanation

- rolling (window=n_days) slides a window of n days
- We use .apply() to join text inside that window
- " | | | " is a separator feel free to change it

Would you like to do this with a fixed **lag** (e.g., headlines strictly *before* the current date), or just rolling including the current day?

You said:

the data i would like to aggregate is text data. I get No numeric types to aggregate error

ChatGPT said:

Ah, perfect catch — that error happens because pandas'.rolling() is designed for **numeric aggregation**, and by default, it doesn't work with text.

But no worries — we can still achieve exactly what you want using a more manual but flexible method:

Solution: Use a Custom Loop for Rolling Text Aggregation

```
python
CopyEdit
import pandas as pd
from datetime import timedelta
# Ensure proper datetime and sort
corn news df["Date"] = pd.to datetime(corn news df["Date"])
corn news df = corn news df.sort values("Date")
# Set how many days back to aggregate
n days = 3
# Create a clean daily version grouped by date
daily grouped = corn news df.groupby("Date").agg({
    "Headline": lambda x: " | | | ".join(x),
    "News": lambda x: " ||| ".join(x)
}).reset index()
# Create rolling text aggregation manually
joined rows = []
for i in range(len(daily grouped)):
    end_date = daily_grouped.loc[i, "Date"]
    start date = end date - timedelta(days=n days - 1)
    # Filter previous n days
    mask = (daily grouped["Date"] >= start date) & (daily grouped["Date"] <=</pre>
end date)
    window = daily grouped.loc[mask]
    joined headlines = " ||| ".join(window["Headline"].dropna())
    joined news = " ||| ".join(window["News"].dropna())
    joined rows.append({
        "Date": end date,
```

```
"Joined Headline": joined headlines,
        "Joined News": joined news
    })
# Final aggregated DataFrame
rolling text df = pd.DataFrame(joined rows)
```



What You Get

Each row contains:

- The **end date** (anchor point)
- Aggregated Joined Headline and Joined News for the previous n days

Want to add this rolling news context to your crossover signal analysis next?

40

You said:

I would like to outer join two dataframes based on the field Date. The fields to be included will be named Joined Headline if df1 and signal from df2

ChatGPT said:

Great — you're looking to outer join two DataFrames on the Date column and include specific fields:

- From df1: Date, Joined Headline
- From df2: Date, Signal

Code to Outer Join on Date

```
python
CopyEdit
# Select only relevant columns from each DataFrame
dfl_selected = dfl[["Date", "Joined_Headline"]]
df2 selected = df2[["Date", "Signal"]]
# Perform outer join on the 'Date' column
merged df = pd.merge(df1 selected, df2 selected, on="Date", how="outer")
# Optional: sort by date
merged df = merged df.sort values("Date").reset index(drop=True)
```

This will include:

- All dates from both df1 and df2
- Nan values where a match is missing
- A final DataFrame with:

```
Date, Joined Headline, Signal
```

Let me know if you also want to forward-fill missing Signal or headline values (e.g., for modeling).

40

You said:

how can i fully view first 20 lines of merged_df? save to csv maybe?

ChatGPT said:

Absolutely — saving to CSV is a great idea for full visibility, especially when working with long text fields like headlines.

Save First 20 Rows to CSV

```
python
CopyEdit
merged_df.head(20).to_csv("merged_first_20_rows.csv", index=False)
```

This will create a file called merged_first_20_rows.csv in your working directory with the first 20 rows.

Or: Print All Fields Fully in Notebook/Console

If you're not using CSV and just want to **see it in full** (e.g., in Jupyter or console), you can expand display settings like this:

```
python
CopyEdit
# Expand column width and display max rows
pd.set_option("display.max_colwidth", None)
pd.set_option("display.max_rows", 20)
print(merged_df.head(20))
```

Let me know if you'd like me to export and show the file directly here too.

You said:

sort dataframe using Date and ascending

ChatGPT said:

You can sort your DataFrame by the Date column in ascending order using:

```
python
CopyEdit
merged_df = merged_df.sort_values("Date",
ascending=True).reset index(drop=True)
```

What it does:

- Sorts the rows by Date in **oldest-to-newest** order
- Resets the index so it's clean and sequential

Let me know if you want to group or aggregate anything by time (like by week, month, etc.) after sorting!

40

You said:

I have a dataframe which contains date, signal and joined_headline fields. I'm going to train a llm for sentiment analysis. so should i drop dates with no text input

ChatGPT said:

Yes — if you're planning to train a **language model** (**LLM**) for sentiment analysis using your <code>joined_headline field</code> as the input, then:

You should drop rows where joined_headline is empty or missing (NaN), because:

- The model can't learn from **missing input**.
- These rows will contribute **no training signal**.
- Many ML pipelines will error out or treat empty strings as neutral by default (which may mislead the model).

X Code to Drop Rows Without Text Input

```
python
CopyEdit
# Drop rows where 'joined headline' is missing or empty
df = df.dropna(subset=["Joined Headline"])
df = df[df["Joined Headline"].str.strip() != ""]
```

This removes:

- NaN values
- Empty or whitespace-only strings



Make sure your remaining Signal labels (targets) are still balanced (e.g., enough positive/negative examples).

Would you like help preparing this cleaned data into a format ready for training (e.g., text and label columns for HuggingFace or PyTorch training)?

You said:

first of all how can i check if the data is balanced

ChatGPT said:

Great question — checking if your **sentiment label** (**signal**) **is balanced** is an important step before training.

Here's how you can quickly inspect the label distribution:

Check Balance of Sentiment Labels

```
python
CopyEdit
# Count occurrences of each label
label counts = df["Signal"].value counts(dropna=False)
print(label counts)
# Calculate proportions (percentages)
label distribution = df["Signal"].value counts(normalize=True) * 100
print(label distribution)
```



Example Output

```
diff
CopyEdit
1.0
      120
     115
-1.0
0.0
        80
NaN
        5
matlab
CopyEdit
1.0
       40.0%
-1.0 38.3%
 0.0
       26.7%
NaN
       1.7%
```



***** What to Watch For

- If one class (e.g., 1.0 or -1.0) dominates, the model may be biased toward predicting it.
- You can consider:
 - o **Downsampling** the majority class
 - o **Upsampling** the minority class
 - Using class weights during training

Let me know your actual label distribution, and I can help you decide if balancing is needed and how to do it.

You said:

show NaN values in a dataframe

ChatGPT said:

To view rows in your DataFrame that contain **NaN values** in any column:

Show All Rows with Any Nan

```
python
CopyEdit
nan_rows = df[df.isna().any(axis=1)]
print(nan rows)
```



Show Nan in Specific Column (e.g., Signal)

```
python
CopyEdit
nan signal rows = df[df["Signal"].isna()]
print(nan signal rows)
```

☆ Optional: Count Missing Values Per Column

```
python
CopyEdit
missing counts = df.isna().sum()
print(missing counts)
```

Let me know if you'd like to drop, fill, or handle them differently depending on the column!

40

You said:

I would like to create t+1 to t+5 fields for signal

ChatGPT said:

Great — you're looking to create **future signal fields** like:

```
• Signal t+1: the signal 1 day after t
• Signal t+2: the signal 2 days after t
• Signal t+5: 5 days after t
```

This is very useful for **supervised learning** when you want to predict upcoming sentiment/trend shifts based on current text.

Code to Create t+1 to t+5 Signal Columns

```
python
CopyEdit
# Create future signal columns
for i in range (1, 6):
    df[f"Signal t+{i}"] = df["Signal"].shift(-i)
```



- shift (-i) moves the Signal column **up** by i rows
- This means for row t, the new column will contain the signal value for t+i

Tip: Dropping Final Rows (Optional)

The last few rows will now have NaN in Signal t+5 (since there's no future data). You can optionally drop them if you want:

```
python
CopyEdit
df = df.dropna(subset=[f"Signal_t+{i}" for i in range(1, 6)])
```

Let me know if you'd like to compute a **combined target** like "Did any signal flip to 1 in next 5 days?" for classification!

You said:

I have a news feed. At certain days news arrive. I would like to make a new field which contains the previous n day's news joined in a single cell for every possible day. So a news will stay in the joined cell for n days then disappear

ChatGPT said:

Ah, got it — you're aiming for a **rolling news memory**: for every day (even if no news arrives), you want to collect the previous n days of news, where each news item "lives" for n days in the aggregated text, then drops off.

This is like simulating how news affects sentiment or market perception over time.

✓ What You Want:

For example, with n = 3, and news on these days:

- Jan 1: News A
- Jan 3: News B Then:
- Jan 1: News A
- Jan 2: News A
- Jan 3: News A + B
- Jan 4: News B
- Jan 5: (empty)

% Code to Build That (Step by Step)

Assumes your DataFrame is called news df with Date and News columns.

```
python
CopyEdit
import pandas as pd
from datetime import timedelta
# Parameters
n days = 3
# Ensure Date is datetime
news df["Date"] = pd.to datetime(news df["Date"])
news df = news df.sort values("Date")
# Create full date range from min to max date
date range = pd.date range(start=news df["Date"].min(),
end=news df["Date"].max())
# Initialize output list
expanded rows = []
# Loop through each date in range
for current date in date range:
    # Find all news from the previous n days (including today)
    start_date = current_date - timedelta(days=n_days - 1)
    mask = (news_df["Date"] >= start_date) & (news_df["Date"] <=</pre>
current date)
    window news = news df.loc[mask, "News"].dropna()
    # Join the news text
    joined news = " ||| ".join(window news)
    # Save the result
    expanded rows.append({
        "Date": current date,
        f"News_Last_{n_days}_Days": joined_news
    })
# Create the final DataFrame
news rolling df = pd.DataFrame(expanded rows)
```

Output

The new DataFrame news rolling df will contain:

- One row for each date
- A field like News Last 3 Days with joined news from the last n days

Let me know if you want to combine **headlines** + **news**, or align this with your **price/signal data**.

40

You said:

in a dataframe populate all missing dates given min and max, and use the last possible price for missing days

ChatGPT said:

Perfect — you're trying to **fill in missing dates** in a price DataFrame and **forward-fill** the last known price on those days.

Goal:

- Ensure one row for **every calendar day** between the min and max date.
- Use the **last known price** to fill in the missing days (forward fill).

★ Code to Do That:

```
python
CopyEdit
import pandas as pd

# Ensure Date is datetime
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values("Date")

# Create a full daily date range
full_range = pd.date_range(start=df["Date"].min(), end=df["Date"].max())

# Set Date as index and reindex
df = df.set_index("Date").reindex(full_range)

# Rename index back to 'Date'
df.index.name = "Date"

# Forward-fill missing values
df = df.ffill().reset_index()
```

What It Does:

- .reindex() ensures a row for every date
- .ffill() fills in missing price with the last known value
- .reset index() turns the Date index back into a column

Let me know if you want to **limit this to business days only**, or fill other fields like volume/signal as well.

You said:

in a dataframe which has date field for each day return the first different value in the signal field for the next n days

ChatGPT said:

Great — you're looking to scan ahead from each day and return the **first** *different* Signal value within the **next** n **days**.

This is perfect for forecasting or tracking trend reversals.



For each row:

- Look at the next n rows (days)
- Return the **first signal value that differs** from the current one
- If no different signal found in the next n days → return NaN

% Code to Do That:

```
result.append(found)
return result

# Example: find first different signal in next 5 days
df["First_Diff_Signal_Next_5"] = first_different_signal(df["Signal"], n=5)
```

Example

Date Signal First_Diff_Signal_Next_5 2024-01-01 1 -1 2024-01-02 1 -1 2024-01-03 1 -1 2024-01-04 -1 0 2024-01-05 -1 NaN

Let me know if you want the code to return the **number of days until the change**, or whether to include **0** as a valid change, or not.

You said:

if there is no change in the next n days return todays value

ChatGPT said:

Perfect — you want to modify the logic so that:

- If a **different** Signal is found in the next n days \rightarrow return that.
- If **no different** signal is found → return the **current signal**.

✓ Updated Code

```
# Apply to your DataFrame
df["First Diff Signal Next 5"] = first different or self(df["Signal"], n=5)
```



You'll get:

- The **first different signal** in the next n rows
- Or the **current value** if all next n values are the same

Let me know if you'd like to also return the **number of days until that change** or a **binary flag** indicating if a change occurred.

40

You said:

as series i want to use a field from a dataframe

ChatGPT said:

Absolutely! If you're using a column from a DataFrame as the input series, you just need to pass it like this:

Example with a DataFrame Column

```
python
CopyEdit
# Let's say your DataFrame is called df and the signal column is "Signal"
df["First Diff Signal Next 5"] = first different or self(df["Signal"], n=5)
```

This will apply the function to the Signal column of df and create a new column called First Diff Signal Next 5.

Just to Recap: Full Code in Context

```
python
CopyEdit
def first different or self(series, n):
```

```
result = []
signals = series.values
for i in range(len(signals)):
    current = signals[i]
    found = current # default: use current signal if no change found
    for j in range(1, n + 1):
        if i + j < len(signals) and signals[i + j] != current:
            found = signals[i + j]
            break
    result.append(found)
return result

# Apply it to a column in a DataFrame
df["First Diff Signal Next 5"] = first different or self(df["Signal"], n=5)</pre>
```

Let me know if you'd like to adapt this for multiple signal columns or for prediction window ranges like t+3 to t+7.

You said:

i have a dataframe with news data as text and labels how can i seperarte the data for train and test

ChatGPT said:

Great! You're ready to train your model — and to do that properly, you'll want to **split your labeled dataset** into **training and testing sets**.

✓ Step-by-Step: Train/Test Split for Text Classification

Assume your DataFrame (df) has:

- A text column (e.g., text or Joined Headline)
- A label column (e.g., Signal)

★ Code Using train_test_split

```
test size=0.2,
    random state=42,
    stratify=y # Optional: maintain class balance in both sets
)
```



Optional: Reconstruct into DataFrames (if needed)

```
python
CopyEdit
train df = pd.DataFrame({"text": X train, "label": y train})
test df = pd.DataFrame({"text": X test, "label": y test})
```

Tips

- Use stratify=y if your classes (signals) are imbalanced helps preserve the label distribution.
- You can later save these to CSV:

```
python
CopyEdit
train df.to csv("train data.csv", index=False)
test df.to csv("test data.csv", index=False)
```

Would you like help preparing this for a specific model format (like HuggingFace datasets, PyTorch, or TensorFlow)?

40

You said:

my data is dated data. does that make any difference

ChatGPT said:

Yes — excellent point!

Since your data is **dated**, it absolutely **does** make a difference in how you split it.

Why You Shouldn't Use Random Split with Time-Series/Text Data:

- Random splitting (like train test split) leaks future information into the past.
- For example, your model could see a headline from January 10 and be tested on a headline from January 5 — that's **not realistic** in production.

✓ Instead: Use a Chronological Split

You want to **train on earlier data** and **test on later data** — just like the real world.

★ Code for Chronological (Time-Based) Split

```
python
CopyEdit
# Make sure your Date column is in datetime format
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values("Date") # ensure chronological order
# Set ratio (e.g., 80% train, 20% test)
split_idx = int(len(df) * 0.8)
# Split chronologically
train_df = df.iloc[:split_idx]
test_df = df.iloc[split_idx:]
```

Result:

- train df = older data
- test_df = newer, unseen data simulates how the model will perform on future headlines

Let me know if you want to apply a **rolling training window** or use a **specific cutoff date** instead of a percentage split!

You said:

now i can train my llm. for corn news sentiment analysis which llm may i use

ChatGPT said:

Awesome — you're at the training stage! For **corn news sentiment analysis**, you'll want a language model that's:

- Good with short texts like headlines
- Ideally has some financial/agriculture/news domain exposure
- Supports custom fine-tuning

✓ Best LLMs to Use for Fine-Tuning Sentiment on Corn News

♦ 1. FinBERT (Highly Recommended)

- Pretrained on **financial news**
- Great for economic/agricultural sentiment
- Supports positive, neutral, negative sentiment out of the box
- Easy to fine-tune on custom labels (like your Signal)
- Model on Hugging Face: ProsusAI/finbert

♦ 2. BERT-base or DistilBERT

- Generic but lightweight and fast
- If you're working with **custom sentiment labels**, this is a solid base model
- Models:
 - bert-base-uncased
 - distilbert-base-uncased

♦ 3. RoBERTa (Finance/RoBERTa)

- Very strong language understanding
- Try yiyanghkust/finbert-tone for financial tone classification

★ What You'll Need to Do

1. **Format your dataset** as:

```
json
CopyEdit
{ "text": "...joined headline...", "label": ...signal value... }
Or as a CSV with text, label.
```

- 2. Use Hugging Face Transformers and Trainer API or PyTorch to fine-tune.
- 3. Choose label mapping:

```
o 1 = positive
o -1 = negative
o 0 = neutral
```

Bonus Options

If you're open to larger LLMs and want zero/few-shot classification without fine-tuning:

- OpenAI GPT-4 / Claude with prompt-based labeling
- Mistral / Mixtral / LLaMA 2/3 with adapters (e.g., LoRA) for small-scale fine-tuning

Would you like help building the training script with Hugging Face and FinBERT? I can scaffold it for you.

You said:

how may i start with finbert

ChatGPT said:

Great choice! Let's walk you through **starting with FinBERT** step by step — from environment setup to fine-tuning on your corn news dataset.

Step-by-Step: Train Your Own Sentiment Classifier with FinBERT



bash
CopyEdit
pip install transformers datasets scikit-learn pandas

📄 2. Load FinBERT Pretrained Model

```
python
CopyEdit
from transformers import AutoTokenizer, AutoModelForSequenceClassification
model_name = "ProsusAI/finbert"
tokenizer = AutoTokenizer.from pretrained(model name)
```

model = AutoModelForSequenceClassification.from pretrained(model name, num labels=3)



📦 3. Prepare Your Dataset

Assuming you have a DataFrame train df with:

- text column (e.g., joined headlines)
- label column (your Signal: -1, 0, 1)

```
python
CopyEdit
from datasets import Dataset
# Hugging Face expects labels as integers starting from 0
label map = \{-1: 0, 0: 1, 1: 2\}
train_df["label"] = train_df["label"].map(label_map)
test df["label"] = test df["label"].map(label map)
# Create Hugging Face Dataset
train dataset = Dataset.from pandas(train df[["text", "label"]])
test dataset = Dataset.from pandas(test df[["text", "label"]])
```

4. Tokenize the Text

```
python
CopyEdit
def tokenize(batch):
   return tokenizer(batch["text"], padding=True, truncation=True)
train dataset = train dataset.map(tokenize, batched=True)
test dataset = test dataset.map(tokenize, batched=True)
```

5. Train the Model

```
python
CopyEdit
from transformers import Trainer, TrainingArguments
training args = TrainingArguments(
    output dir="./finbert-corn-model",
    evaluation_strategy="epoch",
    save strategy="epoch",
    num train epochs=4,
    per device train batch size=8,
    per device eval batch size=8,
    learning rate=2e-5,
    logging dir='./logs',
)
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)
trainer.train()
```

6. Evaluate or Save Model

```
python
CopyEdit
# Evaluate
trainer.evaluate()

# Save
trainer.save_model("./finbert-corn-model")
tokenizer.save pretrained("./finbert-corn-model")
```

7. Label Mapping (for Inference)

You'll need this to decode predictions:

```
python
CopyEdit
reverse map = {0: -1, 1: 0, 2: 1}
```

Want Help?

If you upload your preprocessed train_df and test_df, I can help prepare this code with your real label mapping and structure. Or I can convert your Signal format into a Hugging Face dataset for you.

Would you like that?

40

You said:

how do i import pytorch in notebook

ChatGPT said:

To import **PyTorch** in a Jupyter Notebook, just run this line:

python
CopyEdit
import torch

If it's not installed yet, install it first:

✓ Install PyTorch (CPU-only version, works in most notebooks)

bash
CopyEdit
pip install torch

If you're using a GPU and want GPU acceleration:

• For **CUDA 11.8**:

```
bash
CopyEdit
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118
```

You can check if it's installed and using GPU with:

```
python
CopyEdit
import torch
print(torch.__version__)
print("GPU Available:", torch.cuda.is_available())
```

Let me know if you're using Google Colab, Jupyter Lab, or local Jupyter Notebook and I'll tailor the steps!

You said:

how can i install this with conda: pip install transformers[torch]

ChatGPT said:

Great! If you're using **conda**, it's best to install transformers and PyTorch in a compatible way to avoid version conflicts.

- ▼ Step-by-Step: Install transformers with PyTorch via conda
- **1. ♦** Create a fresh environment (optional but recommended)

```
bash
CopyEdit
conda create -n corn-llm python=3.10 -y
conda activate corn-llm
```

2. Install PyTorch (CPU version)

Go to https://pytorch.org/get-started/locally/ for custom CUDA options.

Basic CPU version:

```
bash
CopyEdit
conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

Or for **CUDA 11.8** (**GPU**):

```
bash
CopyEdit
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c
```

3. Install transformers using pip

```
bash
CopyEdit
pip install transformers
```

pip install transformers [torch] just ensures PyTorch is installed if you don't have it, but you already installed PyTorch manually via conda, which is safer.

Test It Works

```
python
CopyEdit
import torch
import transformers
print(torch.__version__)
print("GPU:", torch.cuda.is_available())
```

Let me know if you'd like to install datasets, accelerate, or other libraries for training too.

40

You said:

how about installing torch for apple silicon

ChatGPT said:

Great question — if you're on **Apple Silicon** (M1/M2/M3), PyTorch has official support for macOS + ARM64, including GPU acceleration via Metal backend.

Here's how to install it properly using **conda** or **pip**.



Option 1: Install PyTorch with conda (Recommended)

bash CopyEdit conda create -n torch-mac python=3.10 -y conda activate torch-mac # Install PyTorch for Apple Silicon (ARM64) conda install pytorch torchvision torchaudio -c pytorch

This installs the ARM64-native version — optimized for your chip.



Option 2: Install with pip (Universal Wheels)

Since PyTorch 1.12+, pip wheels support Apple Silicon natively:

bash CopyEdit pip install torch torchvision torchaudio

To ensure it uses the Metal backend:

python CopyEdit import torch torch.backends.mps.is available()

If it returns True, you're using the **Metal GPU backend** on Apple Silicon





Optional: Test MPS Backend

python CopyEdit import torch

```
print("Torch version:", torch. version )
print("MPS available:", torch.backends.mps.is available())
x = \text{torch.rand}(3, 3).\text{to}("mps") \# \text{ send tensor to Apple GPU}
print(x)
```



Tips

- Works best with **Python 3.9 or 3.10**
- You don't need CUDA; Apple doesn't support it MPS (Metal Performance Shaders) is used instead
- Works well with transformers, datasets, etc.

Let me know if you'd like a full Apple Silicon-friendly conda environment file for LLM finetuning!

40

You said:

do i have train the finbert each time i restart the kernel

ChatGPT said:

Great question — and no, you don't have to retrain FinBERT every time you restart the kernel, as long as you save your trained model after training.



What to Do After Training



♦ 1. Save your trained model

After fine-tuning FinBERT:

```
python
CopyEdit
trainer.save model("./finbert-corn-sentiment")
tokenizer.save pretrained("./finbert-corn-sentiment")
```

This will save:

- Model weights
- Config

Tokenizer



What to Do After Kernel Restart

You can **reload your saved model** like this:

```
python
CopyEdit
from transformers import AutoTokenizer, AutoModelForSequenceClassification
model path = "./finbert-corn-sentiment"
tokenizer = AutoTokenizer.from pretrained(model path)
model = AutoModelForSequenceClassification.from pretrained(model path)
```

You can now:

- Run inference (e.g., predict new headlines)
- Continue training from this checkpoint
- Evaluate on test data

Tip: Hugging Face Format = Portable

The saved folder can even be uploaded to Hugging Face Hub or reused in other projects or servers.

Let me know if you want a helper function for **inference** using your saved FinBERT model!

40

You said:

can you help me interpret long term short term moving average crosovers and device an automatic trading strategy

ChatGPT said:

Absolutely — you're getting into the really fun part now: turning moving average crossovers into an automated trading strategy. Let's break it down step by step:

✓ 1. Understanding MA Crossovers

- Short-Term MA (e.g., 7-day)
 - Reacts quickly to price changes
 - Shows recent market sentiment
- ♦ Long-Term MA (e.g., 30-day)
 - Smooths out noise
 - Represents overall trend

Crossover Logic

Event Description Trading Signal
Bullish Crossover 7-day MA crosses above 30-day MA Buy / Go Long
Bearish Crossover 7-day MA crosses below 30-day MA Sell / Exit / Go Short

2. Automated Trading Strategy (Rule-Based)

- **✓** Strategy Rules:
 - 1. **Buy Signal** (enter position):
 - If 7-day MA crosses above 30-day MA
 - No active position
 - 2. **Sell Signal** (exit position):
 - o If 7-day MA crosses below 30-day MA
 - You have an open long position

You can optionally:

- Only trade if volume confirms the move
- Combine with sentiment or fundamentals

*** 3. Python Code for Strategy Backtest Logic**

Assumes your DataFrame of has Price, MA 7, and MA 30.

```
python
CopyEdit
df["Position"] = 0 # 1 = Long, 0 = Neutral
df["Signal"] = 0  # Buy/Sell signals
# Generate crossover signals
df.loc[(df["MA 7"] > df["MA 30"]) & (df["MA 7"].shift(1) <=
df["MA 30"].shift(1)), "Signal"] = 1 # Buy
df.loc[(df["MA 7"] < df["MA 30"]) & (df["MA 7"].shift(1) >=
df["MA 30"].shift(1)), "Signal"] = -1 # Sell
# Forward-fill position based on signals
position = 0
positions = []
for signal in df["Signal"]:
    if signal == 1:
       position = 1 # Enter long
    elif signal == -1:
       position = 0 # Exit
   positions.append(position)
df["Position"] = positions
```



4. Backtest: Calculate Returns

```
python
CopyEdit
# Daily returns
df["Return"] = df["Price"].pct change()
# Strategy return (only when in position)
df["Strategy Return"] = df["Return"] * df["Position"]
# Cumulative returns
df["Cumulative Strategy"] = (1 + df["Strategy Return"]).cumprod()
df["Cumulative BuyHold"] = (1 + df["Return"]).cumprod()
```

5. Plot Performance

```
python
CopyEdit
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(df["Date"], df["Cumulative Strategy"], label="Strategy Return")
plt.plot(df["Date"], df["Cumulative BuyHold"], label="Buy & Hold")
plt.legend()
plt.title("Cumulative Returns")
plt.grid(True)
plt.show()
```



- Add sentiment filter: Only trade on crossover + **positive sentiment**
- Add stop-loss or take-profit rules
- Use exponential moving averages (EMA) for faster reaction

Would you like me to help you build this out with your real price data and see how it would have performed?

40

You said:

what does epoc number training loss and validation loss mean, how can i interpret it

ChatGPT said:

Great question — these are **core concepts** in model training, especially when you're fine-tuning models like FinBERT.

Let's break it all down simply:



What Do These Terms Mean?



- One full pass through the entire training dataset
- If you train for 4 epochs, the model sees the training data 4 times (shuffled each time)

Training Loss

- How well the model is doing on the training data
- Calculated **after each batch**, then averaged over the epoch
- Should **decrease** over time
- Lower = better fit (on training data)

Validation Loss

- How well the model performs on unseen validation data
- Calculated after each epoch
- Used to detect overfitting

Ideally:

- Validation loss **goes down** with training loss (model is learning)
- If validation loss starts going up while training loss continues down \rightarrow overfitting

✓ How to Interpret Training & Validation Loss Together

Training Loss Validation Loss

Interpretation

\downarrow	\downarrow	Learning well, not overfitting
\downarrow	\rightarrow / flat	Maybe underfitting or too small model
\downarrow	↑	⚠ Overfitting (memorizing training data)

Example (Imaginary)

Epoch Train Loss Val Loss

1	0.95	0.90
2	0.75	0.72
3	0.62	0.65
4	0.55	0.70

Model starts overfitting after epoch 2–3. You might stop early or regularize.

Tips

- Use early stopping: Stop training when validation loss stops improving
- Track accuracy or F1-score too, not just loss
- Save the model from the epoch with **lowest validation loss**

Would you like help adding early stopping or plotting loss curves after training?

You said:

after training the model how may i evaluate the performance of the model using performance metrics

ChatGPT said:

Great follow-up — after training your FinBERT model (or any classifier), evaluating it with **performance metrics** is key to knowing how well it works on **unseen data**.

Let's walk through how to evaluate the model quantitatively.



✓ Step-by-Step: Evaluate a Trained Text Classifier

Prepare Your Test Data

Assuming:

- You have a test set (test df) with text and label
- You've already loaded the trained model and tokenizer

☑Tokenize Test Data

```
python
CopyEdit
from datasets import Dataset
# Create Hugging Face dataset
test dataset = Dataset.from pandas(test df)
test dataset = test dataset.map(lambda x: tokenizer(x['text'],
truncation=True, padding=True), batched=True)
```

⊠Run Inference

```
python
CopyEdit
import torch
from torch.utils.data import DataLoader
from transformers import DataCollatorWithPadding
```

```
# DataLoader
data collator = DataCollatorWithPadding(tokenizer=tokenizer)
test loader = DataLoader(test dataset, batch size=8,
collate fn=data collator)
# Collect predictions
all preds = []
all labels = []
model.eval()
with torch.no grad():
    for batch in test loader:
        input ids = batch['input ids']
        attention_mask = batch['attention_mask']
        labels = batch['label']
        outputs = model(input ids=input ids, attention mask=attention mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1)
        all preds.extend(preds.cpu().numpy())
        all labels.extend(labels.cpu().numpy())
```

■ Calculate Metrics

```
python
CopyEdit
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# If you used label_map earlier, reverse it for human-readable labels
reverse_label_map = {0: -1, 1: 0, 2: 1}
all_preds_mapped = [reverse_label_map[p] for p in all_preds]
all_labels_mapped = [reverse_label_map[l] for l in all_labels]

# Metrics
print("Accuracy:", accuracy_score(all_labels_mapped, all_preds_mapped))
print("\nClassification_report(all_labels_mapped, all_preds_mapped, digits=3))
print("\nConfusion_Matrix:")
print(confusion_matrix(all_labels_mapped, all_preds_mapped))
```

Metrics Explained

Metric What it Means
Accuracy % of total correct predictions

Precision % of positive predictions that were correct

Recall % of actual positives that were predicted correctly

Metric

What it Means

F1 Score Harmonic average of precision & recall

Confusion Matrix Shows how often labels were confused

Example Output (3-class case: -1, 0, 1)

markdown CopyEdit

	precision	recall	f1-score	support
-1	0.78	0.72	0.75	100
0	0.69	0.81	0.75	110
1	0.84	0.80	0.82	120

Let me know if you want help generating ROC curves or visualizing confusion matrices!

40

You said:

can you revise this for apple silicon mps

ChatGPT said:

Absolutely! To run your FinBERT evaluation (or any PyTorch inference) on Apple Silicon (M1/M2/M3) using the MPS (Metal Performance Shaders) backend, you just need to make a few small changes.

Here's how to revise the evaluation loop for MPS:



Revised Code for Apple Silicon (MPS)

```
python
CopyEdit
import torch
from torch.utils.data import DataLoader
from transformers import DataCollatorWithPadding
# Use MPS if available
device = torch.device("mps" if torch.backends.mps.is available() else "cpu")
model.to(device)
# DataLoader
data collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```
test loader = DataLoader(test dataset, batch size=8,
collate fn=data collator)
# Collect predictions
all preds = []
all labels = []
model.eval()
with torch.no grad():
   for batch in test loader:
        input ids = batch['input ids'].to(device)
        attention mask = batch['attention mask'].to(device)
        labels = batch['label'].to(device)
        outputs = model(input ids=input ids, attention mask=attention mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1)
        all preds.extend(preds.cpu().numpy())
        all labels.extend(labels.cpu().numpy())
```

Wey Differences

Line

What Changed

✓ Check It's Using MPS

Add this near the top of your notebook/script:

```
python
CopyEdit
print("Using device:", device)
print("MPS available:", torch.backends.mps.is_available())
```

Let me know if you'd also like to train or fine-tune your model on MPS (it works — though slower than CUDA GPUs).

40

on run inference step i get this error: Unable to create tensor, you should probably activate truncation and/or padding with 'padding=True' 'truncation=True'

ChatGPT said:

Ah yes — that error means your model is receiving text inputs of variable or excessive length, which PyTorch can't stack into a tensor automatically. The fix is exactly what the error suggests: make sure truncation and padding are enabled when tokenizing your text.



Fix: Add truncation=True and padding=True When Tokenizing

Update your tokenizer call like this:

```
python
CopyEdit
def tokenize function (example):
  return tokenizer(
     example["text"],
     # Apply to dataset
test dataset = test dataset.map(tokenize function, batched=True)
```

Why This Works

- padding=True: pads all examples in a batch to the same length
- truncation=True: cuts off long texts that exceed the model's max input size (usually 512 tokens for BERT models)

Optional: Add max length=512

Even with truncation, you can explicitly define the token limit:

```
python
CopyEdit
truncation=True,
max length=512
```

This ensures compatibility with BERT/FinBERT's input size.

After doing this, your model should be able to process all test samples cleanly without tensor creation errors.

Let me know if you want to add a warning system for overly long texts or visualize the input lengths!

40

You said:

it also says could not infer dtype of timestampt

ChatGPT said:

Ah, that's another common hiccup — it means your dataset (likely the test_dataset) contains a datetime column (like Date) and you're passing it into the tokenizer, which expects only strings or numbers.

▼ Fix: Remove Non-Text Columns Before Tokenizing

When you create the Hugging Face Dataset, make sure to include **only** the relevant columns (text, label). Exclude datetime or metadata fields.

♦ Step 1: Trim the DataFrame before conversion

```
python
CopyEdit
from datasets import Dataset

# Just keep text and label for model input
test_df = test_df[["text", "label"]] # or whatever your columns are called
test_dataset = Dataset.from_pandas(test_df)
```

♦ Step 2: Tokenize

```
python
CopyEdit
def tokenize_function(example):
    return tokenizer(
        example["text"],
        padding=True,
        truncation=True,
        max_length=512
)

test_dataset = test_dataset.map(tokenize_function, batched=True)
```

Why This Happens

- transformers tokenizer tries to process all fields in the dataset
- If one of them is a datetime (like a pd. Timestamp), it raises a dtype error, since it doesn't know how to tokenize it

Once you limit your dataset to only "text" and "label", and run proper tokenization, the error will disappear.

Want me to help validate your test df structure before creating the dataset?

40

You said:

this gets error: test dataset = test dataset[["merged news", "first diff"]]

ChatGPT said:

Ah — I see! The error you're getting is likely because you're treating a **Hugging Face Dataset** object like a **Pandas DataFrame**, but the slicing syntax you're using

(test_dataset[["merged_news", "first_diff"]]) only works for Pandas, not for datasets.Dataset.

Fix: Apply column selection before converting to Hugging Face dataset

♦ Step 1: Filter the Pandas DataFrame first

```
python
CopyEdit
# Assuming your original test data is a Pandas DataFrame
test df = test df[["merged news", "first diff"]] # <- This is valid here</pre>
```

♦ Step 2: Rename columns for tokenizer compatibility (optional but helpful)

```
python
CopyEdit
test_df = test_df.rename(columns={"merged_news": "text", "first_diff":
"label"})
```

Step 3: Convert to Hugging Face Dataset

```
python
CopyEdit
from datasets import Dataset
test_dataset = Dataset.from_pandas(test_df)
```

Reminder

Once it's a datasets. Dataset, you can no longer slice columns like this:

```
python
CopyEdit
# X This is invalid for Hugging Face datasets:
test_dataset[["text", "label"]]
# V Use .remove_columns if needed:
test_dataset = test_dataset.remove_columns(["Date", "OtherStuff"])
```

Let me know if you want to preview the cleaned dataset or tokenize it from here!

You said:

I use the following command: test_dataset = Dataset.from_pandas(test_df[["merged_news", "label"]]) however it seems date field is leaking to hugging face dataset

ChatGPT said:

Ah, good catch — that happens because pandas sometimes carries extra metadata (like the index) even when you slice only specific columns. When you call:

```
python
CopyEdit
test_dataset = Dataset.from_pandas(test_df[["merged_news", "label"]])
```

Hugging Face Dataset.from_pandas() automatically includes the DataFrame index as a column. unless told otherwise.

Fix: Prevent Index from Leaking Into the Dataset

Add preserve index=False like this:

```
python
CopyEdit
from datasets import Dataset

# Rename for clarity if needed
test_df = test_df.rename(columns={"merged_news": "text"})

# Convert to Hugging Face dataset without the index
test_dataset = Dataset.from_pandas(test_df[["text", "label"]],
preserve index=False)
```

Why This Happens

Hugging Face tries to preserve everything by default — including the index, which may be Date or Timestamp. This leads to the dtype error later when tokenizing.

Let me know if you'd like to inspect the columns of the resulting dataset to confirm it's clean before moving to tokenization!

40

You said:

it says the merged_news field in my dataset which corresponds to text field is in list format however int is expected

ChatGPT said:

Ah, this is a common issue — if you're seeing:

```
TypeError: Text values should be strings, but got list instead of int
```

It means that your merged news (now text) column contains lists of strings, rather than plain strings.

✓ What You Need

For tokenization to work, each row's text field should be a **single string**, like:

```
python
CopyEdit
"This is a merged news headline from the past few days."
```

What's Happening Now

Your merged news column likely looks like this:

```
python
CopyEdit
[['News A', 'News B', 'News C'], ['News D', 'News E'], ...]
```

So when the tokenizer tries to encode it, it fails — because it's expecting a string, not a list.

★ Fix: Join Each List into a Single String

Before converting to a Hugging Face dataset:

```
python
CopyEdit
# If each row is a list of strings, join them
test_df["merged_news"] = test_df["merged_news"].apply(
    lambda x: " ||| ".join(x) if isinstance(x, list) else str(x)
)
# Then rename and convert safely
test_df = test_df.rename(columns={"merged_news": "text"})
from datasets import Dataset
test_dataset = Dataset.from_pandas(test_df[["text", "label"]],
preserve_index=False)
```

Tip:

Use a separator like " | | | " to preserve the distinction between separate news items, or just a space " " if you're doing sentence-level modeling.

Let me know if you'd like to visualize a sample of the cleaned text input or check token lengths before fine-tuning!

40

You said:

how about this: You should supply an encoding or a list of encodings to this method that includes input_ids, but you provided ['text', 'label']

ChatGPT said:

Great — that error message is very specific and usually occurs during **inference or evaluation** with a Hugging Face model:

"You should supply an encoding or a list of encodings to this method that includes input ids, but you provided ['text', 'label']"

What's Going Wrong?

You're likely passing a Dataset with raw columns like 'text' and 'label' directly into the model, without tokenizing it first.

The model expects:

- input ids
- attention_mask
 (and optionally labels for supervised evaluation)

✓ Fix: Tokenize Your Dataset First

You must run the tokenizer over the dataset before sending it to a DataLoader or into the model:

```
python
CopyEdit
def tokenize_function(example):
    return tokenizer(
        example["text"],
        padding=True,
        truncation=True,
        max_length=512
    )

# Tokenize it
tokenized_test_dataset = test_dataset.map(tokenize_function, batched=True)
```

After Tokenization

Now your dataset will contain:

- input ids
- attention mask
- label (if provided in original)

These are exactly what the model needs.

✓ Now You Can Run Inference or Evaluation

```
python
CopyEdit
from torch.utils.data import DataLoader
from transformers import DataCollatorWithPadding
```

```
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
test_loader = DataLoader(tokenized_test_dataset, batch_size=8,
collate_fn=data_collator)
```

Let me know if you'd like help doing predictions or evaluating metrics after this!